THE DESIGN OF FINITE-STATE MACHINES FOR QUANTIZATION USING SIMULATED ANNEALING

A THESIS
SUBMITTED TO THE DEPARTMENT OF ELECTRICAL
AND ELECTRONICS ENGINEERING
AND THE INSTITUTE OF ENGINEERING AND SCIENCES
OF BILKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

By
Ercan Engin Kuruoğlu
August 1993

THE DESIGN OF FINITE-STATE MACHINES FOR QUANTIZATION USING SIMULATED ANNEALING

A THESIS

SUBMITTED TO THE DEPARTMENT OF ELECTRICAL AND

ELECTRONICS ENGINEERING

AND THE INSTITUTE OF ENGINEERING AND SCIENCES

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

Ercan Engin Kuruoğlu

By

Ercan Engin Kuruoğlu

August 1993

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

[illegible]

Assoc. Prof. Dr. Ender Ayanoğlu (Principal Advisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

[illegible]

Assoc. Prof. Dr. Erdal Arıkan

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

[illegible]

Assoc. Prof. Dr. Enis Çetin

Approved for the Institute of Engineering and Sciences:

[illegible]

Prof. Dr. Mehmet Baray
Director of Institute of Engineering and Sciences

ABSTRACT

THE DESIGN OF FINITE-STATE MACHINES FOR QUANTIZATION USING SIMULATED ANNEALING

Ercan Engin Kuruoğlu
M.S. in Electrical and Electronics Engineering
Supervisor: Assoc. Prof. Dr. Ender Ayanoğlu
August 1993

In this thesis, the combinatorial optimization algorithm known as *simulated annealing* (SA) is applied to the solution of the next-state map design problem of data compression systems based on finite-state machine decoders. These data compression systems which include *finite-state vector quantization* (FSVQ), *trellis waveform coding* (TWC), *predictive trellis waveform coding* (PTWC), and *trellis coded quantization* (TCQ) are studied in depth. Incorporating *generalized Lloyd algorithm* for the optimization of output map to SA, a finite-state machine decoder design algorithm for the joint optimization of output map and next-state map is constructed. Simulation results on several discrete-time sources for FSVQ, TWC and PTWC show that decoders with higher performance are obtained by the SA+GLA algorithm, when compared to other related work in the literature. In TCQ, simulation results are obtained for sources with memory and new observations are made.

Keywords : data compression, finite-state vector quantization, trellis waveform coding, predictive trellis waveform coding, trellis coded quantization, simulated annealing, finite-state machine decoders.

ÖZET

TAVLAMA BENZETİMİ KULLANARAK NİCEMLEME AMAÇLI SONLU DURUM MAKİNELERİ TASARIMI

Ercan Engin Kuruoğlu
Elektrik ve Elektronik Mühendisliği Yüksek Lisans
Tez Yöneticisi: Doç. Dr. Ender Ayanoğlu
Ağustos 1993

Bu çalışmada, sonlu durum makinelerine dayanan bazı veri sıkıştırma dizgelerinde eniyiye yakın kodçözücü tasarımı sorununa bir çözüm önerisi irdelenmiştir. Tezin bu konudaki araştırmalara temel katkısı, kodçözücü durum değiştirme tablosu tasarımında *tavlama benzetimi* olarak bilinen katışımsal eniyileştirme algoritmasının kullanılmasıdır. Çıktı tablosunun eniyileştirilmesinde kullanılan genelleştirilmiş Lloyd algoritması da tavlama benzetimi ile birlikte çalıştırılarak çıktı tablosu ve durum değiştirme tablosunu beraber eniyileştiren bir tasarım algoritması oluşturulmuştur. Sonlu durum vektör nicemleyicisi, çit kaynak kodlaması ve öngörülü çit kaynak kodlaması için elde edilen benzetim sonuçları önerilen algoritma ile daha önce yayınlanmış çalışmalara göre daha yüksek başarımlı kodçözücülerin tasarlandığını göstermektedir. Çit kodlamalı nicemleme için de yeni gözlemlerde bulunulmuştur.

Anahtar sözcükler : veri sıkıştırma, sonlu durum vektör nicemleyicisi, çit kaynak kodlaması, öngörülü çit kaynak kodlaması, çit kodlamalı nicemleme, tavlama benzetimi, sonlu durum makineli kodçözücü.

ACKNOWLEDGEMENT

I would like to thank Dr. Ender Ayanoğlu for his supervision, guidance and suggestions during the development of this thesis. I would like to thank Dr. Erdal Arıkan for many invaluable discussions. I would also like to thank him and Dr. Enis Çetin for reading and commenting on the thesis.

My special thanks go to Sarah (Clay) for her love, patience, understanding and encouragement especially at times of hardship and despair. Of course, I cannot pass without mentioning my parents and my sister here, whose very existence gave me so much courage.

I would like to extend my thanks to Dr. İlknur Özgen and Dr. Varol Akman for their support and closeness. And the last but by no means the least, my sincerest feelings are towards all of my friends for the times we had.

Contents

List of Figures

List of Tables

Chapter 1

INTRODUCTION

The goal of design of any communication system is to build a system which enables the transfer of information-bearing signals from the transmitter to the receiver reliably, that is with "little" or no loss in information. To reach the goal of reliable communication, the information to be transmitted should be converted into a form that is in some way more "convenient" to transmit and then be converted back to the original form after transmission. Let the information source be in the form of a random process X_n. Then, a simple, yet general model of a communication system aiming a reliable transfer of information is as given in Figure 1.1.

A communication system is composed of three parts, an encoder, the channel, and a decoder. The channel represents the medium through which information will be sent. The information which is represented by the input random process X_n is converted by the encoder into another random process U_n which is more convenient to handle and to transmit over the channel. This process U_n is sent through the channel, and at the other end of the channel another random process $\hat{U}_n$ is received by the receiver which is related to U_n through a conditional probability distribution. Then, the decoder performs the reverse

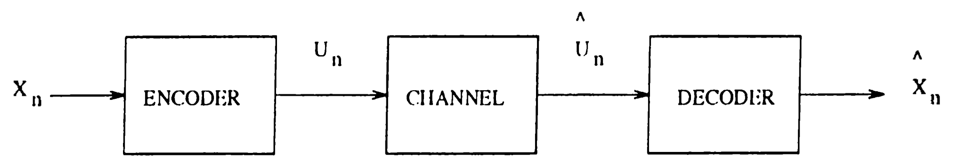


Figure 1.1: Communication system

operation of the encoder, and converts $\hat{U}_n$ into $\hat{X}_n$, the reproduction symbols. It is possible (and may actually be desirable) that $\hat{X}_n \neq X_n$ and the objective of communication system design is to minimize the difference between X_n, the input sequence, and $\hat{X}_n$, the reproduction sequence.

Usually the channel puts severe restrictions on the type and the quantity of signals that can be transmitted through it. For example, most of the time one has to represent an infinite collection of input signals X_n with a finite collection of signals U_n. This introduces quantization errors into the communication process, causing a loss in information since there is no way one can recover X_n from U_n. Other than this, the difference between X_n and $\hat{X}_n$ may be due to some disturbances in the channel which corrupt the signal. These disturbances may be deterministic such as filtering, modulation, aliasing, or random such as additive noise, fading, and jamming. Then, the fundamental issues in communication system design are *source coding* or *data compression* which is the mapping of the input sequence (information source) efficiently into a representation for transmission over the channel and *channel coding* or *error-control coding* for overcoming the noise in the channel.

Claude Shannon formulated both of these issues in his classical papers [1], [2]. One important fact he proved was that in a communication system design, source coding and error-control coding (channel coding) can be considered independently: one can design seperate systems for source and channel coding and then simply cascade them. The result would be as good as the result of designing both systems together at the same time.

In our work, we focused on the problem of data compression, making some assumptions about the channel and then ignoring it completely, the approach being justified with Shannon's above mentioned theorem. Our first assumption about the channel is that it is digital, that its permissible inputs and outputs form a finite set or alphabet. Our second assumption is that the rate of the channel is fixed, that is, for each channel input symbol U_n chosen from the input alphabet there is just one output symbol $\hat{U}_n$. The third assumption is that the channel is noiseless, that is $U_n = \hat{U}_n$.

Data compression involves the design of the encoder and the decoder. As stated above, the encoder is a mapping from the input sequence X_n into U_n, the channel input sequence. This mapping can be performed in many ways, the objective still being to minimize the difference between the reproduction symbols and the input symbols. A special case of data compression is when the encoder is a minimum distortion or nearest neighbor mapping and this particular type of data compression is called *quantization.* Shannon's distortion rate theory formulates the lowest distortion achievable for a given fixed rate with such a system but it makes no suggestions for the ways of actually building systems with optimal performance. Therefore, many researchers since Shannon have concentrated on finding the rules for the design of systems with performance approaching theoretical bounds. Although many good coding systems have been suggested, there is still room before the bounds are reached. The goal of this work is to contribute to these efforts in the fields of trellis waveform coding and related systems using a different design approach.

Before we go into the details of our coding approach, we provide some background information on some important quantization techniques related to our work in the next chapter.

Chapter 2

QUANTIZATION TECHNIQUES

2.1 Scalar Quantization

Scalar quantization is the simplest of quantization techniques. It is simple because it performs quantization on a discrete time sequence considering the samples one by one, in other words, it quantizes the samples independently. The scalar quantizer is defined with a codebook $\mathcal{C} = \{y_1, y_2, \ldots y_N\}$ composed of the codewords y_is which are the reproduction values, and an encoding rule, which determines the way input symbols are encoded to one of y_is. The codewords partition the input space into N regions, $S_1, S_2, \ldots, S_N$, each S_i being composed of the points in the space that are assigned to y_i through the encoding rule. The elements x_n from the input sequence $\{x_1, x_2, \ldots, x_L\}$ are quantized one by one according to the encoding rule, that is, a symbol x_n from the input sequence is quantized to y_i if it falls into the region S_i. An important special case of the quantizer is the Nearest Neighbor (NN) quantizer. In NN encoding, the distance of x_n from each y_i is calculated via the distortion measure $d(x_n, y_i)$ and x_n is quantized in the following way,

$$q(x_n) = y_j, \quad \text{if} \quad d(x_n, y_j) \leq d(x_n, y_i), \qquad \exists j, \forall i \in \{1, 2, \ldots N\}, \tag{2.1}$$

where $q(x)$ denotes the quantizer function. If the equality holds, that is, if there is more than one such j, the choice is made randomly. The encoder being the NN encoding rule, the decoder is simply a look-up table (the codebook itself) which reproduces the codeword with the index received from the channel.

In the design of a scalar quantizer the goal is to come up with an encoding rule and a codebook which gives the best performance for input X_n in terms of a performance criterion, over all possible encoding rules and codebooks. The difference between the input sequence and the reproduction sequence is referred to as distortion. In order to measure the "distance" between the source sequence and the sequence of the quantized samples, one uses a mesaure, known as the distortion measure, between the two sequences. A wide class of distortion measures are known as per-letter distortion measures, i.e., the contribution from individual samples and their quantized values in a sequence have independent effects; for example, for additive distortion measures, these contributions are additive. There are various functions proposed in the literature as measures of sample distortion.

The most common distortion measure between two vectors $\mathbf{x}$ and $\hat{\mathbf{x}}$ whose members are x_i and $\hat{x}_i$, $0 \leq i \leq k-1$, is the squared error distortion or the square of the *Euclidean distance*:

$$d(\mathbf{x}, \hat{\mathbf{x}}) = \sum_{i=0}^{k-1} \mid x_i - \hat{x}_i \mid^2 . \tag{2.2}$$

Other possible distortion measures are *Holder norm*,

$$d(\mathbf{x}, \hat{\mathbf{x}}) = \{\sum_{i=0}^{k-1} \mid x_i - \hat{x}_i \mid^v\}^{1/v}, \tag{2.3}$$

Minkowski norm,

$$d(\mathbf{x}, \hat{\mathbf{x}}) = \max_{0 \leq i \leq k-1} \mid x_i - \hat{x}_i \mid, \tag{2.4}$$

and the *weighted-squares distortion*,

$$d(\mathbf{x}, \hat{\mathbf{x}}) = \sum_{i=0}^{k-1} w_i \mid x_i - \hat{x}_i \mid^2, \tag{2.5}$$

where $w_i \geq 0$, $i = 0, \ldots, k-1$.

These distortion measures are called difference distortion measures since they depend on the vectors $\mathbf{x}$ and $\hat{\mathbf{x}}$ only through the difference vector $\mathbf{x} - \hat{\mathbf{x}}$. Other types of distortion measures are also used in data compression systems but they are more complicated. Throughout this thesis we will only use the squared error distortion measure because it is commonly used in the data compression literature and because of its mathematical tractability.

2.2 Vector Quantization

A direct generalization of scalar quantization (SQ) to higher dimensions, that is, coding of symbols in blocks with length more than one is *vector quantization* (VQ). A vector quantizer Q of dimension k and size N is a mapping from a vector in the k-dimensional Euclidean space $\mathcal{R}^k$ into a finite set $\mathcal{C}$ containing N codewords or reproduction points. That is,

$$Q : \mathcal{R}^k \to \mathcal{C}, \qquad \mathcal{C} = \{\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_N\} \quad \text{and} \quad \mathbf{y}_i \in \mathcal{R}^k. \tag{2.6}$$

With analogy to the scalar case, the codewords $\mathbf{y}_i$s in the codebook $\mathcal{C}$ partition the input vector space $\mathcal{R}^k$ into N regions S_i, each composed of points in the $\mathcal{R}^k$ space which are associated with $\mathbf{y}_i$. Each vector $\mathbf{x}_n$ from the input sequence is encoded into $\mathbf{y}_i$ only if it is in S_i. Again, a special case is *nearest neighbor encoding* where the quantizer computes the distortion between the input vector and each codevector in the codebook and encodes the input vector to the one which gives the smallest distortion. That is,

$$Q(\mathbf{x}_n) = \mathbf{y}_j \quad \text{if} \quad d(\mathbf{x}_n, \mathbf{y}_j) \le d(\mathbf{x}_n, \mathbf{y}_i), \quad \exists j, \forall i \in \{1, 2, \ldots N\}, \tag{2.7}$$

where, again the choice is made randomly if the equality holds.

Vector quantization is more efficient than scalar quantization because it can exploit the correlation between samples which SQ cannot do since it quantizes the samples independently. Moreover, even if there is no statistical correlation to be exploited between the samples, VQ can do better than SQ due to its higher freedom in choosing decision regions for partitioning [4].

Since the quantizer is completely specified by a codebook and an encoding rule, the design objective of a vector quantizer is to find a codebook corresponding to the decoder and an encoding rule corresponding to the encoder, that will give the best performance. For a given codebook C, the average distortion (for empirical data) can be lower bounded according to

$$D \ge \sum_{k=1}^{L} \min_{1 \le i \le N} d(\mathbf{x}_k, \mathbf{y}_i). \tag{2.8}$$

This lower bound is achieved if Q assigns each vector $\mathbf{x}_k$ in the input sequence to a codeword which is the nearest neighbor condition. Looking for the optimal codebook given the partition leads us to the centroid condition. A centroid, cent(S), of a set $S \in \mathcal{R}^k$ is the vector $\mathbf{y}$ which minimizes the expected value of the distance between any point $\mathbf{x}$ in S and $\mathbf{y}$. That is,

$$\text{cent}(S) = \min_{\mathbf{y}}{}^{-1} E(d(\mathbf{X},\mathbf{y}) \mid \mathbf{X} \in S) \tag{2.9}$$

where the inverse minimum notation means the vector $\mathbf{y}$ satisfying the minimum is chosen.

It is easy to show that for the squared error distortion measure, this leads to the center of mass of S. For empirical data, the center of mass of S is,

$$\text{cent}(S) = \frac{1}{\| S \|} \sum_{i=1}^{\|S\|} \mathbf{x}_i \tag{2.10}$$

where the summation is over the vectors $\mathbf{x}_i$ that are in S. Then, given the partition (or equally the encoding rule) the optimum codebook is composed of the center of masses of each partition cell.

These necessary conditions of optimality suggest an iterative means of numerically designing a good vector quantizer. Given S one can start with an arbitrary initial codebook and partition S according to NN. Then the optimal codebook for this partition is found by computing the centroids and the new partition can be found for the new codebook. Iteratively, this procedure proceeds to better codebooks. This algorithm is known as the *Generalized Lloyd Algorithm* (GLA) or the *LBG algorithm* [3]. Although there are various vector quantizer design methods [4], GLA is the most popular and several extensions of it are made in the literature. The extension that is of interest to us is the one intoduced by Stewart *et al.* in [20] in the context of trellis waveform coding which we will discuss in length in the coming sections. At this point we give GLA in its original form as was introduced by Linde *et al.* [3] for VQ:

The Generalized Lloyd Algorithm

1. Begin with an initial codebook C_0. Set $m = 0$, and a threshold value $\epsilon > 0$.
2. Given the codebook, $C_m = \{\mathbf{y}_i\}$,
   partition the training set into cells S_i using the NN condition:
   $S_i = \{\mathbf{x} : d(\mathbf{x},\mathbf{y}_i) \leq d(\mathbf{x},\mathbf{y}_j);\ \forall j \neq i\}$.
3. Compute the centroids of the partition cells, S_i using (2.10),
   update the codebook according to the centroid condition:
   $C_{m+1} = \{\text{cent}(S_i)\}$.

```
4. Compute the average distortion, $D_{m+1}$, for $C_{m+1}$.
   If $(D_m - D_{m+1})/D_m < \epsilon$,
       stop with codebook $C_{m+1}$,
   else, set $m \leftarrow m + 1$.
```

2.3 Finite-State Vector Quantization

Vector quantization, as discussed in the previous section, considers each vector independently and therefore does not take into account the future or past vectors. Therefore the right term to define standard VQ is *memoryless VQ.*

As discussed in the previous section, the superiority of VQ to SQ (scalar quantization) partly comes from the fact that VQ exploits the correlation among the samples in the block. And the bigger the vector (or block) dimension is, the more VQ will exploit the statistical dependences in the sequence, since it will see more samples at a time. At this point, it is clear that contending with a finite vector size and quantizing the blocks seperately, we are ignoring the dependencies between samples in consequetive blocks which could be exploited for even better coding performance. One apparent solution to this problem is to increase the vector dimension indefinitely which is then accompanied with a proportional increase in computational complexity, which becomes unmanagable. Since this solution is not practical, instead of introducing more samples in the quantization process, we should include the information about these samples to the current quantization instant in some way, that is we should introduce *memory* into the quantizer. This memory is used to derive information about statistical dependencies between samples, and this information is exploited for better coding performance. This can be realized in the following manner.

In standard (memoryless) VQ, we base our quantization decisions on a fixed codebook $\mathcal{C} = \{\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_N\}$. Instead, we think of employing a codebook changing with time, we let a different codebook be used at each quantization instant and choose this codebook from a collection of codebooks according to the output of the previous quantization instant. Using the knowledge of previous quantization in deciding the codebook to be used, we effectively incorporate memory into the quantization process. If the separate codebooks are designed to suit different characteristics typical of the source, and the codebook selection procedure is designed properly as a good predictor of the trajectory the source will follow, interblock correlations will be exploited and the performance will increase. This form of vector quantization is called *recursive vector quantization* [4]. The important special case of recursive vector quantization is when the collection of codebooks contains only a finite number of codebooks and this VQ is called *finite-state vector quantization* (FSVQ) [4]. Choosing a different codebook from a finite collection of codebooks at each quantization instant suggests a "state-based structure," where each state is identified with the choice of codewords composing a particular codebook. The state with the codebook associated to it, which we can name as the *state-codebook*, describes the mode the quantizer is in, and is in a way a summary of the past behavior of the source. This "state-based structure" is a finite-state machine specified by a next-state function, determining state transitions and a decoder mapping which decodes the input bit stream to a reproduction symbol (a codeword from the current-state codebook).

The best way to pictorially describe the FSVQ finite state machine is through an example state transition diagram shown in Figure 2.1 with the assumption that the rate of the system is one bit per sample, corresponding to binary transitions or state-codebooks with size two. This FSVQ has four states, represented by circles numbered as 00, 01, 10, and 11. The lines with arrows represent possible state transitions at a decoding instant. $\mathbf{y}_{ij}$s are codewords from the ith state-codebook. For instance, the channel symbol 0 when the quantizer is at state 01 produces a reproduction symbol y_{10}, the codeword from state codebook 1 with index 0, and causes the quantizer to move to state 10. The transitions are equivalent to the next-state function, the labels of transition lines with the states are equivalent to the decoder mapping.

Now, let us give a formal definition of the finite-state vector quantizer (FSVQ):

Define a state space $\mathcal{S}$ as a collection of symbols $\mathcal{S} = \{s_1, s_2, \ldots, s_{K-1}\}$ called states. Let the vector dimension be k. Let the set of channel symbols be $\mathcal{U} = \{\mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_{N-1}\}$. The coding rate is $\log N$ bits per input vector or $k^{-1} \log N$ bits per source symbol. Then, a finite-state encoder is a mapping $\alpha : \mathcal{R}^k \times \mathcal{S} \to \mathcal{U}$, that is, given an input vector $\mathbf{x}$ and the current state s

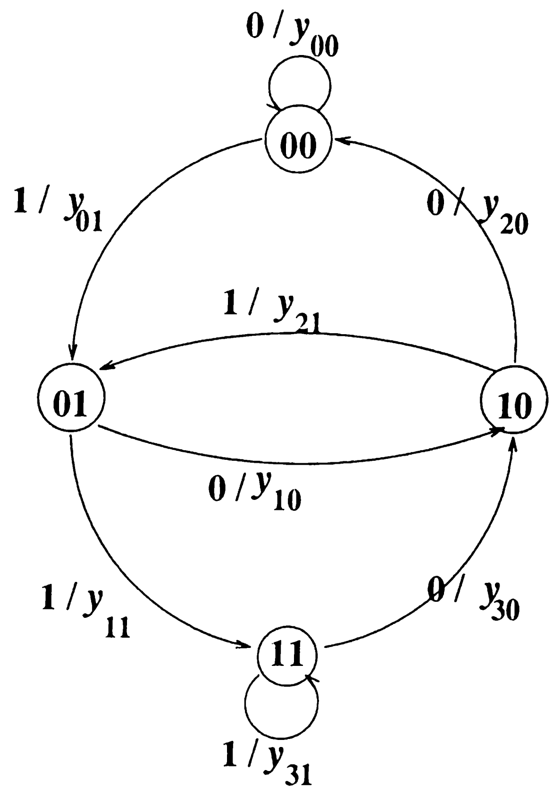


Figure 2.1: State Transition Diagram

the function $\alpha(\mathbf{x}, s)$ maps $\mathbf{x}$ into $\mathbf{u}$, a channel symbol. In addition to the encoder mapping, the quantizer is defined by the next-state mapping which is the essential difference of FSVQ from other VQ techniques. The next-state mapping is a function $f : \mathcal{U} \times \mathcal{S} \rightarrow \mathcal{S}$ which produces the next state $f(\mathbf{u}, s)$, given the current state s and an output symbol $\mathbf{u}$ produced from the input symbol $\mathbf{x}$.

Correspondingly, a finite-state decoder is a mapping $\beta : \mathcal{U} \times \mathcal{S} \rightarrow \mathcal{R}^k$, that is, given the current state s and the channel symbol $\mathbf{u}$ it produces the reproduction symbol $\hat{\mathbf{x}}$.

The output spaces of the encoder and decoder mappings are required to be the same, also the next-state mapping is resctricted to depend only on the current state and the encoder output rather than the input symbol. These two restrictions enable the encoder to track the state sequence given the initial state, and one does not need to send the state information in addition to the channel symbols.

To each state, s, a *state codebook*, $\mathcal{C}_s = \{\beta(\mathbf{u}, s), \mathbf{u} \in \mathcal{U}\}$, is associated which is composed of the possible reproduction vectors in that state.

Then, the encoding process of a random process $\{X_n, n = 0, 1, 2, \dots\}$ can be described as follows. Given an initial state $s_0 \in \mathcal{S}$, the channel symbol sequence, the state sequence, and the reproduction sequence are produced recursively for $n = 0, 1, 2, \dots$ as:

$$\mathbf{U}_n = \alpha(\mathbf{X}_n, S_n) \qquad S_{n+1} = f(\mathbf{U}_n, S_n) \qquad \hat{\mathbf{X}}_n = \beta(\mathbf{U}_n, S_n). \tag{2.11}$$

To complete the definition of FSVQ, we should also specify the encoding rule: FSVQ encodes according to minimum distortion or nearest neighbor condition. That is, using the Euclidean distance as the distortion measure, the encoder mapping α is defined by

$$\alpha(\mathbf{x}, s) = \min_{\mathbf{u}}{}^{-1} d(\mathbf{x}, \beta(\mathbf{u}, s)) \tag{2.12}$$

which means that $\alpha(\mathbf{x}, s)$ is the index $\mathbf{u}$ for which the reproduction codeword $\beta(\mathbf{u}, s)$ yields the minimum possible distortion over all possible reproduction codewords in the state codebook $\mathcal{C}_s$. In our discussion on vector quantization we noted that the minimum distortion encoding rule was the optimal encoding for a given codebook. Although in FSVQ minimum distortion encoding seems the most natural choice, in the long term, it may not be the best choice. Because, FSVQ with minimum distortion encoding optimizes only the short term performance of the system. Because of the memory in the quantizer, a codeword with very small distortion can lead to a state with a bad codebook for the next input vector. But, the minimum distortion rule is intuitively satisfying and no better encoder structure with comparable complexity is found so far. Therefore, we will contend with this encoding rule.

Suboptimality of the minimum distortion encoding in FSVQ is the consequence of FSVQ's having a memory of only one vector size. The remedy is to have an encoder with a memory of input sequence size which leads us to the trellis encoding system that will be discussed in the next section.

Note that VQ is a special case of FSVQ with only one state. Although FSVQ is more general, distortion rate functions of information theory show that optimal achievable performance (average distortion) for a given rate is the same for both cases [5]. But the performances of FSVQ and VQ are the same only when arbitrarily large vector dimensions are allowed. FSVQ, because of its higher ability to exploit correlations between samples, obtains the same performance with VQ using shorter vectors, therefore it provides systems with lower complexity.

Based on the finite-state machine perspective, we can identify two different ways of relating the state sequence and the reproduction sequence. These are pairing of each reproduction vector with a state or with a transition. The first type of FSVQ is called *labeled-state FSVQ* and the second one is called *labeled-transition FSVQ*.

The decoder mapping β of a labeled-state FSVQ depends on the current state and channel symbol only through the induced next state; the current reproduction $\hat{X}_n$ is determined by the next state s_{n+1}. On the other hand, the decoder output of a labeled-transition FSVQ is associated with the transition from the current state to the next state and therefore is determined by both the current state s_n and the next state s_{n+1}. These two configurations correspond to two different finite-state machines: labeled-state FSVQ to the *Moore machine*, and labeled-transition FSVQ to the *Mealy machine* [7]. The two structures are equivalent in the sense that Mealy and Moore machines are equivalent [6]. That is, given one, one can find an equivalent FSVQ of the other form, equivalent in the sense that given an initial state and an input sequence, the two quantizers will yield the same output sequence. The codewords are held constant in transition from one form to the other. For example, going from labeled-transition FSVQ to labeled-state FSVQ the codewords that were assigned to branches are assigned to separate states which amounts to an increase in the number of states.

As noted above, an FSVQ is fully determined by an encoding rule, state codebooks and the next-state function. Therefore, the design of a FSVQ focuses on generating state codebooks and a next-state function. First, we consider finding a good encoder α and a good decoder β given a fixed next-state function f. Finding the best decoder for the given next-state function and given encoder is equivalent to finding the best state codebooks. Suppose we have an input sequence $\{\mathbf{X}_n; n = 1, 2, \ldots, L\}$. If the initial state is s_0, encoding, we obtain the channel symbols $\mathbf{U}_n = \alpha(\mathbf{X}_n, s_n)$ and the state sequence $s_{n+1} = f(\mathbf{U}_n, s_n)$ for $n = 1, 2, \ldots, L$. Then our goal is to find the decoder mapping β minimizing the distortion,

$$D = \frac{1}{L} \sum_{n=1}^{L} d(\mathbf{X}_n, \beta(\mathbf{U}_n, s_n)). \tag{2.13}$$

It is easy to show that optimal decoder codevectors are the centroids [4], that is,

$$\beta(\mathbf{u}, s) = \min_{\mathbf{y}}{}^{-1} \frac{1}{\| M(\mathbf{u}, s) \|} \sum_{n \in M(\mathbf{u},s)} d(\mathbf{x}_n, \mathbf{y}) \tag{2.14}$$

where $M(\mathbf{u}, s)$ is the collection of $\mathbf{U}_n$ such that $\mathbf{U}_n = u$.

This is the optimum decoder for the given encoder and the next-state function, but the encoder may not be the optimum one for the obtained decoder, so the next step is to find the optimum encoder for the given decoder. Now we perform a nearest neighbor encoding using the state codebooks found and the given next-state function, which yields a new partition and therefore a new encoder α. Then, we should find the best decoder for the current encoder and given next-state function and the process of finding the best encoder and decoder continues iteratively until no significant performance gain is obtained by subsequent iterations. This procedure is indeed a variation of the generalized Lloyd algorithm. We can summarize the algorithm as follows.

FSVQ Encoder/Decoder Design Algorithm

```
1. Initialization:
   Given: a state space $\mathcal{S}$,
          an initial state $s_o$,
          an encoder $\alpha_0$
          a next-state function $f$,
          a training sequence $\{\mathbf{X}_n; n = 1, 2, \ldots, L\}$.
          Set $\epsilon > 0$, m=1, $D_0 = \infty$.
2. Encode $\mathbf{X}_n, n = 1, 2, \ldots, L$, using $\alpha_{m-1}$
   to obtain $\{\mathbf{U}_n, s_n\}; n = 1, 2, \ldots, L$.
   The state codebooks are modified into $\beta(\mathbf{u}, s) = \text{cent}(\mathbf{u}, s)$.
3. Replace the encoder $\alpha_{m-1}$ by the minimum distortion encoder $\alpha_m$ for $\beta_m$.
   Compute the distortion $D_m$, if $|D_m - D_{m-1}|/D_m < \epsilon$ quit else goto step1.
```

Then, the problem left is to design the next-state function. Several methods are proposed in the literature to solve this problem. These methods include *Conditional Histogram Design, Nearest Neighbor Design, Set Partitioning, Omniscient Design*, etc. A detailed discussion of these methods can be found in [4].

Conditional histogram design is one of the simplest techniques. First, the algorithm forms a supercodebook through applying GLA with standard VQ. A state is assigned to each codeword thus found. Then the method estimates the conditional probabilities of successor codewords in the supercodebook which is named the *classifier codebook* and forms a labeled-state FSVQ by only including the most probable codeword successors in the classifier codebook to the state codebook. Since the codewords are assigned to states, the choice of state codebooks also determines the next-state function.

The method called *nearest neighbor design* also generates a classifier codebook in the same way, but uses the distortion between the codewords and not the conditional probabilities for selecting the set of allowed new states from a given prior state. For each state assigned to each codeword in the classifier codebook, N nearest neighbors are found and the state codebook is formed with these codewords. Hence the next-state function is formed.

Another FSVQ design technique, called *omniscient design* was introduced by Foster *et al.* [7] and Haoui and Messerschmitt [8], and developed for speech coding by Dunham and Gray [9] and for image coding by Gersho and Aravind [10]. This method is more complicated than nearest neighbor and conditional histogram methods but it usually shows better performance and it can be used with more general classifiers than VQ. Although nearest neighbor and conditional histogram techniques are for the labeled-state FSVQ design, the omniscient method can be used for both labeled-transition and labeled-state FSVQ design. The details of this algorithm can be found in [4] and [7]. Reference [7] also provides simulation results for the comparison of various FSVQ design techniques. These results show that *omniscient labeled-transition design* (OLT) gives the best results for sources of practical interest. This method is also referred to in [4] as the method through which best results are obtained thus far.

One of the contributions of this work is to suggest and show a design algorithm that has better performance than the methods described above, which will be described in Chapters 3–5.

2.4 Trellis Waveform Coding

In this section, we turn our attention to a more advanced data compression scheme, *trellis waveform coding*, which is the main quantization scheme on which our work has focused with the goal of designing a near-optimum decoder.

This coding scheme has been very popular among many researchers since early 1980s, who made considerable progress towards an understanding of trellis encoding systems. The popularity of this data compression system is partly due to the fact that results in information theory have proved the existence of trellis systems which show performance close to the theoretical bounds [11], [12]. But these are only existence proofs, which do not describe the actual ways of constructing good codes. Therefore many researchers concentrated on the problem of finding rules for constructing good codes and came up with various design algorithms. The goal of this thesis is to make a contribution to these efforts, which has been achieved by designing an algorithm for constructing near-optimum codes based on an approach different from other work in the literature.

We will now describe the trellis waveform coding system. In the previous section, we discussed a VQ system called FSVQ. As was noted, FSVQ is superior to VQ due to the incorporation of memory into the quantization process. But as also explained, minimum distortion encoding may not be optimal in FSVQ, since a codeword with very small distortion can lead to a state with a bad codebook for the next input vector. Although through good design one can try to eliminate this problem, we can never be sure about the optimality of the encoding.

This observation leads us to the conclusion that the suboptimality of FSVQ encoding is due to its having a memory size of only one vector and the remedy is to increase the memory of the FSVQ encoder from vector size one to vector size M: instead of making "greedy" quantization decisions on vectors one by one, to delay the decision until M vectors are seen and to decide on these M vectors together. Then the quantizer will make a decision which is good for at least M vectors, and the probability of making bad decisions will decrease. In this manner, we expect to have an optimal encoder as M approaches the length of the vector sequence.

This operation is called *delayed decision encoding*, *lookahead encoding*, *multipath search encoding*, or *trellis encoding*, for reasons that will become apparent. For delayed decision encoding, we employ a finite-state machine for the decoder as in FSVQ where the states summarize the past behavior of the system, and approximate the current mode of behavior of the input sequence. In this case, the forms of encoding, decoding, and next-state mappings are different. In FSVQ, the state transition diagram (finite-state machine) is sufficient to explain the operation of the system; but in delayed decision coding, we need a more elaborate structure that takes the past into account explicitly.

For convenience we repeat here Figure 2.1 as Figure 2.2.a, which shows the state transition diagram of a FSVQ. The extension in time of the state transition diagram is the directed graph given in Figure 2.2.b. The stages in the graph correspond to consecutive time instants of the data compression process and each stage is equivalent to the state transition diagram. Each node corresponds to a distinct state at a given time, and each branch originating from a node represents a transition from that state (node) to some state (the node which the branch is connected to) at the next instant. The graph begins at state s_0 and ends at s_L. To each branch in the graph certain weights are assigned which are the reproduction symbols-or state codewords- in the FSVQ state transition diagram. This directed graph is called a *trellis* and it is a special case of a *tree*, branches of which are self-emerging, that is, branches originating from a common root (node) can meet again at another node later in the tree. The encoding system based on this data structure is called the *trellis source coding system.* To every possible state sequence of the trellis there corresponds a unique path. Given the channel symbols, the trellis can keep track of the state sequence and generate the reproduction symbols out of the state codebooks.

The trellis structure thus described can be used to represent a vector quantizer if the trellis has only one state or a finite-state vector quantizer but to represent a trellis encoding system we introduce measures assigned to each node along the trellis. The measure assigned to a particular node corresponds to the total distortion of a state sequence that starts at state s_0 and ends at that node. The encoder performs a nearest neighbor encoding in the following manner. At each time instant, for each node, it considers the input branches to that node and computes the distortions due to the codewords corresponding to these branches. Summing the node-distortion of each node which these branches originate from and the calculated distance of the corresponding branch, the total distortion faced by a particular path is calculated. The encoder decides on the path with the least distortion and assigns the distortion corresponding to this path to the node under consideration; it also stores the index of the branch connected to that node. Therefore, the encoder is a trellis search algorithm which tries to find the path with the minimum distortion. There are various algorithms in the literature for trellis search, *Viterbi Algorithm* (VA) [13] being the most popular one. The reason for its popularity is that it is an optimal search algorithm. A well-known alternative, the M-L algorithm, is not

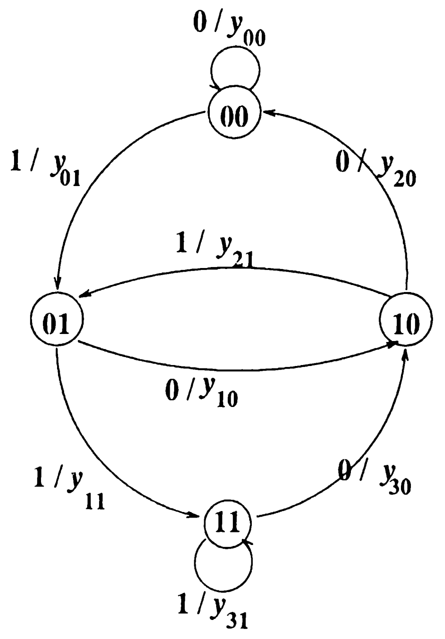


Figure 2.2: (a) State transition diagram

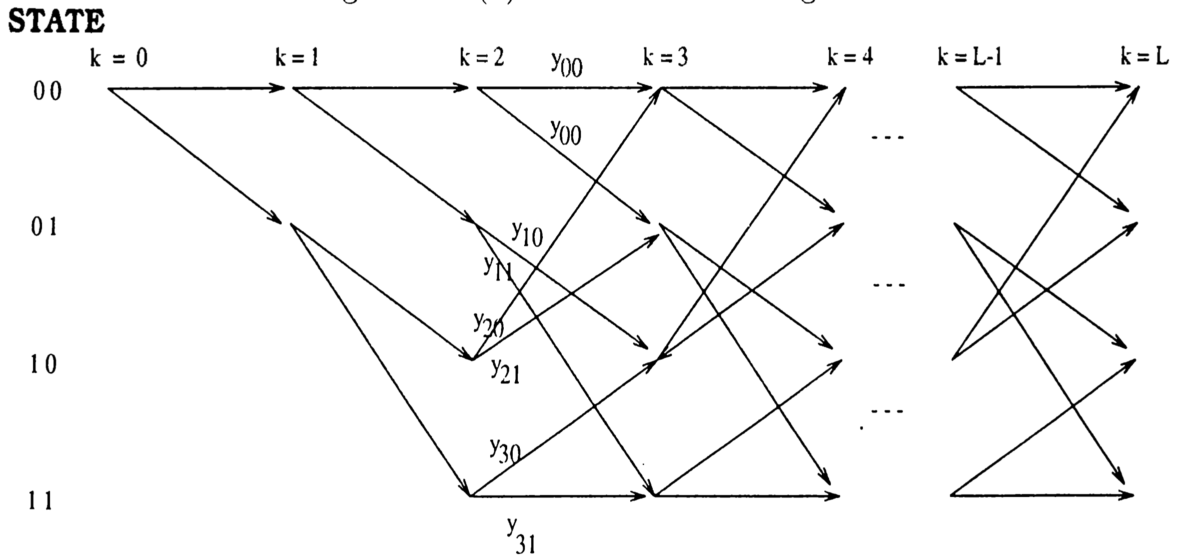


Figure 2.2: (b) Trellis diagram

optimum and it is better suited for tree search since it takes no advantage of the simpler trellis (self-emerging tree) structure.

The Viterbi algorithm was first suggested by Viterbi in 1967 [14]. It was later shown by Omura in 1969 [15] that it was a special case of *dynamic programming*. Here we summarize this important algorithm.

2.4.1 Viterbi Algorithm

```
Given : a collection of states $\mathcal{S} = \sigma_0, \sigma_1, \ldots, \sigma_{M-1}$,
        a starting state $s_0$,
        an input vector sequence $x_1, x_2, \ldots, x_L$,
        a decoder $\beta(u, s)$.
```

Let $d(.,.)$ denote the squared Euclidean distance and $D_j(k)$ denote the total distortion for state k at time j.

```
1. Set $D_k(0) = \infty, \quad 1 \leq k \leq M$
2. For $1 \leq n \leq L$ do
      2.1 For $0 \leq k \leq M$ do
            2.1.1 Calculate $d(j,k)$ for all states $j$ from which a branch
                  to state $k$ exists
            2.1.2 $D_k(n+1) = \min_j (D_j(n) + d(j,k))$
            2.1.3 Eliminate the branches other than the branch
                  achieving the minimum above. Save the optimum branch.
3. Find $\min_k D_k(L)$ which is the minimum distortion obtainable.
   Trace back the survivor path, which is the optimum state sequence.
```

Although the Viterbi algorithm is very favorable due to its optimality, there is a price paid for this. First of all, computational complexity is higher when compared with FSVQ. Second, there is the important practical problem that the algorithm does not make a decision on the optimum path until it reaches the final node. First of all, this amounts to the storage of all the survivor paths until the algorithm execution is completed. One observation enables us to get around this difficulty: most of the time we see that the survivor paths at time

k have a common root some l stages back at time $k - l$. Then the survivor paths are said to have merged at depth l. If all the survivor paths at time k have merged at depth l, we can safely make a decision about the optimum path up to time $k - l$ without waiting until the end of the sequence. An efficient way of performing truncated search is to stop the normal execution of VA at certain instants periodically and perform a back search to find the root, where the survivor paths are merged. When the root is found, a decision can be made for the optimal path before the root and the cursor of the storage array is simply moved to the root. Even if we cannot find a root, considering the path with the lowest distortion up to the decision node, we can force a decision for the optimum path. If we keep the period of searches large enough, that is, if we keep the truncation depth large enough, the probability of making errors in the truncations will be low. In the literature [16], a truncation depth TD of 5 times the constraint length is suggested. Obviously, performing VA with truncated search greatly reduces the memory storage requirements. Instead of storing $L \times N$ integers we just store $TD \times N$ integers and usually $TD \ll L$.

Another reason for preferring truncated search VA to standard VA is that when a coder is to be used in interactive applications, because of practical delay reasons, search lengths should be kept short. For typical interactive speech applications the practically allowable delay is no greater than 40 milliseconds which corresponds to search-length values around 256 in rate 1 bit/sample communication. But, of course, there are no such restrictions in broadcast or storage applications and as long as there is need to do so, long search lengths can be used.

Since the encoder of the trellis waveform coder is simply a trellis search algorithm for which we choose the Viterbi algorithm to use, the problem left is to design the decoder which is the objective of this thesis.

2.5 Predictive Trellis Waveform Coding

Another way of incorporating memory into the quantization process is to include prediction to the encoder. In the literature, this approach was applied to vector quantization and great improvement over standard VQ was reported [4]. An interesting approach was reported later by Ayanoğlu and Gray in [19] who replaced the VQ encoder with a trellis encoder and named the new system *predictive trellis waveform coding* (PTWC).

Our knowledge of DPCM states that for a good predictive encoder, prediction error samples are approximately white. Therefore, if the predictor is well-designed, combined with the advantages of trellis waveform coding, PTWC will exploit most of the redundancy in the source.

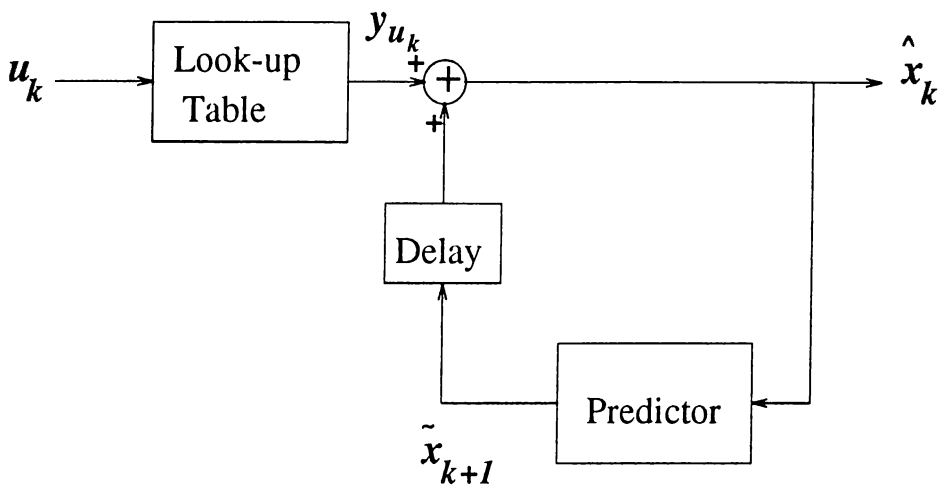
Predictive trellis waveform coding system is expected to perform better when compared to nonfeedback trellis encoders since whitening the source, which prediction does efficiently, means exploiting the statistical redundancy of a source better. It is also expected to perform better than DPCM due to delayed decision encoding for reasons explained before. One more advantage offered by delayed-decision encoding is the stabilizing effect on the decoder prediction filter [18].

2.5.1 System Description

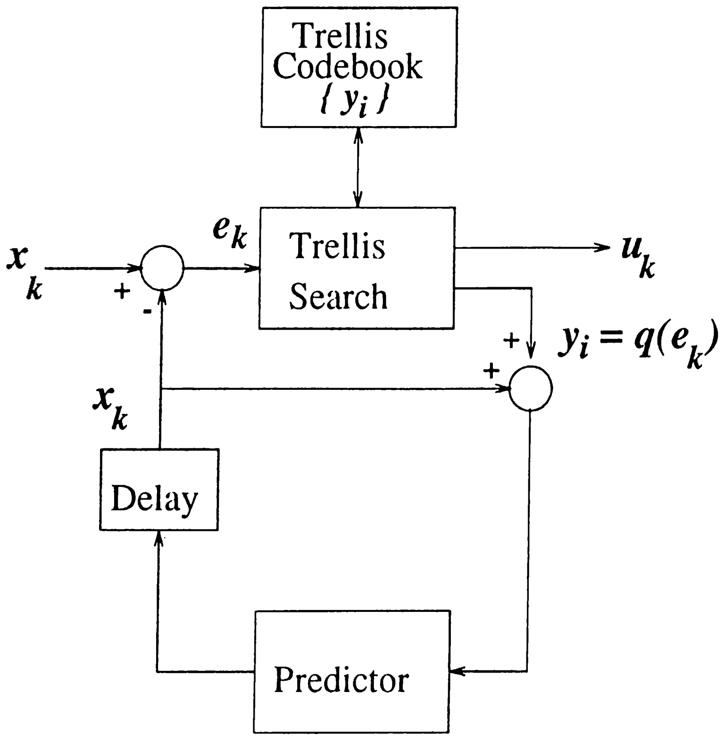
The encoder and the decoder of the predictive trellis encoding system are as given in Figure 2.3. In the encoder, the output of the predictor which tries to approximate the input is subtracted from the input to obtain the error symbols $\{e_k\}$. Having the error symbols as input, the trellis search decides on the best choice of a channel symbol sequence $\{u_k\}$ through minimum distortion encoding. Channel symbols u_k's are sent through the channel and received by the decoder which converts them into codewords with corresponding indexes. The decoded codeword is then added to the predictor output which is the same as the predictor in the encoder. As in the case of DPCM, the reconstruction error $x_k - \hat{x}_k$ is equal to the quantization error $e_k - q(e_k) = (x_k - \tilde{x}_k) - (\hat{x}_k - \tilde{x}_k)$ where $q(e_k)$ is the codeword assigned to e_k.

The most essential part of the system is the predictor, the design of which should be done carefully so that it predicts the input sequence efficiently. Ayanoğlu and Gray used a linear time invariant predictor since this would keep the decoder complexity low and would enable the use of relatively simple design techniques based on linear prediction theory [19].

Therefore, to define the predictive system suggested by Ayanoğlu and Gray



(a)



(b)

Figure 2.3: A predictive trellis coding system (a) Decoder (b) Encoder

[19], we should specify the trellis search algorithm and the codebook and predictor design algorithm. The trellis search algorithm aims at the optimal minimum distortion encoding of the input sequence in the presence of a predictor. Due to the finite state machine structure, a trellis search algorithm is possible and it is a modified version of the Viterbi Algorithm.

2.5.2 Search Algorithm

The search algorithm should keep an estimate for the previous L_p reproduction symbols $\hat{x}_{k-l}$ along the survivor path leading to state j, which will be used by the predictor at time $k+1$. Let $\hat{X}_k(j,l), l = 1, \ldots, L_p$ represent $\hat{x}_{k-l}$ along the path leading to state j at time k. Let $\hat{X}_k(j) = (\hat{X}_k(j,1), \ldots, \hat{X}_k(j,L_p))^t$, $a = (a_1, \ldots, a_{L_p})^t$. Let $D_k(j)$ represent the total distortion associated with the jth node at time k. Let $y(i,j)$ be the codeword on the branch connecting nodes i and j. The predictor order is L_p.

```
0. Initialization:
   $D_0(0) = 0,$
   $D_0(j) = \infty, \qquad 1 \le j \le 2^{K-1} - 1,$
   $\hat{X}_0(j) = 0, \qquad 0 \le j \le 2^{K-1} - 1,$
1. Recursion: For $0 \le k \le L_B - 1$, do
   1.1 For $0 \le j \le 2^{K-1}$, do
      1.1.1. Dynammic programming step:
         $D_{k+1}(j) = min_i(D_k(i) + d(x_k, a^t\hat{X}_k(i) + y(i,j)))$.
         The index $i$ is from the set of all nodes from which a path exists to node $j$.
         Save the argument minimizing this equation as $I_k(j)$.
      1.1.2. Update the first element of $\hat{X}_k(j)$ as
         $\hat{X}_{k+1}(j,1) = a^t\hat{X}_k(I_k(j)) + y(I_k(j), j)$
      1.1.3. Prediction update:
         $\hat{X}_{k+1}(j,l) = \hat{X}_k(I_k(j), l-1), \qquad 2 \le l \le L_p.$
2. When $n = L - 1$, find $j$ such that $D^{min} = \min_j D_{L-1}(j)$.
   Release the corresponding pathmap through the trellis to the channel.
```

The search algorithm is a direct extension of Viterbi algorithm and reduces to it for $a = 0$.

2.5.3 Design Algorithm

The encoder is specified by two sets of parameters: *the linear prediction coefficients*, $a_i; i = 1, \ldots, L_p$; and *the prediction error (residual) codewords*, $y_k; k = 0, \ldots, 2^N - 1$. The performance of the coding system is totally dependent on the good design of these parameters. The design algorithm assumes initial values for these parameters and then iteratively improves them. The initial values for the codewords can be generated with any of the known methods particularly with *extension* [20] or *splitting* [3]. The natural choice for the initial predictor coefficients is the solution of the Wiener-Hopf equation, $\boldsymbol{Ra} = \boldsymbol{v}$, where $\boldsymbol{R} = [R_{i-j}]_{L_p \times L_p}$ and $\boldsymbol{v} = [R_i]_{L_p \times 1}$. This choice for the initial predictor coefficients is based on the assumption that the original source inputs rather that the reproductions are the observables. We use coded reproductions in the predictive system therefore these parameters are not optimal, but these choices are intuitive and are good starting points for the design algorithm which improves them iteratively.

For a fixed prediction vector $\boldsymbol{a}$, the distortion for the given training source is

$$\sum_{n=1}^{L} d(x_n, \hat{x}_n) = \sum_{n=1}^{L} (x_n - \hat{x}_n)^2 = \sum_{n=1}^{L} (\epsilon_n - \hat{\epsilon}_n)^2. \tag{2.15}$$

This distortion is minimized if we change the codewords into centroids of partitioning cells,

$$y_i = \frac{1}{\| S_i \|} \sum_{n \in S_i} \epsilon_n; \qquad i = 0, 1, \ldots, 2^N - 1. \tag{2.16}$$

We try to predict x_n by

$$\tilde{x}_n = \sum_{l=1}^{L_p} a_l \hat{x}_{n-l}. \tag{2.17}$$

The orthogonality principle [4] implies that $\boldsymbol{a}$ should be such that the prediction errors and the observations are orthogonal. This leads to

$$\sum_{n=0}^{L-1} (x_n - \tilde{x}_n) \hat{x}_{n-i} = 0; \qquad i = 1, \ldots, L_p. \tag{2.18}$$

Substituting

$$\hat{x}_k = \sum_{j=1}^{L_p} a_j \hat{x}_{k-j} + q(e_k), \tag{2.19}$$

in (2.18), we obtain the predictor update equation

$$\sum_{j=1}^{k} a_j \sum_{k=1}^{L} \hat{x}_{k-j}\hat{x}_{k-i} = \sum_{k=1}^{L}(x_k - q(e_k))\hat{x}_{k-i}, \qquad i = 1, 2, \ldots, L_p. \tag{2.20}$$

Now, we state the predictive trellis waveform coder design algorithm.

0. Initialization:
   Generate initial codebook $C_0 = y_i^0; i = 0, \ldots, 2^{m-1}$
   Find initial predictor quefficients solving the Wiener-Hopf equation, $\boldsymbol{Ra} = \boldsymbol{v}$, where $\boldsymbol{R} = [R_{i-j}]_{L_p \times L_p}$ and $\boldsymbol{v} = [R_i]_{L_p \times 1}$.
1. Trellis Codebook Update:
   Encode the training sequence using $\{y_i^m\}$ and $\{a_i^m\}$
   in order to obtain $D^m = \sum_{k=1}^{L} d(x_k, \hat{x}_k)$.
   If $|D^{m-1} - D^m|/D^m < \epsilon$ stop with
   $y_i = y_i^m$, $0 \leq i \leq 2^{K-1}$ and $a_i = a_i^m, 1 \leq i \leq L_p$,
   else update the trellis codebook according to (2.16) to obtain $\{y_i^{m+1}\}$.
2. Predictor Update:
   Use $\{y_i^{m+1}\}$ and $\{a_i^m\}$ to encode the training sequence.
   Use (2.20) to obtain the new generation of predictor coefficients $\{a_i^{m+1}\}$.
3. Set $m \leftarrow m + 1$, go to 1.

2.6 Trellis Coded Quantization

Another data compression system based on trellis encoding and finite-state machine decoder is the *trellis coded quantization* (TCQ). This recently introduced data compression system is reported to give good results for memoryless sources and in predictive trellis coding [4].

Trellis coded quantization was first introduced by Marcellin and Fischer [51], who, motivated by the success of *trellis coded modulation* (TCM) in the field of modulation theory, and the results of alphabet-constrained rate distortion theory, constructed the source coding analog of TCM.

In 1982, Ungerboeck formulated coded modulation using trellis coding and introduced the ideas of set partitioning and branch labeling for trellis coder design [52]. The set partitioning ideas introduced in this work were based on the following observation: signal waveforms representing information sequences are most resistant to noise induced errors if they are very different from each other, that is, if there is a large distance in Euclidean signal space between the signal sequences. Following this fact, TCM designs the signal mapping function so as to maximize directly the "free distance" (minimum Euclidean distance) between coded signal sequences. Combined with the use of signal-set expansion to provide redundancy for coding and use of a finite-state encoder, this method led to a modulation scheme superior to conventional modulation techniques.

A particular TCM system is specified by the trellis structure (next-state function) and the codes assigned to trellis branches. As for the trellis structures, Ungerboeck suggested some symmetric trellises for $N = 4$ and $N = 8$ states. The branch connections are similar to the typical trellis of Figure 2.2 but for rates higher than 2, the transitions are multiple, that is each branch on the graph corresponds to 2 or more parallel transitions. On a conventional modulation system for a signal constellation of size 2^m, m bit codes are used to send one of the 2^m symbols. In trellis coded quantization, the signal constellation is first doubled to 2^{m+1} points. Then this constellation is partitioned into $2^{\tilde{m}+1}$ subsets, where $\tilde{m}$ is an integer less than or equal to m. $\tilde{m}$ of the input bits are used to select, by trellis encoding, which of the subsets the channel symbol for the current signaling instant will be chosen from. The remaining $m - \tilde{m}$ bits are used to select one of the $2^{m-\tilde{m}}$ channel symbols in the selected subset. By this way although the rate of the system is held constant, a finer coding is achieved through introducing redundancy.

The basic idea of *alphabet-constrained rate distortion theory* is to find an expression for the best achievable performance for encoding a continuous source using a finite reproduction alphabet. The theory is developed in [53] and [43]. Marcellin and Fischer in [51] inspecting the alphabet constrained rate distortion functions for the uniform i.i.d. source, made the observation that for a given encoding rate of R bits per sample, it is possible to obtain nearly all of the gain theoretically possible over the R bits per sample Lloyd-Max quantizer by using an encoder with an output alphabet consisting of the output points of the $R + 1$ bits per sample Lloyd-Max quantizer.

Motivated by this observation and TCM, Marcellin and Fischer constructed a fixed structure trellis for rate R encoding which employed the output points of rate $R+1$ Lloyd-Max quantizer as the codewords, assigned to trellis branches according to Ungerboeck's set partitioning and branch labelling rules [52]. The system they introduced in [51] is given in Figure 2.4. The trellis can be any of Ungerboeck's amplitude modulation trellises [52]. But the branches presented here no more represent single transitions but multiple ones quantity of which is determined by the rate of the system. Consider an encoding rate of 2. Then the rate 3 Lloyd-Max output points (for uniform i.i.d. source), which will be employed as the codewords are as shown on real line in Figure 2.4. These codewords are partitioned into four subsets by labeling consecutive points with $D_0, D_1, D_2, D_3, D_0, D_1, D_2, D_3, \ldots$ starting with the leftmost (most negative) point and proceeding to the right. Then these subsets are assigned to the trellis branches following branch labelling rules of Ungerboeck [52]:

1. Parallel transitions are associated with codewords with maximum distance between them.
2. The branches originating from the same node should be labeled with subsets with maximum distance between them.
3. The branches terminating at the same branch should be labeled with subsets with maximum distance between them.
4. All codewords should be used with equal frequency in the trellis diagram.

The first rule is satisfied with the above set partitioning. To satisfy second and third rules the subsets are grouped as D_0 with D_2 and D_1 with D_3 and these groups are assigned to leaving and entering branches as shown in the figure. Fourth rule is already satisfied with this labeling.

This system is later modified by Marcellin and Fischer to incorporate prediction and they introduced PTCQ in [51]. The trellis search algorithm they use in their predictive system is the same as the search algorithm of Ayanoğlu and Gray [19] for $R = 1$ and an extension of it for higher rates, but in the design stage they do not train the codebooks and they do not update the predictor coefficients.

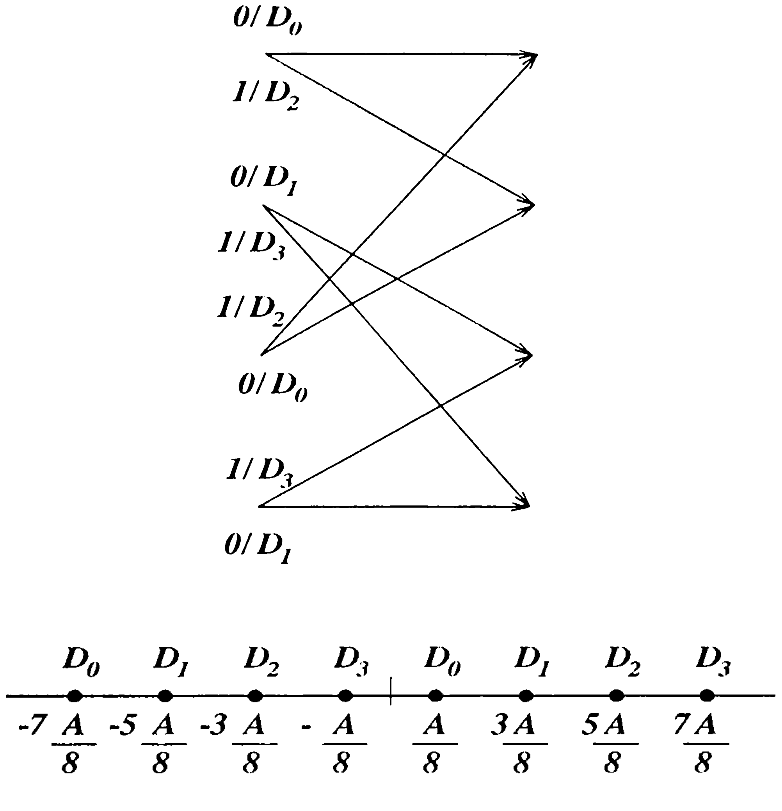


Figure 2.4: Marcellin and Fischer's TCQ system

Chapter 3

SIMULATED ANNEALING

Optimization is an issue of high importance in many diverse areas, in particular, it is a vital element of analysis and design in many fields in electrical engineering. In electrical engineering, particularly in the field of telecommunications, we sometimes deal with discrete variables and may need to carry out combinatorial analysis, that is, we deal with the arrangement, grouping, ordering, or selection of discrete objects. In these analyses, being engineers, our objective is to find out the optimal arrangements, orderings or selection of discrete variables. In other words, we are frequently confronted with *combinatorial optimization* problems.

Many common problems in fields such as electrical engineering, operations research, and computer science are combinatorial optimization problems, but the field particularly owes the existence of its wide range of applications to the advent of digital computers. Most currently accepted methods of solving combinatorial optimization problems would not have been considered seriously 10 or 20 years ago, for the reason that no one could have carried out the computations involved. However, even today, while many powerful digital computers are available, various large scale combinatorial optimization problems cannot be solved in reasonable time. Most of these problems are *NP-complete* problems [21], in other words, they are not solvable by a computational effort bounded by a polynomial function of the size of the problem.

Thus, one is forced to use *approximation algorithms* or *heuristics*. Heuristics are not guaranteed to get the optimum answer, they are designed to give an acceptable answer (hopefully close to the optimum answer) with a reasonable amount of computational effort or equivalently time. That is, using a heuristic, one makes a compromise between the optimum result and the computational effort.

Now, let us give a formal definition of a combinatorial optimization problem. A combinatorial optimization problem is formalized as a pair (S, C), where S is the countable (finite or infinite) *configuration space* or the set of configurations and C is a cost function, $C : S \rightarrow \mathcal{R}$, ($\mathcal{R}$: the set of real numbers) which assigns a real number to each configuration. For convenience, C is defined in such a way that decreasing values of C correspond to better configurations. With this definition, the optimum configuration S_{opt} is the configuration for which C takes its global minimum value. That is,

$$C_{opt} = \min_{i \in S} C(i), \tag{3.1}$$

where C_{opt} denotes the optimum value of the cost function. The objective of a combinatorial optimization problem is to find the configuration that gives C_{opt}.

Simulated annealing is one of the heuristics suggested to solve large-scale combinatorial optimization problems efficiently, although not exactly, with reasonable amount of computational effort. It is a heuristic or an approximation algorithm in the sense that it is not a mechanical sequence of computations to solve a specific problem, and its performance is highly dependent on how the user tailors it for a specific problem. There are various heuristic strategies for solving combinatorial optimization problems such as "constructive" heuristics which construct an answer directly. Simulated Annealing instead is related to "iterative improvement" strategies, which construct an initial suboptimal optimal solution and then perturb this solution slightly, in the direction of a better solution on the average.

The simplest algorithm this strategy suggests is the *iterative improvement algorithm.* Before describing this algorithm we define a neighborhood S_i for each configuration i, consisting of all configurations that can be reached from i in one transition. Let i denote the current configuration, i_{new} the configuration after perturbation.

Iterative Improvement Algorithm:

```
$i \leftarrow i_0$          /*initial configuration*/
repeat
   perturb($i, S_i$);          /*choose randomly $i_{new} \in S_i$*/
   if $C(i_{new}) < C(i)$
      $i \leftarrow i_{new}$          /*current configuration is replaced by the neighbor*/
until no $i_{new} \in S$ exists such that $C(i_{new}) < C(i)$
```

There are two obvious disadvantages of this algorithm. First, although it is certain that the algorithm reaches a minimum, there is no guarantee that it is the global minimum. Instead, the algorithm may get stuck in a local minimum and there is generally no information as to the amount by which this local minimum deviates from a global minimum. Second, the obtained local minimum depends on the initial configuration. There are some proposed ways of getting around these inadequacies. First, one can execute the algorithm for a large number, say N, of initial configurations [22]. For $N \rightarrow \infty$, this algorithm finds the global minimum with probability 1. Second, one can use the information gained through previous runs to improve the choice of an initial configuration for the next run [24]. Third, one can introduce a more complex generation mechanism, in order to be able to "jump out" of the local minima corresponding to a simple generation mechanism. Fourth, one can accept transitions which correspond to an increase in the cost function in a limited way.

The second and third approaches are strongly problem dependent so they do not lead to a general algorithm. The first one was the traditional approach until 1982 when Kirkpatrick *et al.* suggested the fourth one which they called *simulated annealing* [23]. Many experiments verified that simulated annealing is superior to the first approach [24].

Simulated annealing is based on an analogy between a process called annealing of solids in condensed matter physics and large combinatorial optimization problems. Annealing is a process in which a solid is heated up to a maximum value at which all particles of the solid randomly arrange themselves in the liquid phase and then it is cooled down very slowly. Through this process, all particles arrange themselves in the lowest energy configuration if the maximum temperature is high enough and the cooling process is carried out sufficiently slowly. During the cooling process, the solid is allowed to reach *thermal equilibrium* which is characterized by the probability of the solid's being in a state with energy E which is given by the *Boltzmann distribution*:

$$Pr\{Energy = E\} = \frac{1}{Z(T)} \exp(-\frac{E}{k_B T}), \tag{3.2}$$

where $Z(T)$ is a normalization factor depending on T and k_B is the Boltzmann constant.

In 1953 Metropolis *et al.* proposed an algorithm to simulate the evolution of a solid towards *thermal equilibrium* [25]. This algorithm can be summarized as follows: Given the current state (configuration) of the solid which is determined by the configuration of its particles, a randomly chosen particle is slightly moved from its current position. The resulting energy is calculated and compared with the previous energy of the solid. If $\triangle E$ is negative, that is, if the perturbation leads to a decrease in the total energy of the system, the process is continued with the new state. If $\triangle E$ is nonnegative, then the new configuration is accepted as the new state with probability $\exp(-\frac{\triangle E}{k_B T})$. This acceptance rule is called the *Metropolis criterion*. If the algorithm is executed until sufficiently many perturbations are made with this acceptance criterion, the probability distribution of the configurations (or states) approaches the Boltzmann distribution, which states that the system reached thermal equilibrium.

The problem of minimizing the energy of the solid is indeed a combinatorial optimization problem, the configuration space S being the possible configurations of particles in the solid, the cost function C assigning an energy value to each configuration. This observation suggests a way to handle general combinatorial optimization problems. For the problem in hand, we can generate a sequence of configurations with the Metropolis algorithm, that is, using the Metropolis criterion in configuration transitions, and in the end we can reach a configuration of *thermal equilibrium* characteristic to that value of the control parameter. If we repeat this Metropolis process for a sequence of decreasing values of the control parameter, we can hope to reach the global minimum just as nature does in the annealing of solids. The described process is nothing but *simulated annealing*. The analog of energy is the cost function and the analog of configuration of particles is the set of values problem variables take which is a point in the configuration space S. As in the solid state physics analogy, from a configuration i we pass to another randomly chosen configuration j with probability 1, if $\Delta C_{i,j} < 0$ and with probability $\exp(-\frac{\Delta C_{i,j}}{t})$, if $\Delta C_{i,j} > 0$, t being the control variable. Now, we introduce the *simulated annealing* algorithm [24].

Simulated Annealing Algorithm

```
begin
INITIALIZE;
$M := 0$ ;
repeat
    repeat
        PERTURB (config. $i \rightarrow$ config. $j$);
        if $\Delta C_{i,j} \leq 0$ then
          UPDATE(config. $j$)
        else if $\exp(-\frac{\Delta C_{i,j}}{t}) > random[0, 1)$ then
          UPDATE(config. $j$);
    until quasiequilibrium is reached;
    $t_{M+1} := f(t_M)$;
    $M := M + 1$;
until stop criterion;
end.
```

Although we have pointed out the existence of a strong analogy between the annealing of solids, which is known to give optimum results for sufficiently slow cooling, and solving combinatorial optimization problems with simulated annealing, one needs a formal proof for the convergence of simulated annealing to the global optimum. Such proofs are given in [24], [26]. But these convergence proofs are asymptotic convergence proofs; convergence of *simulated annealing* to the global minimum is guaranteed only if infinite length Markov chains are used and infinitely slow cooling schemes are applied. We cannot allow such schemes in practical problem solving. Instead, we should contend with finite speed cooling schemes and finite length Markov chains.

3.1 Practical Implementation

Practice has shown in recent years that simulated annealing is still very successful when finite Markov chains are used and the cooling process is not infinitely slow. But the performance of the algorithm highly depends on the design of the parameters of the algorithm.

In a practical implementation one should specify the following:

- initial value of the control parameter or temperature t_0;
- final value of the temperature t_f (stop criterion);
- length of the Markov chains;
- move-set (neighborhood structure), that is the set of allowable perturbations;
- a rule for changing the current value of the control parameter, t_k, into the next one, t_{k+1}.

We will now cite some simple schemes from the literature for determining the values of these parameters.

The initial value of t is chosen such that virtually all transitions are accepted, that is $\exp(-\Delta C/t_0) \approx 1$ for all transitions. An empirical rule is given by Johnson *et al.* [28]: determine t_0 by calculating the average increase in cost (or energy), for a number of random transitions and solve t_0 from

$$\mathcal{X}_0 = \exp(-\Delta\overline{C}/t_0), \tag{3.3}$$

where $\mathcal{X}_0$ is the *acceptance ratio* defined as the ratio of the number of accepted transitions to the number of proposed transitions.

The final value of "temperature" can be determined by fixing the number of temperature values t_k, for which Metropolis loops are to be executed. Also, the execution can be terminated if the last configurations of consecutive Markov chains are identical for a number of chains. Or, as in determining t_0, we can introduce a parameter $\mathcal{X}_f$, and can terminate execution when the acceptance ratio is smaller than $\mathcal{X}_f$.

The simplest choice for the length of the Markov chain is a value depending polynomially on the size of the problem [29]. Other than [29], various schemes are suggested in the literature. If $N(k)$ represents the length of the kth Markov chain, one can use arithmetic $N(k) = N(k-1) + C$, geometric $N(k) = N(k-1)/\alpha(k)$, logarithmic $N(k) = C/\log(t(k))$ schemes or continue until a number of acceptances are made, or until a number of rejections have occured.

The decrement in the temperature should be chosen such that small Markov chain lengths suffice to reestablish quasi-equilibrium after the decrement. Therefore, the changes in the value of temperature should be small. Simple temperature decrement rules include arithmetic, $t_{k+1} = C + t_k$, geometric $t_{k+1} = \alpha \times t_k$, and logarithmic $t_k = C/\ln(1+k)$ decrement functions.

Each combinatorial optimization problem suggests different neighborhood structures. Therefore the choice of the move-set is problem dependent.

There are more elaborate cooling schedules cited in [24] but those are derived for specific problems. Further, one elaborate schedule that is very successful in one problem can perform far worse than a simple schedule in another problem. Therefore, in the course of our work, we used simple schedules.

In the literature, attempts have been made to give good measures about the general performance of simulated annealing, in terms of the *quality* of the final solution obtained by the algorithm and the *running time* required by the algorithm. Lundy and Mees [30] succeeded in obtaining the worst-case result for the total number of transitions generated during the execution of the algorithm which is $O(\mid S_{neig} \mid \ln \mid S \mid)$, where $\mid S_{neig} \mid$ is the size of neighborhoods and $\mid S \mid$ is the size of the configuration space. Since, for most combinatorial optimization problems, the sizes of the neighborhoods can be chosen to be polynomial and the size of the configuration space $\mid S \mid$ is exponential, this formula shows that the execution of the algorithm takes polynomial time for most combinatorial optimization problems. For a bound on the worst-case performance of the result of algorithm, Sasaki and Hajek [31] provided a probabilistic measure.

Since its introduction in 1982 [23], simulated annealing has been successfully applied to many diverse combinatorial optimization problems. It became most popular in the field of VLSI design especially in *placement* and *routing* problems, where other known methods provide poor results. It has been used in image processing for *image restoration and enhancement* problems. The first paper in this context was published by Geman and Geman [32], in which a generalization of simulated annealing is used to find a maximum posterior distribution for degraded image. El Gamal *et al.* used SA on problems involving source codes, constant weight channel codes and spherical codes. Specifically, they considered the problem of representing the set of all 2^L binary sequences of length L by a much smaller subset of 2^K codewords ($K \ll L$) in such a way that the average Hamming distance between each of the 2^L sequences and its nearest codeword is minimal. They report that the results are very encouraging [33]. Çetin and Weerackody [35] and Flanagan *et al.* [34] applied simulated annealing in codebook design for vector quantization. Other fields simulated annealing has been applied include *neural networks, numerical analysis, biology, materials science, scheduling, statistics* and *graph theory.*

In almost all of these fields, SA has proved to be a successful algorithm, especially in the solution of large-scale problems for which no tailored solutions are known. For more information about the applications of simulated annealing the reader is referred to the survey paper by Collins *et al.* [36].

Chapter 4

PROBLEM DEFINITION AND SOLUTION

The goal of this work is to introduce a new algorithm for the design of trellis-based coding systems with performance higher than other work in the literature and to contribute to the study of these systems. With "trellis-based coding systems," we refer to the coding systems with finite-state machine decoders such as *finite-state vector quantizers* (FSVQ), *trellis waveform coders* (TWC), *predictive trellis waveform coders* (PTWC) and *trellis coded quantizers* (TCQ).

The difference of our design approach when compared to other work in the literature is in the way we design the *next-state function* of the decoder finite-state machine.

As we have noted in our discussion of trellis waveform coding in Chapter 2, the encoder of a trellis waveform coder is simply a trellis search algorithm and there exist various trellis search algorithms in the literature with well-known performance tradeoffs. Therefore, we concluded that the design problem of the trellis waveform coder reduces to the design problem of the decoder finite-state machine. In FSVQ, the encoder is simply the encoder of the corresponding state-VQ which is the nearest neighbor encoding rule. Therefore, similarly, the design problem of FSVQ reduces to the design problem of the decoder finite-state machine.

The decoder finite-state machine is completely specified by the state codebooks or the output map and the next-state map which corresponds to the branch connections in trellis graph. Hence, FSVQ and TWC design problems are composed of two design problems:

- output map design

  and

- next-state map design.

We will consider these two problems first separately, that is, we will focus on the design of the output map for a given next-state map and on the design of the next-state map for a given output map separately. Then, we will combine the solutions to these two problems to propose our decoder finite-state machine design algorithm.

4.1 Next-State Map Design

The central contribution of this thesis is the suggestion of a new heuristic for the design of the next-state map of the decoder finite-state machine. Given the current state and the channel index, the next-state map is equivalent to one-stage of the trellis diagram or the state-transition diagram.

The trellis diagram is specified by the number of nodes and the orientation of branches, that is, connections between the nodes. Each different set of connections correspond to a different trellis structure and therefore a different next-state map. Then, the problem of finding the optimum next-state map is equivalent to the problem of finding the optimum set of connections of the branches.

This is clearly a combinatorial optimization problem, S, the configuration space being the space of possible trellis structures, and C, the cost function, the value of which is to be minimized over S, being the total distortion. For a trellis coding system of rate R, N states and vector size k, the trellis will have N nodes and 2^{kR} branches originating from each node (assuming binary communications). This trellis can be constructed in $N^{2^{kR}N}$ different possible configurations. Since the size of the state space is exponentially dependent on the system variables, it is not practically possible to solve this problem by exhaustive search (a trellis system with unit rate and 32 states for scalar quantization requires more than 10^{96} iterations).

Due to the enormous complexity of the problem, we look for a heuristic. In the literature, some heuristics are suggested for the solution of this problem. The ones suggested by Foster *et al.* [7] in the context of FSVQ were described in Chapter 2. The drawback of these heuristics were that they could only iteratively improve the codewords for a given next-state function, providing no mechanism for improving the next-state function. That is, once the next-state function is designed it is fixed and not tuned to the codebook. Also, these heuristics are not intuitively simple. Dunham and Gray in [9] proposed a stochastic iteration algorithm to allow incorporation of the next-state function design in a probabilistic manner, but their algorithm is not straightforward. An interesting work in the literature is by Juang who suggested obtaining a *minimum degradation network* by a pruning procedure which he called *Pruned Trellis Vector Quantizer* [37]. His algorithm begins with a fully connected trellis and proceeds by pruning the branches, the removal of which causes the minimum degradation. The algorithm stops when a desired rate is reached. This algorithm is a modification of the *branch and cut* algorithm from linear programming. Juang noted that this algorithm was not successful for rates equal to and below 2.

We propose using *Simulated Annealing* for the design of the next-state map of trellis decoder finite-state machine:

- The **state space** consists of all the possible trellis structures (branch connections) with the constraint that there are 2^{kR} branches originating from each node. This restriction is made since we assume binary communications.

- The **cost function or energy function** to be minimized is simply the total distortion calculated by Viterbi algorithm.

- The **move-set** is the changing of the orientation of one of the branches, alternatively the **neighborhood set** is the set of all trellises obtainable by moving the end of a branch from the state it is connected to, to another state.

- The **initial value of the temperature** is calculated in the way Johnson *et al.* [28] suggested.

- The **length of Markov chains** is chosen to be linearly dependent on the number of states as $c \times N$, where N is the number of states, and c is a constant. The constant c is determined by experimental means and using intuition. For example, for fast cooling schedules one needs longer Metropolis loops to stabilize, or as the number of nodes in the trellis is increased, the size of the state space increases and longer Metropolis loops are needed to reach equilibrium.
- *Geometric improvement* is chosen as **the decrement rule** for temperature. That is, $t_{n+1} = C \times t_n$, n: time index.
- No **final value for temperature** is chosen. The cooling is exited when no more significant improvements occur.

For a given output map, the next-state map design algorithm begins with an initial trellis with a corresponding known distortion, and an initial temperature. Then the algorithm perturbs the trellis by breaking the end connection of a branch from its current position and connecting it to another state, hence changing the state-transition matrix. The new distortion is calculated via encoding the input source with the new trellis by the Viterbi algorithm, and compared with the previous distortion. If the new distortion is smaller, the perturbed trellis is accepted as the current trellis, else a random number in the interval $[0, 1)$ is generated and compared with the exponential $\exp((\triangle^{m} - \triangle^{m+1})/T)$. If the exponential is greater, the perturbed trellis is accepted, otherwise it is rejected. The algorithm continues to perturb the trellis this way until the system reaches quasi-equilibrium at this temperature T. The condition for reaching quasi-equilibrium is dictated by the choice of length of the Markov chain. This is one Metropolis loop. Then, the temperature is decreased according to the cooling function (geometric cooling) and another Metropolis loop is started. The algorithm terminates when no more significant improvements are seen at the outputs of the Metropolis loops.

4.2 Output Map Design

We propose the adaptation of GLA for the design of state-codebooks for a given trellis structure. Actually, adaptations of GLA to TWC and FSVQ have been used by many researchers in the literature. GLA was suggested in the context of TWC first by Stewart *et al.* [20] for codebook improvement. They also developed an extension algorithm which increases the constraint length by 1 producing a double size trellis from a given trellis, the performance of which is at least as good as the performance of the trellis before extension. Combining their *codebook improvement algorithm* and *extension algorithms* they proposed an algorithm for the automatic design of a trellis decoder with N states.

GLA is also employed by Foster *et al.* [7] in the context of FSVQ, by Ayanoğlu and Gray in predictive TWC [19], and by Bei and Gray [38] in vector TWC.

We employ Stewart's *codebook improvement* algorithm for improving a fixed trellis structure (fixed next-state function) and we propose an *extension* algorithm of our own for producing a good initial structure and codebook for optimization of the trellis with constraint length increased by 1. Here, we cite Stewart's codebook improvement algorithm:

Codebook Improvement Algorithm

0. Initialization:
   Given a distortion threshold $\epsilon > 0$,
   a binary noiseless channel,
   an N-state decoder,
   an initial codebook C^0 with cardinality $\| C^0 \| = M = 2^{kR}N$,
   and a training sequence $\{x_j : j = 0, 1, \dots, n-1\}$, set $m = 0$.
1. Encoding:
   Given $C^m = \{y_i^m : i = 0, \dots, M-1\}$ the codebook for generation m,
   find the minimum distortion trellis encoding $\{\hat{x}_j : j = 0, \dots, n-1\}$
   of the training sequence.
   This encoding induces a partition on the training sequence
   $\{S_i^m : i = 0, \dots, M-1\}$ with $S_i^m = \{j : \hat{x}_j = y_i^m\}$.
   Each set S_i^m contains the time indexes of those elements of
   the training sequence which are encoded by codeword y_i^m.
2. Compute the average distortion $\Delta_m = n^{-1}\Sigma_{i=0}^{n-1} d(x_j, \hat{x}_j)$.
3. If the decrease in distortion has fallen below the threshold ϵ,
   $(\Delta_m - \Delta_{m-1})/\Delta_{m-1} \leq \epsilon$,
   then halt with C^m as the final codebook. Otherwise goto step 4.

4. Find the optimal codebook C^{m+1} for generation $m+1$ as
   $C^{m+1} = \{y_i^{m+1} : i = 0, \ldots, M-1\}$
   where the y_i^{m+1} are the centroids of the new partition
   $\{S_i^{m+1} : i = 0, \ldots, N-1\}$.
   Replace m by $m+1$ and go to step 1.

The initial codebook has two codewords since the constraint length of the decoder is 1 (trellis has only 1 state). These initial codewords can be chosen simply as -1 and 1.

Now, we introduce our extension algorithm,

Trellis Extension Algorithm

```
0. Given number of states $N$, constraint length $k$, rate $R$, vector dimension $l$,
   the super codebook $C = \{y_{i,j}^k : i = 0, \ldots, N-1, j = 0, \ldots, 2^{lR}\}$ ,
   where $y_{i,j}^k$ is the $j$ th codeword (corresponding to $j$ th branch) of $i$ th state,
   and state transition matrix (or trellis diagram).
   0.1 Increase constraint length by 1 : $k \leftarrow k+1$.
1. Codebook extension :
   1.1 Retain the codebooks of the old states :
       $y_{i,j}^{k+1} = y_{i,j}^k, i = 0, \ldots, N-1, j = 0, \ldots, 2^{lR}\},$
   1.2 Assign codewords to the new branches in the following way:
       $y_{i,j}^{k+1} = y_{i-N,j}^k, i = N, \ldots, (2 \times N) - 1, j = 0, \ldots, 2^{lR}\}.$
2. Trellis extension :
   For ( $0 \le i \le (2 \times N - 1)$) do
      2.1 if $i$ is even then
            Retain the connections of the branches originating from state $i$
            as in the previous trellis;
      2.2 else if $i$ is odd then
            Connect the branches originating from state $i$ to states
            with indexes $N$ more than the indexes of the states
            (addition according to $mod2N$)
            to which the branches were connected in the previous trellis.
```

The trellis can be extended by just doubling the original trellis size (number of states) and inserting an identical copy of the original trellis for the newly generated states. But in this newly formed trellis, the two identical trellises are separate; no branch originating from a state in one of the trellises ends at a state in the other trellis. This structure can be no better than the original mother trellis. To have a chance for significantly better structures the branches should spread. It will take time for SA to form such an "unbiased" next-state map by perturbing. To accelerate this process the extension algorithm introduced above flips some of the branch connections corresponding to the states with the same positions in the two identical trellises. In this way, some of the branches starting from a state in one (half)trellis end at states in the other (half)trellis. The performance is no less than the performance of the original trellis since original connections are preserved. The Viterbi algorithm in the worst case will choose a path identical to the optimum path in the original trellis. Due to the introduction of new paths the performance may even be better.

Stewart *et al.*'s extension algorithm produces an extended trellis which is at least as good as the original trellis, too. Their algorithm extends the codebooks in the same way we did. The difference between the two algorithms is in the way the next-state function is modified. Stewart *et al.* used a shift register decoder (giving the fixed trellis structure) [20]. While extending the trellis they simply added a new cell to the shift register. Since we are not using a state-transition matrix instead of a shift register decoder, we preferred to use the algorithm described above which is applicable to a general state-transition matrix.

4.3 Trellis Decoder Design Algorithm

Assuming the rate is unity and scalar quantization is performed:

```
0. Initialization:
   $N = 1$,
   codebook: $y_0 = +1, y_1 = -1$
   0.1. GLA
   0.2. EXTEND      /* $N \leftarrow 2$ */
   0.3. GLA
```

```
  0.4. Calculate distortion $\Delta^0$
  0.5. best-config.-reached = 0
1. while $N < N_{max}$
  1.1. while best-config.-reached = 0 do
      1.1.1. SA
      1.1.2. GLA
      1.1.3. Calculate distortion $\Delta^m$
      if $(\Delta^m - \Delta^{m-1})/\Delta^{m-1} < \epsilon$,
        best-config.-reached = 1
  1.2. EXTEND      /* $N \leftarrow 2 \times N$ */
```

The initial codebook and the trellis are generated as described before in the text. For a given codebook, the trellis structure is optimized using SA, and for this structure, the codebook is modified using GLA. Then for the new codebook, the trellis structure is reoptimized. The process is continued until the system reaches an equilibrium, with respect to the SA criteria. Having found the optimum trellis for constraint length k, the trellis is extended to a constraint length $k + 1$ trellis by the extension algorithm described above. Then, SA and GLA are run iteratively in the same way for the extended trellis. In this way, the algorithm automatically designs near-optimum trellis coding systems with increasing constraint lengths for a given input sequence whose statistics are not known.

Chapter 5

SIMULATION RESULTS

To test the performance of the trellis decoder design algorithm we introduced, several coding systems based on finite-state machine decoders such as trellis waveform coder, finite-state vector quantizer, predictive trellis waveform coder, and trellis coded quantizer were designed for coding independent identically distributed (i.i.d.) Gaussian, Gauss-Markov (autoregressive Gaussian), and speech model sources. These sources are of high practical and theoretical interest and are commonly used in the source coding literature for testing the performance of quantization systems. The results obtained via simulations are compared with the results of other work published in the literature.

Gaussian sources used in the simulations were generated by Knuth's ***algorithm-P*** [39] using the uniformly distributed random sequence generated by the random number generator, *random*( ), from the mathematical library of the SunOS operating system, Release 4.1, by Sun Microsystems Computer Corporation.

To be able to compare the performance of different design methods, we calculated *signal to quantization noise ratio* (SQNR), for each decoder designed via simulations. SQNR is a commonly used measure of distortion due to quantization and is defined as,

$$SQNR = -10 \log_{10} \frac{\triangle}{\sigma_x^2}, \tag{5.1}$$

where $\triangle$ is the total distortion calculated by the square of the Euclidean distance, that is, $\triangle = \sum_n (\hat{X}_n - X_n)^2$, and σ_x^2 is the source power, $\sum_n X_n^2$.

5.1 Trellis Waveform Coding

5.1.1 Memoryless Gaussian Source

The memoryless Gaussian source consists of samples drawn independently from a Gaussian probability density with zero mean and unit variance. The distortion-rate function for this source evaluated at $R = 1$ yields the bound SQNR = 6.02 dB [44]. The 1-bit Lloyd-Max scalar quantizer [44] has SQNR = 4.40 dB.

The trellis waveform coder is designed using SA+GLA on a memoryless Gaussian training sequence of 10,000 samples. Then, the performance of the trained decoder is measured by coding a test sequence different from the training sequence but whose distribution and length are the same. The simulation results are given in Table 5.1 and in Figure 5.1 along with the results obtained by Linde and Gray [42], Stewart *et al.* [20], Pearlman [43] and Freeman *et al.* [44].

Linde and Gray [42] state that the problem of designing a good time-invariant tree-coding data compression system is equivalent to that of finding a good low rate "fake process" for the original source. The fake process problem is basically the problem of designing a filter which, when driven by a discrete uniform, i.i.d. process, produces an output that "looks like" the process that one wishes to compress. Following their statement they suggested a *scrambling function decoder* (SFD) and Viterbi encoding. The encoder finds the sequence of codewords which best describes the input data by carrying out a trellis search, and the corresponding index sequence is released to the channel. The decoder receives the channel symbols through a shift register and at each decoding instant applies the sum of the contents of the register to a nonlinear filter (scrambling function) to produce the reproduction symbols.

Stewart *et al.* [20] designed trellis waveform coders with fixed next-state function via GLA on a training sequence of 20,000 samples. They used table-lookup shift register decoders with random codewords (as the initial guess). Then, they tested the performance on data from outside the training sequence.

	SA+GLA		PA64		CGA		SFD	GLA
K	train	test	train	test	train	test	test	test
2	4.65	4.65	4.70	4.70	4.85	4.70		4.40
3	5.09	5.06	5.05	4.85	5.13	5.07	4.45	4.70
4	5.23	5.15	5.20	5.03	5.35	5.18	4.90	4.92
5	5.36	5.21	5.40	5.05	5.47	5.30	5.00	5.07
6	5.49	5.31	5.70	5.15	5.55	5.42	5.00	5.12

Table 5.1: SQNR [dB] results for the memoryless Gaussian source. K: constraint length, SA+GLA: trellis waveform coder with simulated annealing and generalized Lloyd algorithm, PA64: Powell's 1964 algorithm, CGA: conjugate gradient algorithm, SFD: Linde and Gray's scrambling function decoder, GLA: generalized Lloyd algorithm.

Pearlman [43] approached the design of trellis source coders through rate-distortion theory for constrained size reproduction alphabets. Solving the constrained rate-distortion function, he obtained reproduction levels. Then, he constructed sliding-block codes by distributing the reproduction values over one level of the trellis, the structure (next-state function) of which is fixed. He reported simulation results for large trellises of 256 and 512 states.

Freeman *et al.* [44] viewed the encoder simulation as the evaluation of an objective function of the code assignment variables. They used two optimization methods due to Powell. The first one is a nonderivative descent method called Powell's 1964 algorithm (PA64) [45] and the second is a gradient descent method called Powell's conjugate gradient algorithm (CGA) [46], [47]. Each of these methods performs a series of line searches in conjugate search directions. A line search is in effect the minimization with respect to one parameter, that gives the position along a straight line in the space of independent variables [44].

The important difference of our approach from these works is that, while they keep the next-state function fixed, we optimize it. Looking at Table 5.1, first note the improvement from GLA to SA+GLA. This improvement is due to the optimization of the trellis structure and is larger for larger constraint lengths. As the constraint length increases, the size of the configuration space increases exponentially, therefore, the fixed trellis structure (or the next-state function) used by GLA [20] in design becomes less and less probable to be "the best" structure. Also, as the size of configuration space is increased there

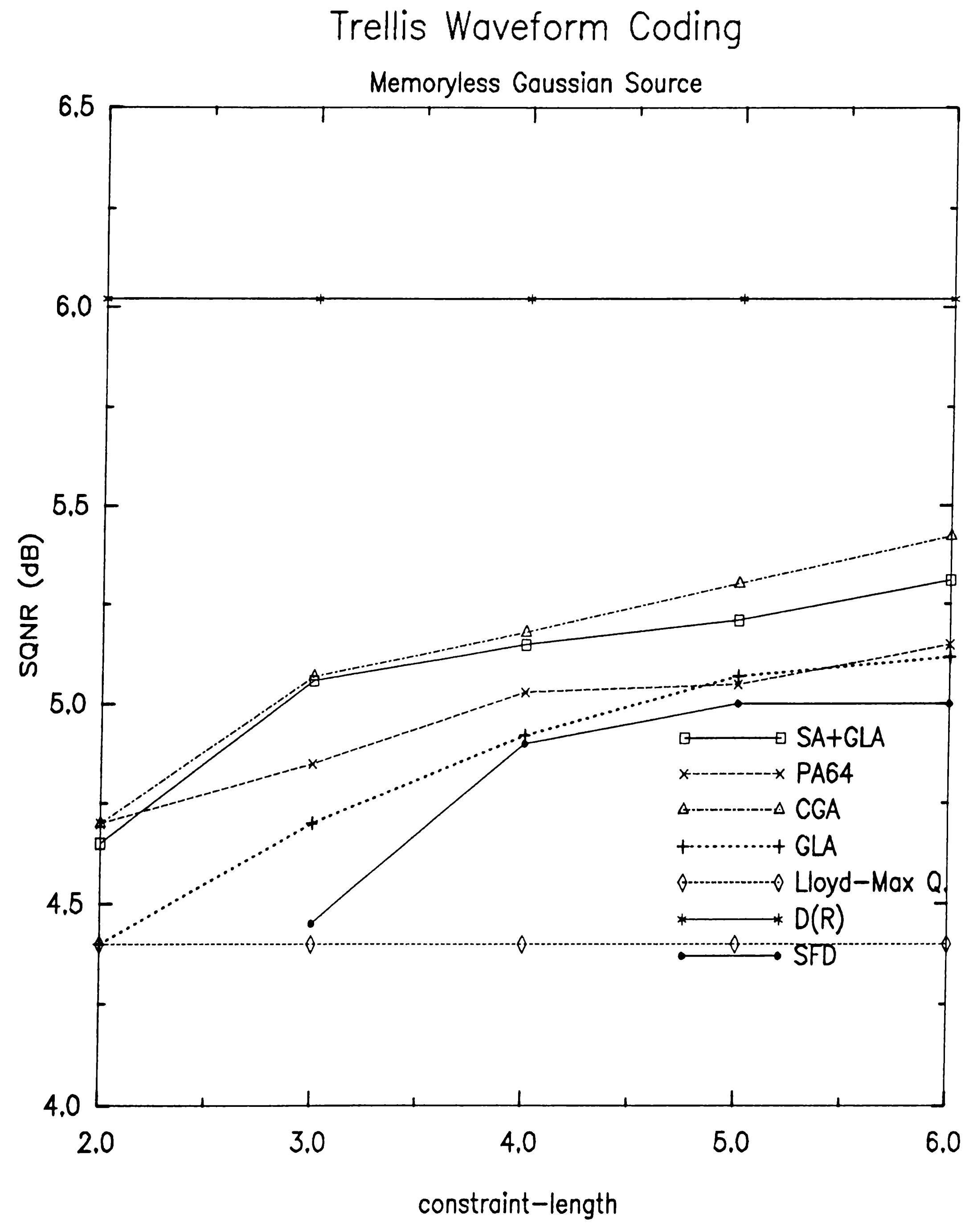


Figure 5.1: Trellis waveform coder, SQNR results for Gaussian i.i.d. source, SA+GLA: Trellis waveform coder with simulated annealing and generalized Lloyd algorithm, PA64: Powell's 1964 algorithm, CGA: conjugate gradient algorithm, SFD: Linde and Gray's scrambling function decoder, GLA: generalized Lloyd algorithm.

are more "good" structures introduced, and therefore SA has the chance to choose a "good" trellis from a wider set. These two facts explain the higher performance improvement for larger constraint-lengths.

As stated above, GLA is not the only method for improving the output map. The improvement possible with codebook design algorithms other than GLA can be judged by comparing the trellis waveform coding system results of Freeman *et al.* with GLA results: CGA performs much better than GLA. Noting the improvement from the GLA results to the SA+GLA results, and noting the improvement from the GLA results to the CGA results, one can speculate that using the conjugate gradient algorithm instead of GLA for codebook design and employing SA for trellis structure optimization, i.e., using SA+CGA, better performance can be obtained for memoryless Gaussian sources.

When test results are compared, SA+GLA outperforms Powell's 1964 algorithm PA64 and Linde and Gray's scrambling function decoder. Also, Pearlman's results for $K = 9$ and $K = 10$ are 5.18 dB and 5.21 dB respectively which SA+GLA outperforms with only a $K = 5$ trellis. On the other hand, conjugate gradient algorithm CGA gives the best results among all, although our results (SA+GLA) are almost the same for constraint-lengths $2, 3, 4$, and are only slightly worse (about 0.1 dB) for $K = 5, 6$.

5.1.2 First Order Gauss-Markov Source

The advantage of a waveform coding system with memory, such as the trellis waveform coder, is in getting high performance in encoding sources with memory. Therefore, a better source to test the performance of our algorithm is the Gauss-Markov autoregressive source. To this end, in this work, trellis waveform coding systems of different constraint lengths and of rate one were trained using SA and GLA by a first order Gauss-Markov source $\{X_n\}$ defined by

$$X_n = aX_{n-1} + W_n \qquad n = 1, 2, \dots \tag{5.2}$$

where W_n is a white and zero-mean Gaussian time series, and $a = 0.9$. This source was chosen since it is a common model of real data and it is widely used in comparing data compression systems [4]. The $D(R)$ bound for this source is 13.2 dB [19].

For constraint lengths of 2–8, signal-to-quantization-noise ratios (SQNR) were computed. Then the system was tested using a test sequence with the same statistics. In Table 5.2 and Figure 5.2 the SQNR values are given (SA+GLA) together with the results of Stewart *et al.* (GLA) [20]. Results obtained using SA are better than those of [20], especially for structures with small constraint lengths. The difference in performance comes from the optimization of the trellis structure (or the next-state function). For constraint-lengths 3, 4 and 5 the improvement is more than 1 dB (for $K = 4$ is 1.5 dB), which is significant since the GLA performance is within 2.5 dB of $D(R)$ bound for $K = 5$. With increasing K, performance improvement decreases to about 0.3 dB for $K = 8$, which is again quite significant since the GLA performance is within 1 dB of $D(R)$ bound for this constraint-length. We have not seen any significant improvement for $K = 2$; this is because the configuration space for this constraint-length is very narrow, and apparently the shift-register trellis used by Stewart *et al.* [20] is a good trellis among few possible ones. With the increasing constraint-length (with the widening configuration space), the improvement increases and becomes maximum at $K = 4$. The improvement for this constraint-length is about 1.5 dB. For larger constraint-lengths the improvement decreases since large codebooks already provide good precision for quantization and improvement due to SA becomes less significant when compared with the improvement with enlarged codebooks. Actually, during simulations it was observed that for $K > 6$ perturbing the trellis with SA does not lead to significant improvements. This is partly due to the fact that temperature is quite low at these instants and that since the optimization of trellis structure at a certain constraint-length K begins with the extension of the optimum trellis structure for constraint-length $K - 1$ trellis, and therefore the initial structures can be expected to be already good structures for high constraint-length trellises.

As can be seen from Table 5.2, the difference between the training sequence and test sequence SQNRs is increasing for increasing constraint-length. For $K = 8$ the difference is almost 0.35 dB for SA+GLA results, which points the inadequate training of the source, that is, we need longer training sequences or equally longer trellises. But, we cannot increase the trellis length indefinetely since this is accompanied with increasing storage requirements and longer trellis searches during encoding which means longer execution times. In Chapter 2, Section 4, we touched upon these practical difficulties in using the Viterbi algorithm and suggested a way to get around this inconvenience, which was to perform a "truncated search" instead of full trellis search performed by

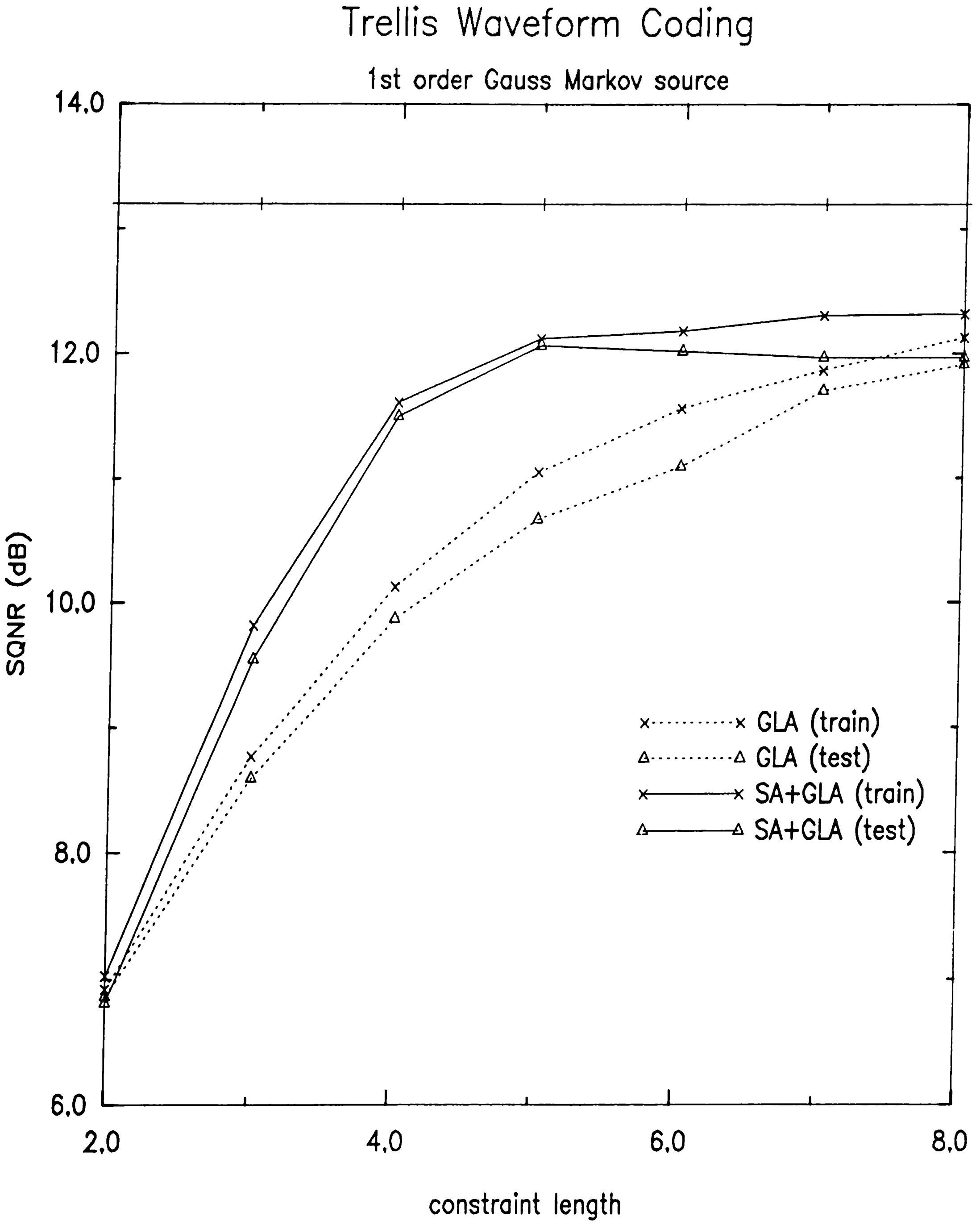


Figure 5.2: Trellis waveform coder, SQNR results for first order Gauss-Markov source, SA+GLA: Simulated Annealing and Generalized Lloyd Algorithm, GLA: Generalized Lloyd Algorithm only.

	SA+GLA		GLA	
K	train	test	train	test
2	7.03	6.81	6.92	6.86
3	9.82	9.55	8.77	8.59
4	11.61	11.50	10.13	9.87
5	12.12	12.06	11.05	10.67
6	12.18	12.02	11.56	11.09
7	12.31	11.97	11.87	11.70
8	12.32	11.97	12.13	11.91

Table 5.2: SQNR [dB] results for the first order Gauss-Markov source. K: constraint length, SA+GLA: simulated annealing and generalized Lloyd algorithm, GLA: generalized Lloyd algorithm only.

	Truncation Depth:						
K	full search	$100K$	$50K$	$10K$	$5K$	$3K$	$2K$
2	5.482	5.482	5.482	5.476	5.448	5.358	5.122
3	8.709	8.709	8.709	8.683	8.592	8.449	8.235
4	11.304	11.304	11.304	11.294	11.153	10.706	10.183
5	11.550	11.550	11.550	11.539	11.315	10.821	10.359
6	12.103	12.103	12.103	12.057	11.732	11.194	10.805

Table 5.3: SQNR [dB] results for the first order Gauss-Markov source with different truncation depths. K: constraint length, TD: truncation depth.

the Viterbi algorithm. This modification in trellis search would allow us to store only a part of candidate paths (typically of length $10 \times K$) and therefore enable us to use larger training sequences such as 50,000 or 100,000 samples. This argument needs experimental justification: we should show that both full search Viterbi algorithm and its modified version performing truncated search give the same results and there is no performance loss. This is done by running SA+GLA having a full-search encoder and testing the optimum trellis (with optimum codebooks and state-transitions) obtained by this run with the same Gauss-Markov source and truncated-search encoder with various search depths. The results are given in Table 5.3.

As can be seen, there is no performance loss at all for VAs with truncation depths of 10 times the constraint length and above. For small constraint lengths, truncation depths of even 5 times the constraint length are satisfying.

But, for truncation depths shorter than 5 times the constraint length, the performance difference becomes significant. Therefore, we are justified to use the modified Viterbi algorithm with a search depth of $10 \times K$.

5.2 Vector Trellis Waveform Coding

As discussed in Chapter 2, Section 2, coding symbols in blocks rather than one by one is expected to yield higher performance since there is a higher degree of freedom in choosing decision regions for quantization in block coding and since this enables exploitation of the correlation between samples. Simulation results for coding symbols in pairs are given along with the results for scalar coding using simulated annealing and the generalized Lloyd algorithm in Table 5.4 and Figure 5.3. Although there exists a significantly large difference between training and test sequence results, indicating the insufficient size of the training sequence, it can still be concluded from this table that vector coding results are better than scalar coding results. However, the difference is not significantly large. This observation can be interpreted to suggest that scalar trellis waveform or delayed-decision coding by itself exploits the correlation between samples quite well, and there is not much left for improvement by vector trellis waveform coding.

A related work is Bei and Gray's labeled state vector trellis encoding system [38]. Their approach is to design a FSVQ using the methods introduced by Foster *et al.* [7] and use this decoder with Viterbi encoding in trellis waveform coding. A comparison of our results with those of Bei and Gray's (for which two related data points are shown in Table 5.4) indicate *(i)* the observation that the performance improvement while increasing the vector length for this source is limited is shared in [38], and *(ii)* the system and the design technique proposed here outperform that in [38].

5.3 Finite-State Vector Quantization

As noted above, FSVQ is a special case of TWC where the search length of the encoder (Viterbi algorithm) is reduced to one stage. Therefore, our arguments for truncated-search Viterbi algorithm in Chapter 2, Section 1.2 applies to

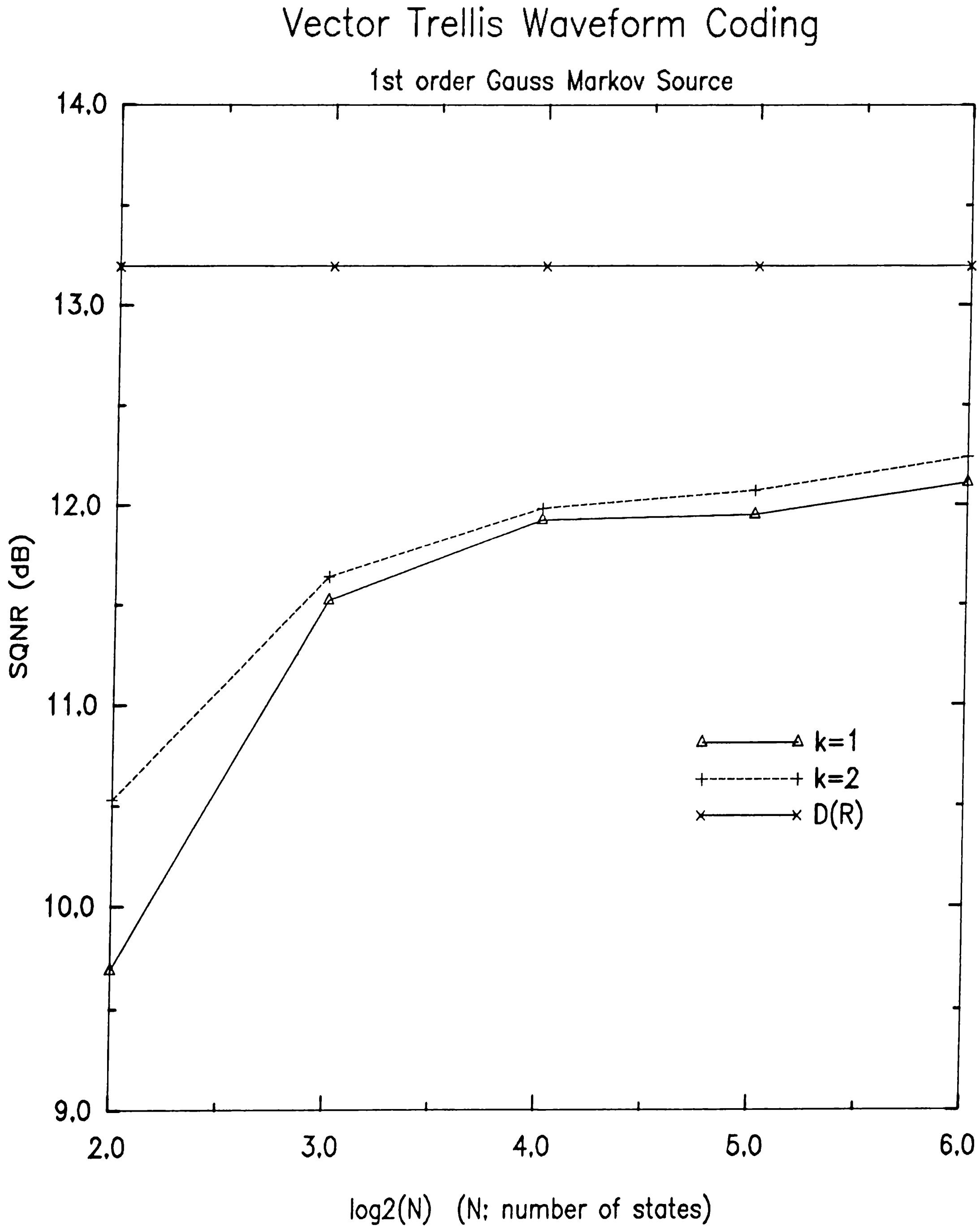


Figure 5.3: Vectoral TWC vs scalar TWC, first order Gauss-Markov source

	SA+GLA				LSVTE			
	$k=1$		$k=2$		$k=3$		$k=4$	
N	train	test	train	test	train	test	train	test
4	9.83	9.53	10.62	10.44				
8	11.59	11.44	11.74	11.54				
16	11.95	11.90	12.04	11.72				
32	12.00	11.90	12.30	11.84	11.4	11.4		
64	12.25	11.97	12.54	11.93			11.7	11.6

Table 5.4: SQNR [dB] results for scalar and vector trellis waveform coding where the systems with $k = 1$ and $k = 2$ are designed using SA+GLA and results for $k = 3$ and $k = 4$ are those of the labeled state vector trellis encoding system. N: number of states, k: vector length, LSVTE: labeled state vector trellis encoding system.

FSVQ, with a search depth of one vectorsize. Looking at Table 5.3, we noted before that the performance of TWC is the same for truncation depths $10 \times K$, but the loss in performance becomes significant for truncation depths less than $5 \times K$, which is due to the fact that the truncated search Viterbi algorithm cannot perform an optimal search for this short constraint lengths. Therefore, it is apparent that the performance of FSVQ will be significantly less than the performance of TWC. Yet still, FSVQ is important practically, since it has much less computational complexity than TWC and there is only one vector-size delay involved.

We obtained simulation results first designing the finite-state vector quantizer for the first order Gauss-Markov source and then testing the design with a source from outside the training data. For designing FSVQ's with vector-lengths 1, 2, 3, and 4, a fixed training sequence of length 20,000 samples is divided into blocks of 1, 2, 3, and 4 sample lengths respectively, and the resulting vector sequences are used for training the FSVQ.

Our results are given in Table 5.5, and in Figure 5.4 together with the results obtained by Foster *et al.* [7] and memoryless vector quantization results. Foster *et al.* used a method based on a heuristic approach called *Omniscient Labeled Transitions* (OLT) for the design of the next-state map, and GLA for the codebook design. Our results show that SA+GLA performs much better than VQ and generally better than OLT although not much better. OLT is known to yield the best results obtained in FSVQ so far [4]. Our results show that

SA+GLA is a contender for performing better, and we believe the performance improvement may be more significant for more complicated sources, such as speech samples. Also, considering the results of Bei and Gray [38] cited in Chapter 5, Section 5.1, who used the trellis designed for FSVQ by the methods of [7] for TWC with Viterbi encoding, as a plagiarized decoder, we can conclude that SA+GLA is generally a better algorithm for designing finite-state machine decoders than the methods of [7].

	SA+GLA		VQ		OLT	
k	train	test	train	test	train	test
1	9.46	9.50	4.42	4.40	9.21	9.14
2	10.65	10.81	7.90	7.86	11.04	10.90
3	11.17	11.26	9.24	9.17	11.22	11.08
4	11.53	11.38	10.15	10.07	11.34	11.12

Table 5.5: SQNR [dB] results for 8-state FSVQ and VQ for the first order Gauss-Markov source. k: vector length, SA+GLA : FSVQ with simulated annealing and generalized Lloyd algorithm, VQ: memoryless vector quantizer, OLT : FSVQ with omniscient labeled transition design method.

5.4 Predictive Trellis Waveform Coding

In [19], Ayanoğlu and Gray incorporated prediction into trellis waveform coding, the idea being similar to predictive vector quantization [4], with a trellis encoder replacing the memoryless vector quantizer and a finite-state machine decoder replacing the vector quantizer codebook. The predictive trellis coding system they used and the design algorithm they suggested were described in Chapter 2, Section 5. Their approach in designing predictive trellis waveform coder was to keep the next-state function fixed, improving the codebooks iteratively with GLA and regularly updating the prediction coefficients according to modified state-codebooks. Through simulations, they designed predictive trellis waveform coders [19] and compared the SQNR results with the SQNR results for nonpredictive trellis waveform coders designed with GLA [20]. The results they report show that, there is a very significant performance improvement introduced by incorporating prediction to the encoding process, which is about 4 dB for $K = 2$, 2.5 dB for $K = 3$ and generally more than 1 dB for higher constraint-lengths.

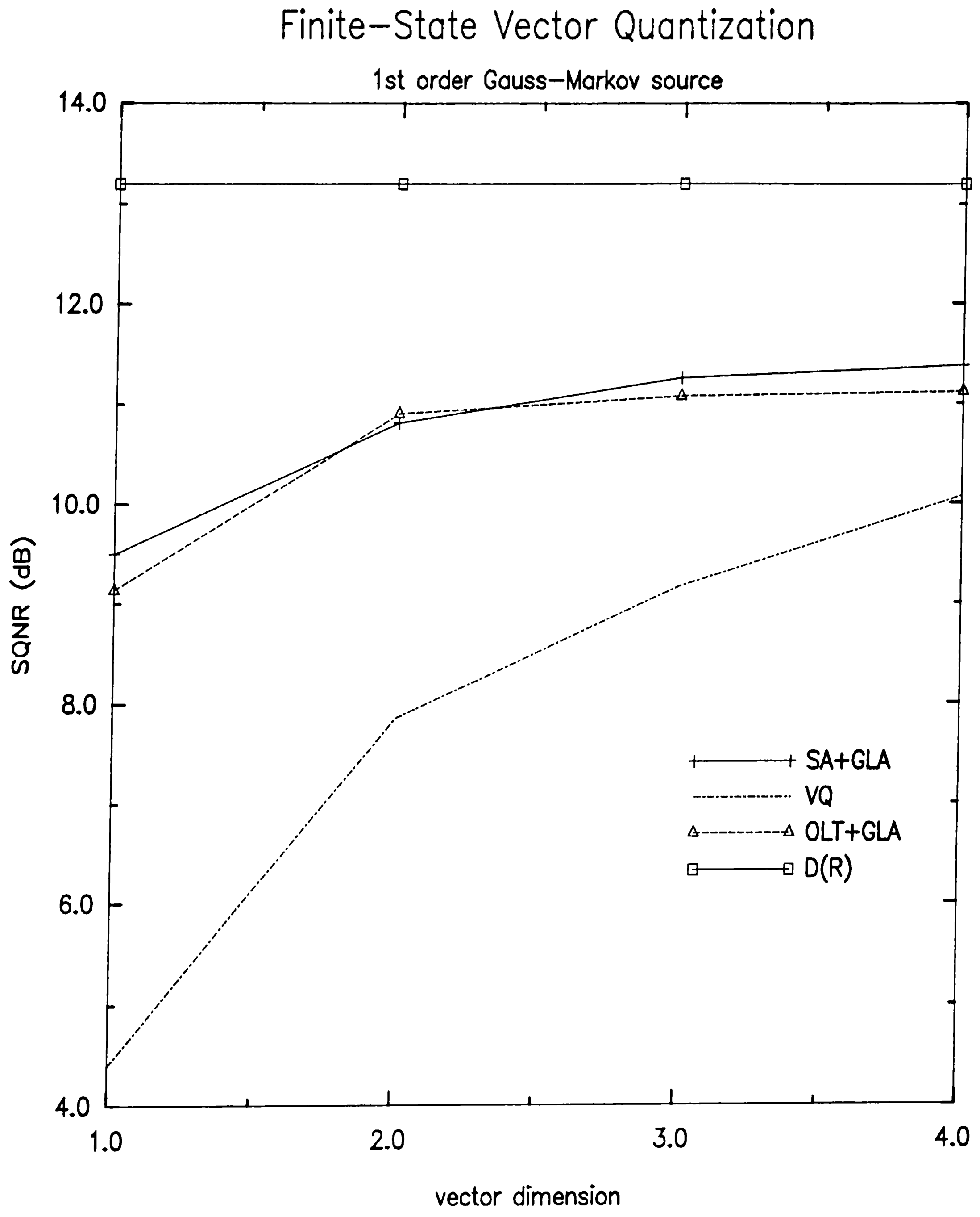


Figure 5.4: Finite-state vector quantization, SQNR results for first order Gauss-Markov source, 8 state trellis, SA+GLA : FSVQ with simulated annealing and generalized Lloyd algorithm, VQ: memoryless vector quantizer, OLT+GLA : FSVQ with omniscient labeled transition design method and generalized Llod algorithm.

Following our arguments in the previous sections of the thesis, we can conjecture that, as in the nonpredictive case, optimizing also the next-state function we can obtain even better performance with the predictive system. Considering the performance improvement SA provided over the performance of GLA, we incorporated SA into the predictive system of Ayanoğlu and Gray: in the design algorithm we suggested for TWC in Chapter 4, Section 3, we inserted the design algorithm of Ayanoğlu and Gray [19] in place of GLA. The resulting algorithm, at each constraint-length, starts the design process with a fixed trellis, and then perturbs it into new trellises using SA. When SA exit criterion is satisfied, the codewords are modified by GLA and the predictor coefficients are updated according to new codewords (output map) and new next-state map. This process is repeated until no significant improvement is observed on consequent SA terminations. The initial trellis for each constraint length K is obtained by extending the optimum trellis of contraint-length $K-1$.

5.4.1 First Order Gauss-Markov Source

A PTWC is designed using the method just described on a first order Gauss-Markov source training sequence of 10,000 samples and the design is tested with a sequence of the same length. The SQNR results for nonpredictive and predictive trellis coders designed by GLA and SA+GLA are given below in Table 5.6, and Figure 5.5.

	nonpredictive				predictive			
	GLA		SA+GLA		GLA		SA+GLA	
K	train	test	train	test	train	test	train	test
1	4.35	4.28	4.35	4.28	10.01	9.65	9.98	10.11
2	6.92	6.86	7.03	6.81	11.08	10.73	11.08	11.21
3	8.77	8.59	9.82	9.55	11.53	11.18	11.61	11.74
4	10.13	9.87	11.61	11.50	11.84	11.47	12.09	12.20
5	11.05	10.67	12.12	12.06	12.18	11.83	12.26	12.33
6	11.56	11.09	12.18	12.02	12.38	11.96	12.38	12.45
7	11.87	11.70	12.31	11.97	12.52	12.52	12.41	12.50

Table 5.6: SQNR [dB] results for the first order Gauss-Markov source. K: constraint length, SA+GLA: simulated annealing and generalized Lloyd algorithm, GLA: generalized Lloyd algorithm only.

The GLA on nonpredictive system (NS), and GLA on predictive system

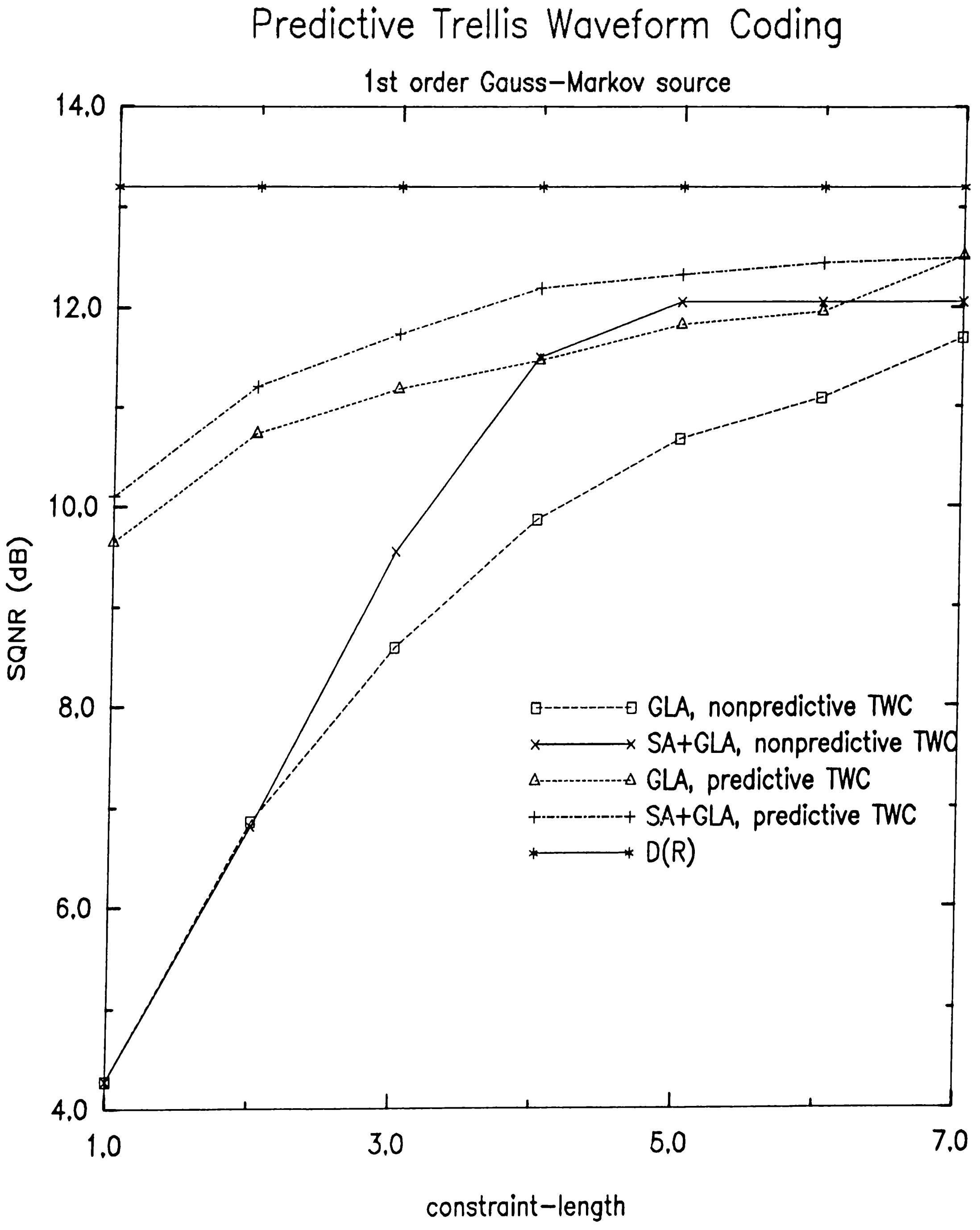


Figure 5.5: Predictive trellis waveform coder, SQNR results for first order Gauss-Markov source.

(PS) results in Table 5.6 are reproduced from [19]. Comparing these two systems, we see the significant improvement provided by the predictive system as reported in [19]. The performance improvement from GLA on nonpredictive system to SA+GLA on nonpredictive system was discussed in Section 5.1 in the context of TWC. When the results of SA+GLA on nonpredictive system and GLA on predictive system, that is, our TWC results and Ayanoğlu and Gray's PTWC results are compared, their results are significantly better for structures with small constraint lengths ($K = 1, 2, 3$). This is expected since the predictive system has a higher system complexity. However, for higher constraint lengths our results are quite close to those of [19], so that the nonpredictive system once again becomes attractive. Next, comparing our SA+GLA results for nonpredictive and predictive systems, we observe a significant performance improvement provided with the predictive system (especially for $K = 1, 2, 3, 4$) which paralels the results of Ayanoğlu and Gray [19]. Finally, when GLA and SA+GLA results for the predictive system are compared, the superiority of SA+GLA is obvious. The SA+GLA results are good since even with a constraint length of 4, the algorithm shows a performance which is within 1 dB of the $D(R)$ bound.

5.4.2 Speech Model Source

Another source of importance is the *speech model source*. In [49] Wilson and Husain used the speech data obtained by McDonald [50] to obtain a third-order Gauss-Markov model for speech. The model is described by the difference equation

$$X_n = 1.75X_{n-1} - 1.22X_{n-2} + 0.301X_{n-3} + W_k \tag{5.3}$$

where W_k's are independent, identically Gaussian distributed with zero mean. The variance σ_W^2 is 0.097. The process X_k is stationary with unit variance. The $D(R)$ bound for this source is calculated to be 14.4 dB at rate 1 bit/sample [44]. The SQNR of 1 bit DPCM for this source is 8.4 dB. Simulation results are given in Table 5.7 and Figure 5.6. with the results of Ayanoğlu and Gray's predictive system [19], CGA [44] and GLA.

These results indicate that predictive systems show significantly better performance over nonpredictive systems. This is expected since the source is more complex (third order) than the previously used sources, and a higher order linear predictor is used. Second, the performances of nonpredictive GLA

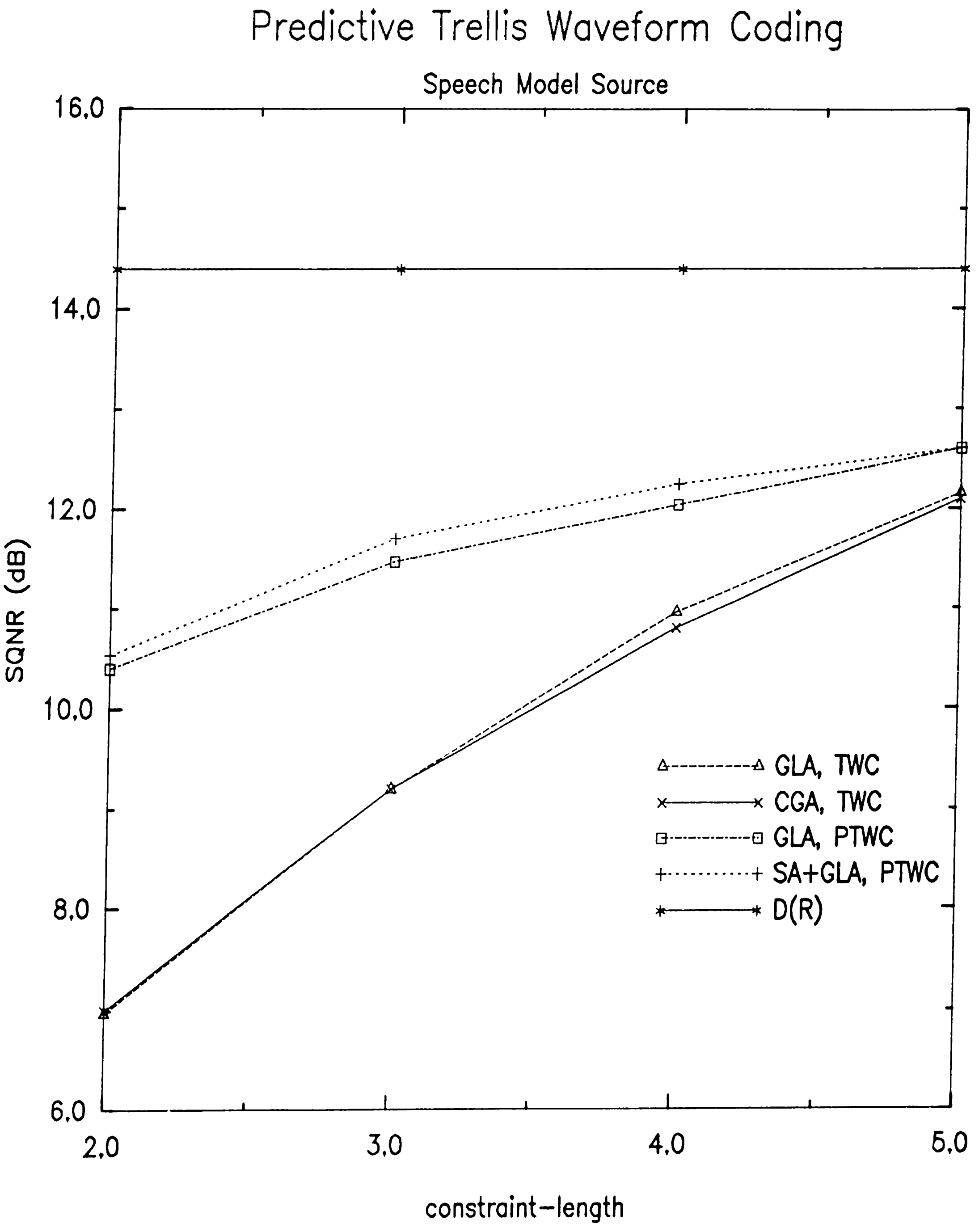


Figure 5.6: Predictive trellis waveform coder, SQNR results for speech model source.

	nonpredictive		predictive	
K	GLA	CGA	GLA	SA+GLA
2	6.97	7.00	10.40	10.53
3	9.20	9.20	11.47	11.70
4	10.96	10.80	12.04	12.25
5	12.16	12.10	12.60	12.60

Table 5.7: SQNR [dB] results for the speech model source. K: constraint length, SA+GLA: simulated annealing and generalized Lloyd algorithm, CGA: Powell's conjugate gradient algorithm, GLA: generalized Lloyd algorithm only.

and CGA are almost the same. Remember that for the memoryless Gaussian source, the performance of CGA was significantly better than the performance of GLA. This observation shows that we cannot generalize our argument about CGA and GLA on the memoryless Gaussian source to other sources. The performance of the algorithms is dependent on the source used. Third, predictive system designed with SA+GLA has a higher performance than the predictive system designed with GLA only. This result points out the potential of improving the performance of predictive trellis waveform coders by optimizing the next-state function.

5.5 Trellis Coded Quantization

5.5.1 Memoryless Gaussian Source

As was discussed in Chapter 2, Section 6, Marcellin and Fischer suggested TCQ in [51], basing their arguments on an observation made in alphabet-constrained rate distortion theory for the uniform i.i.d source. In [51], Marcellin and Fischer also noted that they had no intuitively pleasing distance property arguments to justify using TCQ for memoryless Gaussian source but alphabet constrained rate distortion theory indicated that a substantial performance increase over the Lloyd-Max quantizer was possible. The simulation results they obtained following this observation for memoryless Gaussian sources are given along with the simulation results for uniform i.i.d. and Laplacian i.i.d. sources in [51]. Among these, the memoryless Gaussian results are of interest to us since we have already tested the performance of our design approach on this source

R	TCQ	TCQ(+SA)	TWC(SA+GLA)	TWC(CGA)	L-M Q.	$D(R)$
1	4.54	4.57	4.65	4.85	4.40	6.02
2	10.06	10.06	10.19		9.30	12.04

Table 5.8: Comparison of trellis coders for Gaussian i.i.d. source, $N = 4$, L-M Q.: Lloyd-Max quantizer, CGA: Conjugate gradient algorithm

in the previous sections.

Marcellin and Fischer used the Ungerboeck trellis structure described in Chapter 4, Section 6 as the next-state function for the rates $R = 1, 2, 3$. The output map was constructed by assigning the $R + 1$ bits/sample Lloyd-Max output points to the trellis branches according to Ungerboeck's branch labeling rules [52]. They report that although the SQNR results for the Gaussian i.i.d. source were quite higher than the Lloyd-Max quantizer results, the results were still far away from the $D(R)$ bound. To improve performance, they developed a training sequence based numerical optimization procedure for output alphabet design and they obtained better results with this algorithm. They also report that the performance diverged away from the distortion rate function as the rate growed. Therefore, they examined all the trellises other than the one described to see whether there are other trellis structures fitting better to TCQ, but they report that little could be gained over Ungerboeck's trellises. In several cases they found trellises that performed better than Ungerboeck's, but the improvement was insignificant. To observe if any improvement can be gained by SA, we first constructed a trellis following the procedure they gave for TCQ, obtained the SQNR value for this trellis, and then perturbed the trellis structure with SA and obtained SQNR values for the new trellises. The SQNR results for "plain" TCQ system, TCQ system with SA applied, Lloyd-Max quantizer and rate-distortion bound for $R = 1$, $R = 2$ are given in Table 5.8 together with our SA+GLA results for TWC.

Results show that almost no improvement is gained due to changing the next-state map. But some improvement is seen when the output points are trained. Taking a look at our previous TWC results for the memoryless Gaussian source, we also see that CGA gives much better performance than "plain" TCQ although with a price of higher computational complexity. These results draw our attention to two facts: First, for rate R encoding, rate $R + 1$ Lloyd-Max output points are not the best choices as codewords for memoryless

Gaussian sources. Better output maps can be obtained with design algorithms like GLA and CGA. Second, as was discussed in Chapter 2, trellis coding efficiently exploits the correlation between the samples, that is its real success is in coding sources with memory. Since Gaussian i.i.d. source does not have memory, the trellis structure supplies no important advantages, therefore, changing the next-state map does not affect the performance significantly.

To verify the last statement more strongly an exhaustive search was performed over the possible $R = 1$, $N = 4$ trellises, and it was observed that other than the pathological cases, the performance for most trellises were very close. It is worth noting that this exhausted search also showed that TWC with SA had found the best trellis.

5.5.2 First Order Gauss-Markov Source

As discussed before, the success of trellis source coding is in coding sources with memory. In [51], Marcellin and Fischer did not give any simulation results for coding sources with memory by TCQ. Therefore, we performed simulations for the first order autoregressive Gauss-Markov source, using the TCQ system introduced in [51] for memoryless sources. That is, the trellis is an Ungerboeck trellis with Lloyd-Max output points assigned to brances as described in [51]. Then, the trellis structure (next-state map) is perturbed into new structures via SA. The results are given in Table 5.9. The improvement from the trellis of [51] by SA is significant (about 0.7 dB), which shows that for coding first order Gauss-Markov sources there exist trellis structures significantly better than the one used in [51]. The improvement by SA was expected, because the source is highly correlated (correlation coefficient: 0.9) and the next-state map becomes important. But even with optimizing next-state function for the given output map, the performance is far from the $D(R)$ bounds. Our simulation results obtained for the first order Gauss-Markov source are also shown in the table to indicate the need for optimizing output map and next-state map together for the design of high performance trellis coder. The improvement from "plain" TCQ to TWC with SA+GLA is more than 4.5 dB for $R = 1$ and 6.5 dB for $R = 2$. Observingthe difference between the TCQ results and the TWC (GLA) results obtained by [20], we see that using Lloyd-Max quantizer output points does not guarantee any good codebook.

R	TCQ	TCQ(+SA)	TWC(SA+GLA)	TWC(GLA)	$D(R)$
1	4.71	5.45	9.55	8.59	13.23
2	6.85	7.65	13.58		19.25

Table 5.9: $N = 4$, first order Gauss-Markov source, $a = 0.9$

5.5.3 Predictive Trellis Coded Quantization

Marcellin and Fischer incorporated linear prediction to TCQ to form predictive TCQ (PTCQ). The search algorithm, that is the encoder, of their predictive system is similar to that of Ayanoğlu and Gray [19], but the design algorithm is different in that they do not train the codebooks and they do not update the predictor coefficients. For comparison we give simulation results for PTCQ of [51], PTWC of [19] and our PTWC with SA+GLA on first order Gauss-Markov source in Table 5.10 and on speech model source in Table 5.11. As can be seen, in both cases our PTWC with SA+GLA performs better than the other two systems. This superiority is due to optimizing the next-state map.

	PTCQ	PTWC(GLA)		PTWC(SA+GLA)	
K	test	train	test	train	test
3	11.19	11.53	11.18	11.61	11.74
4	11.60	11.84	11.47	12.09	12.20
5	11.89	12.18	11.83	12.26	12.33
6	12.13	12.38	11.96	12.38	12.45
7	12.22	12.52	12.52	12.41	12.50

Table 5.10: Predictive trellis coding results for first order Gauss-Markov source

K	PTCQ	PTWC(GLA)	PTWC(SA+GLA)
3	11.03	11.47	11.70
4	11.65	12.04	12.25
5	12.24	12.60	12.60

Table 5.11: Predictive trellis coding results for speech model source

5.5.4 Codebook Assignment to Branches in TCQ

During our simulations with SA on the initial Ungerboeck trellis used by Marcellin and Fischer, we noticed that some of the new trellises reached by a series of SA perturbations were Ungerboeck trellises used in [51], but the branch labelings were different. These trellises mostly had better (sometimes significantly better) performance than the initial trellis. Moreover, we noticed that the branch labelings satisfied Ungerboeck's branch labeling rules [52]. This observation seemed very interesting to us, since nothing was mentioned in the papers by Ungerboeck [52] and Marcellin and Fischer [51] about the possibility of existence of other trellises labeled according to Ungerboeck branch labeling rules but showing different performances.

To see the performances of those trellis coders we first produced all of the possible different branch labelings (codebook assignments) satisfying Ungerboeck's rules for the trellis with the structure of Figure 2.4. The supercodebook is generated and the set partitioning is done as described in [51]. These trellis coders are shown in Figure 5.7.

There are symmetries among some of the trellises: If the nodes 0, 1, 2 and 3 are relabeled as 3, 2, 1 and 0 respectively, trellis-e becomes trellis-a, trellis-f becomes trellis-b, trellis-g becomes trellis-c and trellis-h becomes trellis-d, that is, trellises in (a) and (e), (b) and (f), (c) and (g), and (d) and (h) are equivalent. Therefore, we need to consider only trellis-a, trellis-b, trellis-c and trellis-d.

Trellis-a is the one used by Marcellin and Fischer [51]. The difference between the performance of this trellis and the others was noticed while coding Gauss-Markov source with correlation coefficient $a = 0.9$, and before for memoryless Gaussian source ($a = 0.0$) we saw noted that the performances for most trellises (other than patological ones) were almost the same. For this reason we are tempted to look at the performance for various values of a. We first calculated SQNR for trellis-a, trellis-b, trellis-c and trellis-d on first order Gauss-Markov source with several correlation coefficients, on memoryless Gaussian source and on speech model source, using Lloyd-Max output points as codewords as in [51]. The results are given in Table 5.12.

The results show that trellis-b shows the best performance and trellis-d shows the worst among all. Moreover, we see that the performance difference

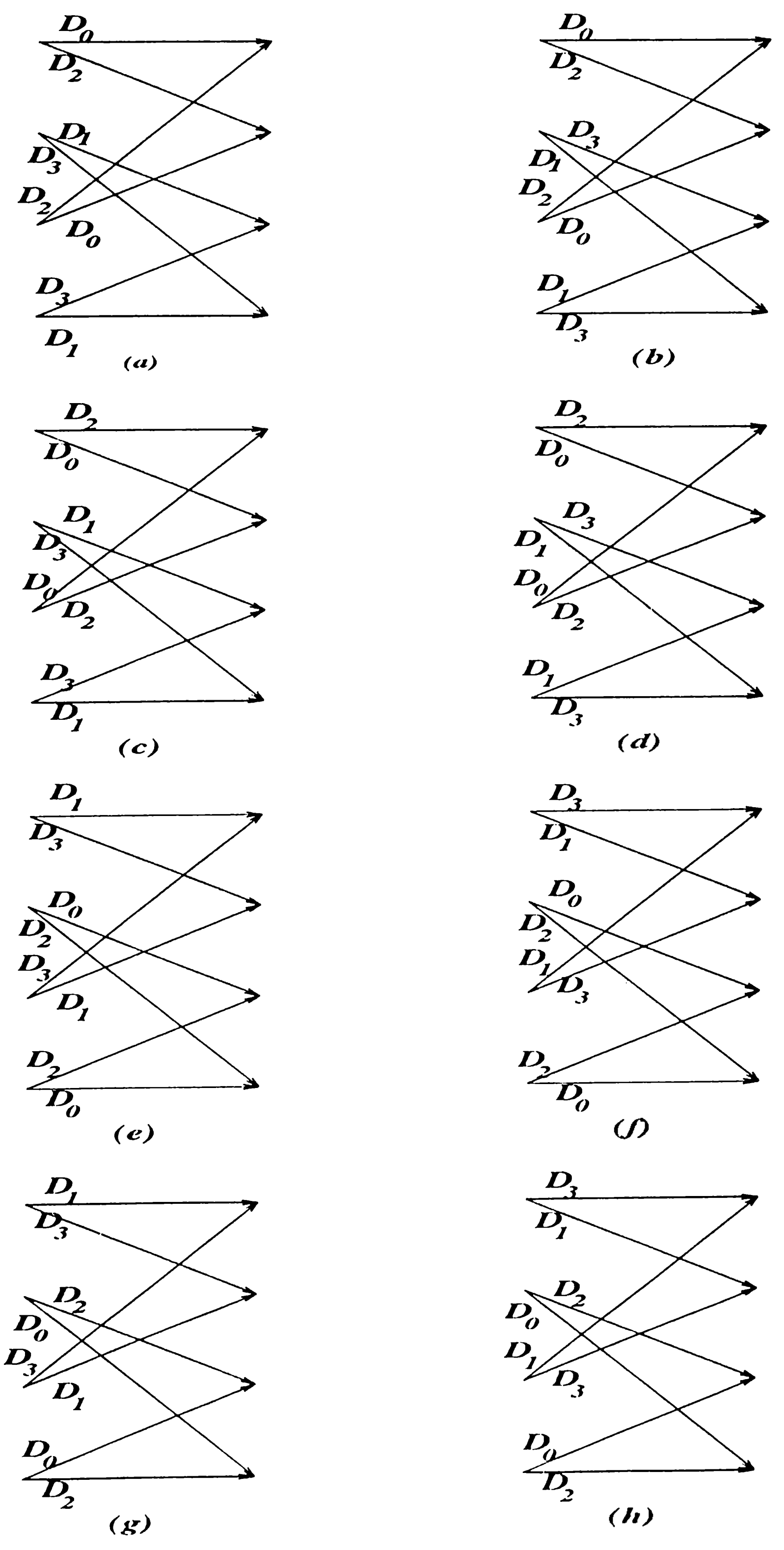

D_0 D_2 D_1 D_3 D_2 D_0 D_3 D_1
(a)
D_0 D_2 D_3 D_1 D_2 D_0 D_1 D_3
(b)
D_2 D_0 D_1 D_3 D_0 D_2 D_3 D_1
(c)
D_2 D_0 D_3 D_1 D_0 D_2 D_1 D_3
(d)
D_1 D_3 D_0 D_2 D_3 D_1 D_2 D_0
(e)
D_3 D_1 D_0 D_2 D_1 D_3 D_2 D_0
(f)
D_1 D_3 D_2 D_0 D_3 D_1 D_0 D_2
(g)
D_3 D_1 D_2 D_0 D_1 D_3 D_0 D_2
(h)

Figure 5.7: Ungerboeck trellises satisfying the branch labeling rules of Ungerboeck

a	trellis-a	trellis-b	trellis-c	trellis-d
0.95	4.71	5.54	4.60	4.41
0.9	4.58	5.30	4.53	4.35
0.7	4.69	5.01	4.61	4.60
0.5	4.70	4.82	4.66	4.71
0.0	4.71	4.71	4.71	4.71
speech	4.52	4.80	4.55	4.33

Table 5.12: $R = 1$, performance comparison of possible branch labelings for Ungerboeck trellis, Gauss-Markov sources

decreases with decreasing correlation coefficient and there is no performance difference for $a = 0.0$, the memoryless case.

Trellis-b shows always better performance than the other trellis coders. To gain more insight to the matter we concentrated on this coder and trellis-a, the trellis coder used in [51] and obtained data for negative values of correlation coefficient ($a < 0$), and for coding sequences with correlation coefficient, a with a decoder designed for a sequence with correlation coefficient $-a$, that with output map as the Lloyd-Max quantizer output points for source with $-a$. The results are given in Table 5.13.

On this data, we can make the following observations: The difference between the performances of the two trellises increases with increasing a.

With decreasing a, the performance improvement supplied by SA decreases.

For the positive and negative values of a, trellis-b shows always better performance. This is true even when the codebook designed for the source with $-a$ is used for source with a.

While the SQNR values for trellis-b increases significantly (from 4.65 dB to 5.35 dB) with increasing a, the SQNR values for trellis-a stays almost the same.

In the light of these results, we can draw the following conclusions about TCQ:

As was discussed in previous chapters, the design problem of trellis source coding is equivalent to the design problem of the next-state map and the output

	Q1 ($a = 0.9$)				Q2 ($a = -0.9$)			
a	t-a	+SA	t-b	+SA	t-a	+SA	t-b	+SA
0.9	4.61	5.34	5.34	5.34	4.58	5.26	5.26	5.26
-0.9	4.57	5.45	5.39	5.52	4.49	5.40	5.30	5.37

(a)

	Q1 ($a = 0.7$)				Q2 ($a = -0.7$)			
a	t-a	+SA	t-b	+SA	t-a	+SA	t-b	+SA
0.7	4.69	5.12	5.01	5.12	4.66	5.09	4.99	5.09
-0.7	4.63	5.16	5.00	5.08	4.62	5.13	4.99	5.13

(b)

	Q1 ($a = 0.5$)				Q2 ($a = -0.5$)			
a	t-a	+SA	t-b	+SA	t-a	+SA	t-b	+SA
0.5	4.69	4.91	4.79	4.91	4.67	4.88	4.78	4.90
-0.5	4.72	4.95	4.76	4.95	4.72	4.93	4.75	4.98

(c)

	Q1 ($a = 0.1$)				Q2 ($a = -0.1$)			
a	t-a	+SA	t-b	+SA	t-a	+SA	t-b	+SA
0.1	4.65	4.71	4.71	4.74	4.64	4.67	4.69	4.71
-0.1	4.67	4.69	4.69	4.71	4.66	4.69	4.67	4.69

(d)

Table 5.13: Trellis-a and trellis-b comparison (t-a: trellis-a, t-b: trellis-b), +SA: performance with SA on the trellis the SQNR of which is given in the previous column, Q1 and Q2 denote the quantizers with Lloyd-Max output points calculated for S1 (source 1) and S2 (source 2) respectively, Source 2 has a correlation coefficient that is negative of Source 1's.

map. Being a trellis source coder design approach, the TCQ technique suggests a fixed next-state map during trellis decoder design procedure: the next-state map is not optimized. The next-state map TCQ suggests is the Ungerboeck trellis which was shown in Figure 2.4.

For memoryless Gaussian source our results showed that this next-state map was fairly good, but it is one of the many good ones, it is not particularly the best trellis. As also noted in [20], the next-state map is not very important in coding memoryless sources with trellis coders, therefore, other than the pathological cases, most of the trellises would give approximately the same performance. This was verified above by the results of exhaustive search we performed for $R = 1$, $N = 4$ trellis. The Ungerboeck trellis, having a symmetric structure, is just one of the better ones.

TCQ's suggestion for output map design involves the generation of a supercodebook, partitioning this supercodebook into subsets and labeling the branches with indexes of the subsets, that is, assigning the subsets to the branches. As we noted even for memoryless Gaussian source, this choice of Lloyd-Max output points does not look like a good one, since we have shown the possibility of obtaining significantly better codebooks with GLA or CGA.

Set partitioning and branch labeling are done in a way to increase the distance between the codewords. These two approaches were borrowed from TCM where they are well justified. In TCM, set partitioning and branch labeling according to Ungerboeck's rules leads to the maximization of free distance between code sequences. This means that the code sequences are made as far as possible from each other, which decreases the probability of deciding on a wrong code in the decoder due to channel noise. Making codes robust to channel noise is a common goal in modulation and channel coding. But in quantization, as discussed in Chapter 1, it is assumed that the channel is lossless, and therefore the goal is not to design robust codes but to compress the data so that the redundancy is removed and communication can be done with less bits per sample. Therefore, maximizing the distance between the codewords is not a step towards better compression. Yet, for memoryless uniform sources, maximizing the distance between the codewords available at a state seems a intuitively good approach since this allows an even distribution of codewords for finer quantization. But, it is not that straightforward for sources with memory, since not codewords but codeword sequences become significant due to memory. During design, one should take into account consequent stages, not just one stage. Above we stated that the arguments of modulation do not carry to quantization. Even if such an analogy exists according to Marcellin and Fischer's arguments, this analogy can be only on the basis of memoryless sources. For sources with memory, carrying the idea of maximizing the distance between codes in modulation to trellis quantization leads to the maximizing the distance between available reproduction sequences. As showed by our simulations in Section 1.1 this chapter, there is no significant loss in performance if the Viterbi algorithm makes truncated search instead of full search, with a truncation depth of $5 \times K$ for small constraint lengths and $10 \times K$ for for higher constraint lengths. Then, following the above observation, for a trellis 4 states the codebooks should be assigned to trellis branches considering a section of trellis with 15 stages. Therefore, unlike branch labeling rules of TCM, or of TCQ on memoryless source who consider only one stage of trellis the branch labeling rules to be designed must take into account a long trellis section. This is not an easy task.

We can conclude that the rules of Ungerboeck do not carry to sources with memory. This fact is verified with our simulation results for Gauss-Markov sources. We have seen that there exist significantly better trellis structures and that optimizing next-state map and output map, substantial gains are possible.

Our last observation about different trellises satisfying Ungerboeck rules but having significantly varying performances, also point out the lack of analogy in this case with TCM.

Chapter 6

SUMMARY AND CONCLUSIONS

The main contribution of this thesis is the employment of simulated annealing (SA) for the optimization of the next-state map of the decoder for data compression systems based on finite-state machines, such as finite-state vector quantization, trellis waveform coding, predictive trellis waveform coding, and trellis coded quantization. A decoder design algorithm for the joint optimization of the output map and the next-state map is obtained by incorporating the generalized Lloyd algorithm (GLA), a well-known algorithm for codebook design, into design.

Simulation results were obtained for Gaussian sources such as Gaussian i.i.d., first order Gauss-Markov, and third order Gauss-Markov (speech model) sources. Comparison of these results with other related work in the literature shows *(i)* the need for optimization of the next-state map of finite-state machine decoders, and *(ii)* SA is very succesful when employed for this purpose.

During simulations, theoretical as well as heuristic methods were used for choosing the SA parameters. In most of the simulations, Metropolis loop lengths of $20 \times N$ or even $10 \times N$ sufficed to reach quasi-equilibrium where N is the number of states. For the selection of the initial temperature, Johnson's algorithm turned out to be a good method in almost all of the cases, but in some cases for the speech model source (third order Gauss-Markov source), this algorithm gave too low initial temperatures which had to be increased manually. As the cooling or temperature decrement function, geometric cooling was used. In almost all of the simulations for finite-state vector quantization and trellis waveform coding, the values between 0.8 and 0.9 seemed to be the ideal choices for the cooling coefficient. For predictive trellis waveform coding, cooling coefficients as small as 0.6 led to good cooling schedules. After each output map optimization with the GLA, and after each trellis extension, the temperature was multiplied with two different constants to increase the probability of moving out of local minima for the new structures. Experimentally the ideal values of these constants were found to be 5 and 3, respectively. The exit criterion was that program terminated when the relative improvement was below 0.001 which was also determined experimentally.

The main drawback of the SA+GLA is the computational complexity due to running the Viterbi algorithm for each new structure during SA and each new output map during GLA. It has been observed during the simulations that for high constraint length trellises the SA improvement is not very significant, most of the improvement is provided by the GLA. Following this observation, one can simply perform only GLA for high constraint lengths and speed up the design. Another way to speed up the design is to bring some restrictions to the state space such as a subset of the previously defined state space in Chapter 4, but for which it is more likely to obtain the optimum trellis structure. For example, intuitively, symmetric structures can be expected to give better performance. With this motivation we brought the following restrictions to the state-space: the state-space of all trellis structures with two branches coming out of each branch and two branches going into each node. The trellis structures in this set have a fair amount of symmetry. The simulation results for rate $R = 1$ scalar trellis waveform coder showed that the execution time was reduced to less than one fifth of the original, while there was no performance loss. However, this approach did not work well for vector trellis waveform coding, and a significant loss of performance was seen due to the smaller state-space.

The trellis coded quantization results show that this quantization technique does not have a sufficiently high performance for sources with memory, and the analogies from trellis coded modulation which work well for trellis coded quantizer design for memoryless sources do not carry over to trellis coded quantization for sources with memory.

APPENDIX

In this appendix, we give the optimal decoders obtained for each constraint-length of the quantization systems simulated. The SQNR values of these decoders were given in Chapter 5. Also, the typical SA parameters for obtaining good decoders for each quantization system are given.

Appendix A

Trellis Waveform Coders

A.1 Memoryless Gaussian Source

Typical SA Parameters

Markov chain length : $20 \times N$

Initial temperature :Johnson's method for

number of iterations : 20

$X_0 = 0.8$

decrement coefficient for temperature : 0.85

exit epsilon : 0.001

temperature increment coefficient after GLA : 5.0

temperature increment coefficient after EXTEND : 3.0

Best Decoders

n	branch 0	branch 1
0	0	1
1	0	1

(a) next-state map

n	branch 0	branch 1
0	0.400727	-0.394668
1	-0.394668	0.400727

(b) output map

Table A.1: $K = 2$, TWC, Gaussian i.i.d. source

n	branch 0	branch 1
0	0	2
1	1	3
2	1	3
3	0	2

(a) next-state map

n	branch 0	branch 1
0	0.498106	-0.933209
1	-0.402267	1.083091
2	1.227503	-0.398080
3	-1.353667	0.345442

(b) output map

Table A.2: $K = 3$, TWC, Gaussian i.i.d. source

n	branch 0	branch 1
0	4	0
1	5	6
2	0	4
3	6	5
4	1	7
5	3	2
6	7	1
7	2	3

(a) next-state map

n	branch 0	branch 1
0	1.023595	-0.433611
1	-1.612412	0.335513
2	1.116483	-0.380100
3	-1.130612	0.358584
4	1.153563	-0.410210
5	-1.143184	0.336238
6	1.392741	-0.369801
7	-0.864818	0.504675

(b) output map

Table A.3: $K = 4$, TWC, Gaussian i.i.d. source

n	branch 0	branch 1
0	8	10
1	1	0
2	7	2
3	12	5
4	2	3
5	5	14
6	3	8
7	11	9
8	12	7
9	0	1
10	10	13
11	15	6
12	11	4
13	6	14
14	13	4
15	9	15

(a) next-state map

n	branch 0	branch 1
0	0.956935	-0.626999
1	-1.449356	0.204309
2	0.814988	-0.308483
3	-1.167243	0.433341
4	1.111163	-0.523422
5	-1.096058	0.288956
6	0.943482	-0.522604
7	-0.771833	0.770788
8	0.834855	-0.469643
9	-1.992718	0.243803
10	1.181968	-0.317668
11	-0.970490	0.477060
12	1.338492	-0.386708
13	-0.889492	0.465952
14	1.799364	-0.248257
15	-0.646306	0.547582

(b) output map

Table A.4: $K = 5$, TWC, Gaussian i.i.d. source

n	branch 0	branch 1
0	1	21
1	19	26
2	14	5
3	26	0
4	6	30
5	14	17
6	25	0
7	28	18
8	21	23
9	10	3
10	24	7
11	20	31
12	30	27
13	4	29
14	7	15
15	11	13
16	12	1
17	29	27
18	8	6
19	23	5
20	31	16
21	2	10
22	9	28
23	9	25
24	4	22
25	3	11
26	2	8
27	17	20
28	16	12
29	18	19
30	22	24
31	13	15

(a) next-state map

Table A.5: $K = 6$, TWC, Gaussian i.i.d. source

n	branch 0	branch 1
0	1.005928	-0.570051
1	-0.894044	0.467453
2	0.613815	-0.522761
3	-1.114618	0.621003
4	1.069456	-0.303017
5	-0.648785	0.418263
6	2.132303	-0.179864
7	-1.415704	0.359435
8	0.789710	-0.417666
9	-1.250954	0.331562
10	0.965719	-0.264935
11	-1.904037	0.112167
12	1.522146	-0.424552
13	-0.651077	0.664537
14	1.045453	-0.478977
15	-0.972378	0.489671
16	0.983003	-0.240081
17	-0.650244	0.561341
18	0.724704	-0.396233
19	-0.861156	0.593744
20	1.017855	-0.321219
21	-0.700058	0.744010
22	1.442099	-0.026367
23	-1.419970	0.370383
24	1.071631	-0.259449
25	-1.731396	0.190351
26	1.122810	-0.510312
27	-1.562663	-0.017035
28	1.665159	-0.203703
29	-0.964871	0.832433
30	0.730172	-0.718638
31	-0.888529	0.347167

(b) output map

Table A.5: $K = 6$, TWC, Gaussian i.i.d. source

A.2 First Order Gauss-Markov Source

Typical SA Parameters

Markov chain length : 10
Initial temperature :Johnson's method for
number of iterations : 20
$X_0 = 0.8$
decrement coefficient for temperature : 0.83
exit epsilon : 0.0001
temperature increment coefficient after GLA : 5.0
temperature increment coefficient after EXTEND : 3.0

Best Decoders

n	branch 0	branch 1
0	1	0
1	0	2
2	3	1
3	2	3

(a) next-state map

n	branch 0	branch 1
0	-1.317558	-3.555688
1	-1.332245	1.172386
2	1.021110	-0.217269
3	1.215147	3.409187

(b) output map

Table A.6: $K = 3$, TWC, first order Gauss-Markov source

n	branch 0	branch 1
0	1	4
1	0	3
2	7	6
3	5	1
4	0	4
5	2	6
6	5	3
7	2	7

(a) next-state map

n	branch 0	branch 1
0	-1.842838	-3.240623
1	-1.975841	-0.679682
2	3.087688	1.917083
3	0.215805	-0.841480
4	-3.148938	-4.784435
5	1.869517	0.624560
6	1.424980	0.151375
7	3.055708	4.850288

(b) output map

Table A.7: K = 4, TWC, first order Gauss-Markov source

n	branch 0	branch 1
0	3	8
1	4	7
2	7	3
3	11	2
4	0	12
5	13	14
6	13	9
7	6	9
8	1	4
9	0	11
10	15	5
11	14	1
12	8	12
13	10	6
14	5	2
15	10	15

(a) next-state map

n	branch 0	branch 1
0	-1.595288	-2.337436
1	-2.210103	-1.149231
2	0.061080	-0.579140
3	-1.019557	-0.389978
4	-2.454559	-3.815359
5	2.472368	1.243834
6	1.307222	0.577187
7	0.340460	-0.875333
8	-2.180346	-3.326029
9	-1.469261	0.010402
10	3.740247	2.583740
11	0.513443	-1.161432
12	-3.595695	-5.154974
13	2.746586	1.722071
14	1.391175	0.439584
15	3.669924	5.266708

(b) output map

Table A.8: K = 5, TWC, first order Gauss-Markov source

n	branch 0	branch 1
0	9	27
1	20	23
2	23	1
3	19	18
4	1	12
5	29	21
6	18	22
7	14	7
8	0	4
9	17	2
10	15	29
11	11	19
12	8	20
13	25	30
14	5	6
15	26	31
16	12	3
17	4	9
18	21	7
19	16	3
20	16	28
21	13	2
22	5	25
23	30	0
24	17	24
25	8	11
26	31	22
27	6	13
28	24	28
29	10	14
30	26	27
31	10	15

(a) next-state map

Table A.9: $K = 6$, TWC, first order Gauss-Markov source

n	branch 0	branch 1
0	-0.974455	-0.164520
1	-2.000049	-1.074749
2	-0.822482	-1.552700
3	-0.869755	-0.507039
4	-1.129692	-3.601913
5	1.946057	1.050050
6	0.756962	1.086598
7	1.321117	-0.374331
8	-1.589065	-2.634410
9	-1.057536	-1.395539
10	3.244459	2.753728
11	0.498574	-0.758142
12	-2.647436	-3.348706
13	0.217420	0.929271
14	0.305753	1.399723
15	4.046789	5.346585
16	-1.930892	-1.402306
17	-1.966713	-2.423832
18	0.315046	0.558480
19	-1.129013	0.068317
20	-2.174525	-4.064084
21	0.225421	-0.379364
22	1.763938	0.744750
23	0.120431	-0.492896
24	-2.389042	-3.315650
25	-1.845309	0.159638
26	3.820021	2.367039
27	-0.520255	-0.463210
28	-3.867087	-5.286714
29	2.733174	1.553090
30	2.280112	1.301552
31	3.778540	5.272804

(b) output map

Table A.9: K = 6, TWC, first order Gauss-Markov source

n	branch 0	branch 1
0	34	27
1	49	0
2	13	2
3	37	46
4	32	44
5	61	63
6	18	54
7	48	62
8	25	4
9	57	48
10	53	23
11	11	52
12	41	20
13	43	22
14	13	6
15	15	37
16	28	4
17	36	8
18	2	15
19	11	7
20	17	12
21	38	21
22	61	55
23	16	46
24	16	43
25	40	35
26	29	22
27	55	1
28	60	28
29	42	29
30	26	5
31	42	47

(a) next-state map

Table A.10: $K = 7$, TWC, first order Gauss-Markov source

n	branch 0	branch 1
32	41	39
33	59	51
34	58	32
35	19	34
36	33	12
37	3	17
38	39	31
39	10	53
40	9	20
41	52	24
42	47	58
43	45	25
44	40	23
45	51	30
46	45	38
47	26	31
48	44	19
49	36	1
50	54	50
51	9	63
52	49	24
53	5	62
54	30	57
55	59	27
56	8	0
57	7	18
58	21	14
59	6	33
60	56	60
61	10	14
62	56	3
63	50	35

(a) next-state map

Table A.10: K = 7, TWC, first order Gauss-Markov source

n	branch 0	branch 1
0	-1.118349	-0.652145
1	-1.575397	-1.157551
2	0.381212	-0.917564
3	-0.372262	1.069438
4	-2.081864	-2.368968
5	2.045488	0.518434
6	0.639121	1.468953
7	-0.996002	-0.415278
8	-1.667547	-2.743954
9	-0.665897	-1.634842
10	2.069734	0.980288
11	0.540117	-0.768559
12	-3.082379	-4.027615
13	0.645130	1.507162
14	1.672200	1.538610
15	-0.594325	0.017553
16	-4.580405	-2.220619
17	-2.406700	-2.352795
18	0.361871	0.371523
19	-0.964185	-0.581324
20	-3.152089	-4.112184
21	1.419376	0.170540
22	2.730453	1.453511
23	-1.352295	-0.307346
24	-3.336836	-0.924279
25	-1.404181	-0.689395
26	3.304664	3.017478
27	0.298691	-0.643394
28	-4.743217	-5.863248
29	3.179764	2.414539
30	2.075268	0.819766
31	3.508097	5.098598

(b) output map

Table A.10: K = 7, TWC, first order Gauss-Markov source

n	branch 0	branch 1
32	-1.516683	-0.436919
33	-0.245306	-0.668541
34	0.347109	-0.876801
35	-0.587441	-0.059613
36	-1.179526	-3.105496
37	-0.228464	-1.346078
38	1.276110	2.972176
39	1.279942	0.689286
40	-1.810527	-3.017648
41	-2.730923	-2.188176
42	3.775604	2.540935
43	0.269531	-0.414943
44	-2.560996	-1.025332
45	0.335405	1.036889
46	0.409701	1.763624
47	3.774790	5.190171
48	-1.911748	-1.151869
49	-1.662895	-1.472138
50	1.433393	2.031889
51	-0.247752	0.645106
52	-1.651646	-2.968583
53	1.735594	1.185651
54	1.755470	0.871120
55	0.864863	0.111568
56	-2.775165	-1.053425
57	-0.461909	-0.105301
58	1.241838	1.945119
59	0.522274	-0.631521
60	-3.438734	-4.309810
61	2.667530	1.931792
62	-0.295367	0.708645
63	0.997236	0.645050

(b) output map

Table A.10: K = 7, TWC, first order Gauss-Markov Source

Appendix B

Vector Trellis Waveform Coders

Typical SA Parameters

Markov chain length : 10
Initial temperature :Johnson's method for
number of iterations : 20
$X_0 = 0.8$
decrement coefficient for temperature : 0.80
exit epsilon : 0.0001
temperature increment coefficient after GLA : 5.0
temperature increment coefficient after EXTEND : 3.0

Best Decoders

n	branch 0	branch 1	branch 2	branch 3
0	0	0	1	2
1	3	1	3	3
2	0	3	0	1
3	0	3	2	1

(a) next-state map

n	branch		
0	0	4.741133	4.700180
	1	-4.857343	-4.858268
	2	3.289365	2.596924
	3	-3.388359	-2.716634
1	0	2.478996	2.560070
	1	1.630951	1.721158
	2	0.138875	0.033560
	3	0.889355	-0.222991
2	0	-1.956920	-1.877931
	1	-1.186885	-0.657885
	2	-2.590685	-3.327544
	3	-0.579778	0.261204
3	0	2.801722	3.399782
	1	1.383932	1.249191
	2	-0.606197	-1.231465
	3	0.416988	0.799233

(b) output map

Table B.1: N=4, VTWC, first order Gauss-Markov source

n	branch 0	branch 1	branch 2	branch 3
0	0	0	1	6
1	0	1	7	3
2	2	1	6	5
3	4	1	2	5
4	0	3	5	2
5	3	5	1	7
6	6	7	0	3
7	5	2	6	7

(a) next-state map

n	branch		
0	0	5.270138	5.218014
	1	-5.576530	-5.527416
	2	3.943297	3.183580
	3	-4.038607	-3.366626
1	0	3.383417	4.156414
	1	2.740852	2.686939
	2	1.373504	0.337192
	3	1.974406	1.642668
2	0	-1.428479	-1.499126
	1	0.877330	2.004853
	2	-0.634315	-1.514517
	3	-0.845216	-0.335287
3	0	2.032316	2.725143
	1	1.311425	1.859928
	2	-0.072939	-0.827797
	3	0.803080	0.267072
4	0	3.740659	3.647718
	1	3.100620	2.306767
	2	2.436588	1.530980
	3	1.529577	0.450096
5	0	0.530888	0.972506
	1	1.526149	1.300949
	2	0.164933	0.919070
	3	0.291570	-0.357172
6	0	-2.896511	-2.815017
	1	-2.250079	-1.477163
	2	-3.545522	-4.247868
	3	-1.251571	0.129170
7	0	-0.142969	0.540490
	1	-0.260402	-0.052904
	2	-1.642102	-2.460466
	3	-1.040778	-1.044247

(b) output map

Table B.1: N=8, VTWC, first order Gauss-Markov source

n	branch 0	branch 1	branch 2	branch 3
0	0	0	1	6
1	12	9	15	11
2	2	1	6	5
3	12	9	10	13
4	4	4	3	2
5	11	13	10	15
6	6	7	0	15
7	13	10	14	15
8	10	3	8	10
9	0	1	7	3
10	15	9	14	13
11	4	1	15	5
12	12	11	13	5
13	3	5	1	7
14	14	15	13	11
15	5	2	6	7

(a) next-state map

Table B.2: N=16, VTWC, first order Gauss-Markov source

n	branch		
0	0	5.604480	5.611066
	1	-5.938118	-5.984645
	2	4.492712	3.605468
	3	-4.516740	-3.750058
1	0	3.024328	3.570780
	1	2.746552	2.830484
	2	1.208910	0.531043
	3	1.974774	1.823300
2	0	-0.876491	-0.773329
	1	1.055753	2.265142
	2	-0.828843	-1.921983
	3	0.104732	0.095432
3	0	1.629170	2.521529
	1	1.279947	1.411081
	2	0.256865	-0.464566
	3	0.613959	0.099513
4	0	4.369026	4.495937
	1	3.152935	2.799014
	2	3.052041	1.906039
	3	1.837109	0.728823
5	0	0.705043	1.227415
	1	1.777746	1.143278
	2	0.836047	0.362670
	3	0.103672	-0.328298
6	0	-3.316464	-3.323153
	1	-3.016271	-1.996450
	2	-4.022808	-4.645373
	3	-2.040093	-1.322354
7	0	-0.357065	0.618988
	1	-0.853582	-0.425589
	2	-2.204035	-2.552752
	3	-1.234572	-1.214583

(b) output map

Table B.2: N=16, VTWC, first order Gauss-Markov source

n	branch		
8	0	0.019146	0.019146
	1	0.019146	0.019146
	2	0.019146	0.019146
	3	0.019146	0.019146
9	0	3.398979	4.366274
	1	2.697483	2.764611
	2	1.130342	-0.176796
	3	1.709224	1.536646
10	0	-0.953227	-1.975671
	1	0.637947	1.763029
	2	-0.585642	-1.493568
	3	-0.624613	-0.514568
11	0	2.057704	2.858293
	1	1.891900	1.965196
	2	0.269043	-1.350487
	3	1.135860	0.466150
12	0	4.120543	3.749097
	1	3.328598	2.627894
	2	2.476284	1.190209
	3	2.464518	1.824183
13	0	0.360137	0.776775
	1	1.390415	1.456153
	2	0.108352	1.173122
	3	0.019502	-0.640070
14	0	-1.443214	-1.477200
	1	-2.673143	-2.308079
	2	-1.853312	-0.883801
	3	-1.241634	0.353378
15	0	-0.730426	0.127879
	1	-0.078537	0.043096
	2	-1.987503	-2.829045
	3	-1.397843	-1.232957

(b) output map

Table B.2: N=16, VTWC, first order Gauss-Markov source

Appendix C

Finite-State Vector Quantizers

Typical SA Parameters for $k = 1$

Markov chain length : $50 \times N$

Initial temperature :Johnson's method for

number of iterations : 20

$X_0 = 0.8$

decrement coefficient for temperature : 0.90

exit epsilon : 0.0001

temperature increment coefficient after GLA : 7.0

temperature increment coefficient after EXTEND : 5.0

Best Decoders

n	branch 0	branch 1
0	4	2
1	7	1
2	4	6
3	4	7
4	3	0
5	4	3
6	2	6
7	3	1

(a) next-state map

n	branch 0	branch 1
0	-0.310431	-2.219085
1	1.854699	4.178630
2	-0.981642	-2.910005
3	-0.399322	1.480231
4	0.490338	-1.386205
5	0.038175	0.038175
6	-2.201912	-4.442060
7	0.525304	2.473366

(b) output map

Table C.1: $k = 1$, $N = 8$, FSVQ, first order Gauss-Markov source

Typical SA Parameters for $k = 2$

Markov chain length : $25 \times N$
Initial temperature :Johnson's method for
number of iterations : 20
$X_0 = 0.8$
decrement coefficient for temperature : 0.85
exit epsilon : 0.0001
temperature increment coefficient after GLA : 5.0
temperature increment coefficient after EXTEND : 5.0

n	branch 0	branch 1	branch 2	branch 3
0	0	0	2	2
1	2	1	1	3
2	0	2	2	1
3	1	3	1	7
4	0	6	2	3
5	3	1	0	2
6	3	0	5	5
7	3	3	1	7

(a) next-state map

Table C.2: $k = 2$, $N = 8$, FSVQ, first order Gauss-Markov source

n	branch		
0	0	5.265943	5.168925
	1	3.750847	3.579158
	2	2.684506	2.194793
	3	1.418963	0.600342
1	0	0.769608	1.267821
	1	-1.535065	-1.561955
	2	-0.477617	-0.253232
	3	-2.727873	-3.128867
2	0	2.894946	3.238086
	1	1.646706	1.668941
	2	0.516535	0.343445
	3	-0.652627	-1.192525
3	0	-1.375284	-0.610177
	1	-3.520238	-3.500049
	2	-2.622608	-2.111162
	3	-4.760429	-4.935087
4	0	4.982733	3.395519
	1	1.071062	2.719021
	2	1.303239	2.599440
	3	3.424378	7.463252
5	0	-2.038737	-1.136977
	1	0.459546	-1.348421
	2	1.991501	-1.163665
	3	0.120761	-1.472381
6	0	5.004342	7.290382
	1	-0.032637	1.297522
	2	0.380164	2.049153
	3	-2.219444	1.268733
7	0	-1.962890	-1.477953
	1	-4.720466	-4.143184
	2	-3.565314	-2.581057
	3	-6.209052	-6.024692

(b) output map

Table C.2: $k = 2$, $N = 8$, FSVQ, first order Gauss-Markov source

Typical SA Parameters for $k = 3$

Markov chain length : $10 \times N$
Initial temperature :Johnson's method for
number of iterations : 20
$X_0 = 0.8$
decrement coefficient for temperature : 0.85
exit epsilon : 0.0001
temperature increment coefficient after GLA : 5.0
temperature increment coefficient after EXTEND : 5.0

n	br 0	br 1	br 2	br 3	br 4	br 5	br 6	br 7
0	0	0	3	0	1	1	1	6
1	0	1	3	3	6	0	1	7
2	0	2	2	1	1	5	7	5
3	0	3	1	7	3	6	3	7
4	1	5	6	0	4	7	5	5
5	1	0	5	6	0	3	6	0
6	0	3	3	6	1	7	6	7
7	6	6	6	7	1	1	7	7

(a) next-state map

Table C.3: $k = 3$, $N = 8$, FSVQ, first order Gauss-Markov source

n	branch			
0	0	5.109363	5.431272	5.164586
	1	4.006186	3.966095	3.738970
	2	1.130406	0.475162	0.265883
	3	2.514161	2.846415	3.161463
	4	2.636059	1.596125	0.482898
	5	3.634866	2.720881	1.935693
	6	1.693023	1.668988	1.912491
	7	0.526378	-0.701334	-1.088612
1	0	2.703965	3.483591	3.653046
	1	2.530346	2.378704	1.762619
	2	-0.534182	-0.637212	-0.412121
	3	1.615769	1.135916	0.318295
	4	0.871124	-0.258427	-0.879948
	5	1.253705	1.691569	2.252157
	6	0.312138	0.395390	1.042463
	7	-0.577184	-1.672618	-2.266603
2	0	-7.873535	-7.343118	28.097795
	1	-12.485530	-7.631879	6.858236
	2	2.909985	-21.846602	-0.878008
	3	17.380016	5.414954	-1.483955
	4	9.173210	-3.319110	-7.088783
	5	-3.803651	11.083963	-1.407703
	6	7.891477	4.432295	-0.264960
	7	11.722192	12.466121	-0.571655
3	0	1.918854	2.708772	2.806156
	1	1.054827	0.970893	0.381185
	2	0.614771	1.372642	1.913235
	3	-0.379710	-1.375547	-2.187086
	4	-0.440361	-0.024965	0.646896
	5	0.157639	-0.232453	-0.814859
	6	-1.299189	-1.364472	-0.691068
	7	-1.861581	-2.547947	-2.845411

Table C.3: $k = 3$, $N = 8$, FSVQ, first order Gauss-Markov source

n	branch			
4	0	4.423141	-0.535716	-10.151269
	1	2.569156	5.758779	-28.459314
	2	0.784074	-0.115801	-6.596109
	3	-2.953858	-3.355364	0.302886
	4	-6.919418	8.108907	11.269861
	5	-12.787813	10.286900	-4.243959
	6	-2.459163	1.542001	4.164834
	7	-13.148629	7.328683	0.870785
5	0	7.759699	5.494298	3.877136
	1	-5.943651	-2.899627	6.206387
	2	-10.869227	14.524001	-2.006455
	3	-1.314725	0.206836	20.874896
	4	-9.774140	4.949367	-6.447612
	5	5.948494	-15.910822	8.628721
	6	12.463370	-25.775613	2.236852
	7	-14.569044	-3.913555	7.314335
6	0	0.853615	1.632691	1.960263
	1	-1.454003	-0.925700	-0.064284
	2	-0.020440	-0.023677	-0.495230
	3	-2.272457	-2.050835	-1.447365
	4	-0.385407	0.364243	0.914068
	5	-1.639155	-2.316810	-2.953503
	6	-0.857495	-1.113238	-1.559765
	7	-2.932331	-3.645203	-3.713961
7	0	-1.069803	-0.928287	-1.215668
	1	-3.499389	-2.818865	-1.776407
	2	-2.296375	-1.847692	-1.712942
	3	-2.520621	-2.806058	-3.420323
	4	-0.593835	0.313559	0.890702
	5	-2.223030	-0.993426	0.015999
	6	-4.155631	-3.840168	-3.562237
	7	-5.118240	-5.376744	-5.165783

(b) output map

Table C.3: $k = 3$, $N = 8$, FSVQ, first order Gauss-Markov source

Typical SA Parameters for $k = 4$

Markov chain length : $25 \times N$
Initial temperature :Johnson's method for
number of iterations : 20
$X_0 = 0.8$
0 decrement coefficient for temperature : 0.85
exit epsilon : 0.0001
temperature increment coefficient after GLA : 5.0
temperature increment coefficient after EXTEND : 5.0

n	b0	b1	b2	b3	b4	b5	b6	b7	b8	b9	b10	b11	b12	b13	b14	b15
0	0	0	4	2	4	0	0	3	2	5	1	5	0	2	2	7
1	4	5	0	5	0	6	1	3	1	6	0	7	2	3	2	7
2	0	1	0	5	4	3	2	2	4	2	1	3	3	1	3	3
3	0	4	1	5	5	2	7	3	3	2	7	3	6	3	7	7
4	1	5	6	0	4	7	5	5	4	4	0	1	0	4	4	6
5	1	1	0	0	4	1	2	3	4	2	1	5	0	5	5	3
6	6	3	0	3	5	3	6	3	4	5	4	6	0	3	0	6
7	4	5	2	3	0	6	7	3	3	2	7	7	0	7	3	7

(a) next-state map

Table C.4: $k = 4$, $N = 8$, FSVQ, first order Gauss-Markov source

n	branch				
0	0	5.227879	5.222703	5.180445	4.962482
	1	3.061840	3.529504	3.024169	2.644407
	2	4.908993	4.109995	3.460792	2.490690
	3	2.315391	1.986947	1.432104	0.813344
	4	3.210274	3.822244	4.296479	4.324479
	5	1.348381	1.430499	1.716049	1.979447
	6	2.976751	2.070146	1.984638	2.548806
	7	1.310724	0.500959	-0.688282	-1.526424
	8	3.400473	2.922449	1.815959	1.119154
	9	2.273855	2.174724	0.413194	-0.624835
	10	2.086431	0.647237	0.443988	0.580190
	11	0.429293	-0.070658	0.601813	1.125022
	12	1.901406	2.147314	3.231333	3.423555
	13	0.836969	0.946234	0.549445	0.079659
	14	-0.362503	-0.517407	-0.394530	-0.353687
	15	0.042171	-1.200428	-1.428519	-1.629724
2	0	1.109072	1.868644	2.886931	3.242663
	1	0.804820	0.975939	1.178270	0.894042
	2	1.419470	0.888401	1.108795	2.206163
	3	0.554578	-0.248155	0.332569	0.579718
	4	2.546227	3.282443	3.383271	3.270037
	5	1.088489	1.269104	-0.077087	-0.561724
	6	-0.340642	-0.962293	-1.045435	0.031829
	7	-1.172360	-2.039238	-2.147176	-1.562863
	8	1.867937	2.163727	1.973565	1.090582
	9	-0.017820	0.104221	0.000968	-1.040702
	10	-0.306480	0.380124	1.125690	1.866454
	11	-0.321653	-0.626501	-1.691397	-2.386711
	12	-1.074334	-0.303466	0.104046	0.412994
	13	0.589971	-0.442100	-1.182364	-1.347421
	14	-1.366746	-1.438010	-0.677140	-0.753578
	15	-1.748543	-2.503087	-3.005085	-3.532139

(b) output map

Table C.4: $k = 4$, $N = 8$, FSVQ, first order Gauss-Markov source

n	branch				
2	0	1.653353	3.323528	3.995395	3.655310
	1	0.157950	-0.188828	-0.619951	-0.021862
	2	0.219413	0.890520	1.634637	2.170611
	3	1.485394	1.330426	0.344219	0.052146
	4	1.289104	1.932611	2.261041	2.354643
	5	-0.390299	-0.896879	-1.409742	-1.373270
	6	-0.555592	-0.422140	0.435892	1.126510
	7	-1.383174	-1.196924	-0.371784	-0.412408
	8	1.977054	2.699583	2.202050	1.574635
	9	1.379172	0.264099	0.357382	1.485374
	10	0.184771	0.879950	1.238693	0.867818
	11	-1.699794	-2.190999	-1.872276	-1.331113
	12	0.675461	0.179512	-0.893515	-1.690347
	13	0.225928	0.351949	0.552408	-0.507554
	14	-0.720230	-1.578199	-2.385555	-2.918827
	15	-2.108860	-3.361140	-4.005695	-3.788009
3	0	0.039789	0.761580	2.588775	2.778637
	1	-0.545327	0.178386	0.965966	1.527636
	2	-0.201165	-0.098703	-0.533126	-0.654125
	3	-1.675274	-1.364454	-0.828726	-0.376175
	4	0.385342	1.299466	1.271423	0.740254
	5	-1.093258	-0.097195	0.210242	0.064081
	6	-0.976682	-1.437113	-2.548065	-3.363974
	7	-3.485885	-2.994728	-2.593587	-2.067435
	8	-1.010938	-1.074361	-1.264308	-1.714685
	9	-2.681102	-2.508308	-1.403470	-0.434489
	10	-2.594999	-1.918280	-1.706093	-2.441286
	11	-3.861101	-4.397837	-3.940016	-3.255873
	12	-1.748360	-1.039979	0.055208	1.043799
	13	-1.638354	-2.539036	-2.687967	-1.738728
	14	-2.579102	-2.998587	-3.579586	-3.866772
	15	-4.023214	-4.533536	-5.178437	-5.526924

(b) output map

Table C.4: $k = 4$, $N = 8$, FSVQ, first order Gauss-Markov source

n	branch				
4	0	5.281671	5.681374	5.775602	5.433521
	1	3.010793	3.246764	3.089777	3.146135
	2	4.854619	4.515278	3.981652	3.107831
	3	2.831109	2.289188	1.364458	0.336485
	4	3.351190	3.883814	4.627867	4.761534
	5	1.448312	1.649043	1.685533	1.283646
	6	2.978435	2.005127	1.873545	2.119432
	7	0.946447	0.147288	-1.023231	-1.058876
	8	3.656195	3.141818	2.477588	1.521517
	9	2.279281	1.379370	0.295886	-0.871526
	10	2.136742	1.106592	0.182649	0.586265
	11	0.735370	0.209262	0.996495	1.872894
	12	1.733753	1.939007	2.519213	3.099075
	13	0.420364	0.646570	0.684226	0.126489
	14	-0.106914	-0.768176	-0.488993	-0.200048
	15	-0.388612	-1.712114	-2.038898	-2.567068
5	0	0.865816	2.138511	3.028739	3.037902
	1	0.808163	1.336477	1.471085	0.716005
	2	0.286856	0.310225	0.405434	1.337242
	3	0.687882	-0.160595	0.019041	-0.177723
	4	2.359523	3.063505	3.441359	3.545694
	5	1.428320	1.196454	-0.046013	-0.203154
	6	-0.259998	-0.519088	-1.001004	-0.145809
	7	-1.177289	-1.695135	-1.928733	-1.418465
	8	1.940139	2.007018	1.713247	1.897120
	9	-0.445266	0.441076	0.217998	-0.745537
	10	-0.011374	0.490391	1.740862	2.180284
	11	-0.210068	-1.082106	-2.257986	-2.961311
	12	-0.965287	-0.715514	0.381692	0.760455
	13	0.356798	-0.262727	-1.121685	-1.695942
	14	-1.399474	-1.372330	-0.609834	-0.400839
	15	-1.887589	-2.828381	-3.287188	-3.245541

(b) output map

Table C.4: $k = 4$, $N = 8$, FSVQ, first order Gauss-Markov source

n	branch				
6	0	3.038388	3.807328	3.646188	3.460132
	1	-0.066479	-0.168720	-0.413650	0.649414
	2	-0.380786	0.751973	1.963218	2.908115
	3	1.752275	1.485710	0.646039	-0.244543
	4	1.365431	1.940973	2.796231	2.879169
	5	-0.702142	-0.770116	-1.329488	-1.988381
	6	-1.452346	-0.752503	0.021547	1.249908
	7	-1.784706	-1.451505	-0.865718	-0.592809
	8	1.835899	2.077102	1.990262	1.676482
	9	1.309921	0.905866	0.706612	1.306185
	10	-0.192321	0.694702	1.089092	0.708565
	11	-1.916234	-2.682508	-2.290774	-1.457715
	12	0.311850	-0.492539	-1.051893	-1.093283
	13	-0.311282	-0.217276	0.194151	-1.046666
	14	-1.059258	-1.679216	-2.215159	-3.082141
	15	-2.458228	-3.746438	-4.221353	-5.329292
7	0	-0.028298	1.388451	1.847877	2.204583
	1	-1.342149	-0.123337	0.685931	1.346408
	2	-0.354274	-0.676146	-0.590660	-0.310146
	3	-1.792123	-1.787193	-1.479759	-0.703968
	4	-0.232535	0.999194	1.193569	0.563240
	5	-1.737747	-0.392301	0.378597	-0.456511
	6	-2.304754	-2.063374	-2.568177	-3.569643
	7	-4.056797	-3.735753	-2.805000	-1.888820
	8	-1.232179	-0.904354	-1.142411	-1.882263
	9	-3.322765	-2.447452	-1.068927	-0.320465
	10	-3.315508	-1.858131	-1.689576	-1.870707
	11	-4.709952	-4.622735	-4.202162	-3.710412
	12	-2.244998	-1.336790	-0.361302	0.505608
	13	-1.792309	-2.384165	-2.712839	-2.028203
	14	-3.345218	-3.380766	-3.420138	-3.602545
	15	-5.405321	-5.777733	-6.136681	-5.951677

(b) output map

Table C.4: $k = 4$, $N = 8$, FSVQ, first order Gauss-Markov source

Appendix D

Predictive Trellis Waveform Coders

D.1 First Order Gauss-Markov Source

Typical SA Parameters

Markov chain length : $20 \times N$
Initial temperature :Johnson's method for
number of iterations : 20
$X_0 = 0.8$
decrement coefficient for temperature : 0.70
exit epsilon : 0.001
temperature increment coefficient after GLA : 5.0
temperature increment coefficient after EXTEND : 3.0

Best Decoders
Predictor coefficient : 0.906695

n	branch 0	branch 1
0	0	1
1	0	1

(a) next-state map

n	branch 0	branch 1
0	1.111982	-0.384706
1	0.413708	-1.123029

(b) output map

Table D.1: $K = 2$, PTWC, first order Gauss-Markov Source

Predictor coefficient : 0.916341

n	branch 0	branch 1
0	2	1
1	2	1
2	0	3
3	0	3

(a) next-state map

n	branch 0	branch 1
0	0.915178	-0.269363
1	0.312816	-0.944278
2	0.911924	-0.301855
3	0.281040	-0.980543

(b) output map

Table D.2: $K = 3$, PTWC, first order Gauss-Markov source

Predictor coefficient : 0.927614

n	branch 0	branch 1
0	2	6
1	3	5
2	0	1
3	2	3
4	4	1
5	4	7
6	4	7
7	6	1

(a) next-state map

n	branch 0	branch 1
0	1.075853	-0.130653
1	0.251032	-0.954735
2	0.935542	-0.205267
3	0.189845	-1.117598
4	1.031421	-0.161472
5	0.300580	-1.071149
6	0.923144	-0.381030
7	0.291341	-1.082695

(b) output map

Table D.3: $K = 4$, PTWC, first order Gauss-Markov source

Predictor coefficient : 0.927650

n	branch 0	branch 1
0	2	15
1	11	13
2	0	8
3	10	11
4	4	1
5	12	15
6	4	5
7	7	4
8	0	14
9	10	11
10	8	1
11	2	3
12	12	9
13	12	5
14	13	15
15	6	1

(a) next-state map

n	branch 0	branch 1
0	1.134871	-0.137760
1	0.181215	-0.986359
2	0.987349	-0.213432
3	0.174623	-1.138433
4	1.060623	-0.073275
5	0.356516	-1.036582
6	0.982620	-0.414797
7	0.058227	0.058227
8	1.117464	-0.093702
9	0.201276	-0.934484
10	0.846399	-0.280867
11	0.160655	-1.089513
12	1.010280	-0.137835
13	0.292415	-1.086113
14	0.807014	-0.436842
15	0.323473	-1.070942

(b) output map

Table D.4: $K = 5$, PTWC, first order Gauss-Markov source

Predictor coefficient : 0.929305

n	branch 0	branch 1
0	2	15
1	27	29
2	0	8
3	26	27
4	4	1
5	28	31
6	4	5
7	23	20
8	0	14
9	26	27
10	8	1
11	18	19
12	12	9
13	28	21
14	13	15
15	22	17
16	18	31
17	11	13
18	16	24
19	10	11
20	20	17
21	12	15
22	20	21
23	7	4
24	16	30
25	10	11
26	24	17
27	2	3
28	28	25
29	12	5
30	29	31
31	6	1

(a) next-state map

Table D.5: $K = 6$, PTWC, first order Gauss-Markov source

n	branch 0	branch 1
0	1.068216	-0.141438
1	0.227482	-1.016312
2	1.001803	-0.180752
3	0.129795	-1.155584
4	1.061888	-0.097036
5	0.334116	-1.024486
6	1.014661	-0.371507
7	0.058227	0.058227
8	1.043461	-0.196263
9	0.178796	-0.875891
10	0.835491	-0.301874
11	0.116516	-1.075254
12	0.932344	-0.087264
13	0.338087	-1.068908
14	0.822509	-0.543447
15	0.364198	-1.023614
16	1.191688	-0.109190
17	0.163243	-0.975402
18	1.001946	-0.276178
19	0.160787	-1.052673
20	0.966187	0.018772
21	0.414337	-1.053512
22	1.015183	-0.456718
23	0.058227	0.058227
24	1.104032	-0.127150
25	0.226031	-0.933410
26	0.800843	-0.253945
27	0.157525	-1.032063
28	1.029958	-0.143985
29	0.328459	-1.113034
30	0.766298	-0.519169
31	0.309725	-1.034781

(b) output map

Table D.5: $K = 6$, PTWC, first order Gauss-Markov source

Predictor coefficient : 0.981607

n	branch 0	branch 1
0	4	3
1	63	48
2	10	12
3	60	59
4	0	6
5	58	51
6	6	7
7	61	57
8	23	19
9	62	63
10	2	9
11	46	51
12	8	14
13	48	55
14	14	15
15	53	61
16	20	19
17	37	34
18	22	29
19	44	43
20	16	23
21	37	35
22	21	9
23	45	41
24	24	27
25	34	47
26	18	25
27	36	35
28	24	30
29	32	39
30	30	31
31	37	45

(a) next-state map

Table D.6: $K = 7$, PTWC, first order Gauss-Markov source

n	branch 0	branch 1
32	36	39
33	31	32
34	42	44
35	28	27
36	32	38
37	26	29
38	38	39
39	25	25
40	55	51
41	26	31
42	34	41
43	14	19
44	40	46
45	16	23
46	46	47
47	21	29
48	52	51
49	5	2
50	54	61
51	12	11
52	48	55
53	5	3
54	53	41
55	9	9
56	56	59
57	2	15
58	50	57
59	4	3
60	56	62
61	0	7
62	62	63
63	5	13

(a) next-state map

Table D.6: $K = 7$, PTWC, first order Gauss-Markov source

n	branch 0	branch 1
0	1.067527	-0.227400
1	0.058227	0.058227
2	1.045111	-0.044962
3	0.239541	-1.215145
4	1.123499	-0.254090
5	0.482145	-0.798489
6	0.868862	-0.393713
7	0.158135	-1.010741
8	1.063410	-0.328876
9	0.507587	-1.089764
10	1.239051	-0.062521
11	0.168174	-1.034438
12	0.860863	-0.228660
13	0.394628	-0.969272
14	0.889886	-0.255689
15	0.179361	-0.952348
16	1.154937	-0.345874
17	0.058227	0.058227
18	1.358414	-0.109824
19	0.202926	-1.122350
20	0.858289	-0.292539
21	0.473892	-0.735464
22	1.104599	-0.642922
23	0.396518	-0.983133
24	0.947004	-0.370033
25	0.433335	-0.812679
26	1.139252	0.008647
27	0.212970	-1.003097
28	0.836071	-0.384255
29	0.378602	-0.894847
30	0.969908	-0.274176
31	0.131837	-0.857242

(b) output map

Table D.6: K=7, PTWC, first order Gauss-Markov source

n	branch 0	branch 1
32	1.050203	-0.242625
33	0.058227	0.058227
34	1.034266	-0.198651
35	0.228873	-1.215367
36	1.083966	-0.251974
37	0.445761	-0.823869
38	0.764979	-0.358087
39	0.152590	-0.926611
40	0.996126	-0.181070
41	0.434391	-1.087660
42	1.084295	-0.139640
43	0.058105	-1.211544
44	0.928252	-0.196057
45	0.361007	-0.893359
46	0.881515	-0.259517
47	0.208972	-0.942371
48	1.031775	-0.330460
49	0.058227	0.058227
50	1.371263	0.082492
51	0.217393	-0.989503
52	0.915519	-0.289374
53	0.560198	-0.811990
54	1.059585	-0.272314
55	0.337927	-1.154558
56	1.051480	-0.306598
57	0.470205	-0.920356
58	1.061990	0.032295
59	0.180428	-1.059504
60	0.843564	-0.370121
61	0.386182	-0.961029
62	0.873250	-0.267560
63	-0.077626	-0.926265

(b) output map

Table D.6: K=7, PTWC, first order Gauss-Markov source

D.2 Speech Model Source

Typical SA Parameters

Markov chain length : $20 \times N$
Initial temperature :Johnson's method for
number of iterations : 20
$X_0 = 0.80$
decrement coefficient for temperature : 0.65
exit epsilon : 0.001
temperature increment coefficient after GLA : 5.0
temperature increment coefficient after EXTEND : 3.0

Best Decoders

a_0	a_1	a_2
1.720968	-1.157482	0.260467

(a) predictor coefficients

n	branch 0	branch 1
0	0	1
1	0	1

(b) next-state map

n	branch 0	branch 1
0	0.511626	-0.054820
1	0.073658	-0.493039

(c) output map

Table D.7: $K = 2$, PTWC, speech model source

a_0	a_1	a_2
1.471274	-0.757848	0.063235

(a) predictor coefficients

n	branch 0	branch 1
0	3	1
1	0	3
2	2	1
3	2	1

(b) next-state map

n	branch 0	branch 1
0	0.346273	-0.207067
1	0.185423	-0.443003
2	0.527068	-0.094146
3	0.199848	-0.417819

(c) output map

Table D.8: K=3, PTWC, speech model source

a_0	a_1	a_2
1.422285	-0.625918	-0.043350

(a) predictor coefficients

n	branch 0	branch 1
0	6	5
1	0	7
2	7	4
3	4	7
4	0	1
5	0	3
6	2	4
7	6	1

(b) next-state map

n	branch 0	branch 1
0	0.271190	-0.245824
1	0.173037	-0.415635
2	0.468551	-0.104078
3	0.199708	-0.391784
4	0.308147	-0.218104
5	0.170122	-0.422712
6	0.495243	-0.092068
7	0.222125	-0.406521

(c) output map

Table D.9: K=4, PTWC, speech model source

a_0	a_1	a_2
1.446420	-0.620597	-0.070605

(a) predictor coefficien'

n	branch 0	branch 1
0	6	5
1	8	15
2	10	13
3	9	11
4	12	5
5	14	1
6	8	11
7	14	9
8	12	13
9	0	7
10	14	12
11	4	5
12	2	9
13	8	3
14	10	3
15	6	5

(b) next-state map

n	branch 0	branch 1
0	0.281038	-0.201709
1	0.091144	-0.477598
2	0.424088	-0.024323
3	0.123631	-0.333086
4	0.237027	-0.210885
5	0.183590	-0.367477
6	0.456388	-0.054239
7	0.134848	-0.411201
8	0.210858	-0.243100
9	0.120873	-0.395746
10	0.483714	-0.033224
11	0.156109	-0.349602
12	0.305478	-0.126260
13	0.099484	-0.323703
14	0.387079	-0.038689
15	0.204252	-0.505380

(c) output map

Table D.10: K=5, PTWC, speech model source

Bibliography

[1] C. E. Shannon, "A mathematical theory of communication," *Bell Syst. Tech. J.,* vol. 27, pp. 379–423, 623–656, 1948.

[2] C. E. Shannon, "Coding theorems for a discrete source with a fidelity criterion," *IRE National Convention Record,* Part 4, pp. 142–163, 1959.

[3] Y. Linde, A. Buzo, and R. M. Gray, "An algorithm for vector quantizer design," *IEEE Trans. Commun.,* vol. COM-28 pp. 84–95, Jan. 1980.

[4] A. Gersho, R. M. Gray, *Vector Quantization and Signal Compression,* Kluwer Academic Publishers, 1991.

[5] R. M. Gray, *Source Coding Theory,* Kluwer Academic Press, Boston, 1990.

[6] J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Language, and Computation,* Addison-Wesley, Reading, Mass., 1979.

[7] J. Foster, R. M. Gray, M. O. Dunham, "Finite-state vector quantization for waveform coding," *IEEE Trans. Info. Theory,* vol. IT-31, pp. 348–359, May 1985.

[8] A. Haoui and D. G. Messerschmitt, "Predictive vector quantization," in *Proc. ICASSP-84,* San Diego, CA, Mar. 19–21, 1984.

[9] M. O. Dunham and R. M. Gray, "An algorithm for vector quantizer design," *IEEE Trans. Commun.,* vol. COM-28, pp. 84–95, Jan. 1980.

[10] A. Gersho and B. Ramamurthi, "Image coding using vector quantization," *Proc. ICASSP-82,* vol 1, pp. 428–431, Paris, April 1982.

[11] F. Jelinek and J. B. Anderson, "Tree encoding of memoryless discrete time sources with a fidelity criterion," *IEEE Trans. Inform. Theory,* vol. IT-15, pp. 584–590, Sept. 1969.

[12] R. M. Gray, "Time-invariant trellis encoding of ergodic discrete-time sources with a fidelity criterion," *IEEE Trans. Inform. Theory,* vol. IT-23, pp. 71–83, Jan. 1977.

[13] G. D. Forney, "The Viterbi algorithm," *Proc. IEEE,* vol. 61, pp. 268–278, March 1973.

[14] A. J. Viterbi, "Error bounds for convolutional codes and an asmptotically optimum decoding algorithm," *IEEE Trans. Inform. Theory,* vol. IT-13, pp. 260–269, Apr. 1967.

[15] J. K. Omura, "On the Viterbi decoding algorithm," *IEEE Trans. Inform. Theory,* vol. IT-15, pp. 177–179, Jan. 1969.

[16] A. J. Viterbi, J. K. Omura, *Principles of Digital Communication and Coding,* McGraw-Hill, 1979.

[17] L. C. Stewart, " Trellis data compression," Stanford Electron. Lab., Stanford, CA, Tech. Rep. L905-1, July 1981.

[18] C. C. Cutler, "Delayed encoding: stabilizer for adaptive coders," *IEEE Trans. Commun. Technol.,* vol. COM-19, pp. 898–907, Dec. 1971.

[19] E. Ayanoğlu, R. M. Gray, "The design of predictive trellis waveform coders using the generalized Lloyd algorithm," *IEEE Trans. Commun.,* vol. COM-34, pp. 1073–1081, Nov. 1986.

[20] L. C. Stewart, R. M. Gray, Y. Linde, "The design of trellis waveform coders," *IEEE Trans. Commun.,* vol. COM-30, pp. 702–711, April 1982.

[21] M. R. Garey and D. D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness,* W.H. Freeman and Co., San Francisco, 1979.

[22] A. H. G. Rinnooy Kan and G. T. Timmer, "Stochastic methods for global optimization," Econometric Institute, Erasmus University, Rotterdam, *Report* 8317/0, 1983.

[23] S. Kirkpatrick, C. D. Gelatt, M. P. Vecchi, "Optimization by simulated annealing," *Science,* vol. 220, pp. 671–680, May 1983.

[24] P. J. M. van Laarhoven, E. H. L. Aarts, *Simulated Annealing: Theory and Applications,* MIA D. Reidel Publishing Company, 1988.

[25] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, E. Teller, "Equation of state calculations by fast computing machines," *J. of Chem. Physic* vol. 21, pp. 1087–1092, 1953.

[26] R. H. J. M. Otten, L. P. P. P. van Ginneken, *The Annealing Algorithm,* Kluwer Academic Publishers, 1989.

[27] W. Feller, *An Introduction to Probability Theory and Its Applications,* vol.1, Wiley, New York, 1950.

[28] D. S. Johnson, C. R. Aragon, L. A. McGeoch, C. Shevon, "Optimization by simulated annealing: an experimental evaluation, Parts I and II," *Oper. Res.,* vol. 37, pp. 865–892, December 1989, and vol. 39, pp. 378–406, June 1991.

[29] R. E. Burkard and F. Rendl. "A thermodynamically motivated simulation procrdure for combinatorial optimization problems," *European J. of Oper. Res.,* vol. 17, pp. 169–174, 1984.

[30] M. Lundy and A. Mees, "Convergence of an annealing algorithm," *Math. Prog.,* vol. 34, pp. 111–124, 1986.

[31] G. H. Sasaki and B. Hajek, "The time complexity of maximum matching by simulated annealing," *Journal of the ACM,* vol. 35, pp. 387–403, 1988.

[32] S. Geman and D. Geman, "Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images," *IEEE Proc. Pattern Analysis and Machine Intelligence,* vol. PAMI-6, pp. 721–741, 1986.

[33] A. El Gamal, L. A. Hemachandra, I. Shperling and V. K. Wei, "Using simulated annealing to design good codes," *IEEE Trans. Inform. Theory,* vol. IT-33, pp. 116–123, 1987.

[34] J. K. Flanagan, D. R. Morrell, R. L. Frost, C. J. Read and B. E. Nelson, "Vector quantization codebook generation using simulated annealing," *Proc. ICASSP-89,* pp. 1759–1762, 1989.

[35] A. E. Çetin and V. Weerackody, "Design of vector quantizers using simulated annealing," *IEEE Trans. Circ. Syst.,* vol. CAS-35, p. 1550, Dec. 1988.

[36] N. E. Collins, R. W. Eglese, B. L. Golden, "Simulated annealing—an annotated bibliography," *Amer. J. of Math. and Man. Sci.,* vol. 8, pp. 205–307, 1988.

[37] B. H. Juang, "Design and performance of trellis vector quantizers for speech signals," *IEEE Trans. Acoust., Speech, Signal Processing,* vol. ASSP-36, pp. 1423–1431, Sept. 1988.

[38] C.-D. Bei, R. M. Gray, "Simulation of vector trellis encoding systems," *IEEE Trans. Commun.,* vol. COM-34, pp. 214–218, March 1986.

[39] D. E. Knuth, *The Art of Computer Programming,* Addison-Wesley, Reading, Mass. 1981.

[40] S. P. Lloyd, "Least squares quantization in PCM," *IEEE Trans. Inform. Theory,* vol. IT-28, pp. 129–137, Mar. 1982, reprint of unpublished 1957 rep.

[41] J. Max, "Quantizing for minimum distortion," *IRE Trans. Inform. Theory,* vol. IT-6, pp. 7–12, Mar. 1960.

[42] Y. Linde and R. M. Gray, "A fake process approach to data compression," *IEEE Trans. Commun.,* vol. COM-26, pp. 840–847, June 1978.

[43] W. A. Pearlman, "Sliding-block and random source coding with constrained size reproduction alphabets," *IEEE Trans. Commun.,* vol. COM-30, pp. 1859–1867, Aug. 1982.

[44] G. H. Freeman, J. W. Mark, I. F. Blake, "Trellis source codes designed by conjugate gradient optimization," *IEEE Trans. Commun.,* vol. COM-36, pp. 1–12, January 1988.

[45] M. J. D. Powell, "An efficient method for finding the minimum of a function of several variables without calculating derivatives," *Comp. J.,* vol.7, pp. 155–162, May 1964.

[46] M. J. D. Powell, "Restart procedures for the conjugate gradient method," *Math. Prog.,* vol. 12, pp. 241–254, 1977.

[47] P. E. Gill, W. Murray, and M. H. Wright, *Practical Optimization,* Academic Press, London, 1981.

[48] R. M. Gray, "Time-invariant trellis encoding of ergodic discrete-time sources with a fidelity criterion," *IEEE Trans. Inform. Theory,* vol. IT-23, pp. 71–83, Jan. 1977.

[49] S. G. Wilson, S. Husain, "Adaptive tree encoding of speech at 8000 bits/s with a frequency-weighted error criterion," *IEEE Trans. Commun.,* vol. COM-27, pp. 165–170, Jan. 1979.

[50] R. A. McDonald, "Signal-to-quantization noise ratio and idle channel performance of DPCM systems with particular application to voice signals," *Bell Syst. Tech. J.,* vol. 45, pp. 1123–1151, Sept. 1966.

[51] M. W. Marcellin, T. R. Fischer, "Trellis coded quantization of memoryless and Gauss-Markov sources," *IEEE Trans. Commun.*, vol. COM-38, pp.82–93, Jan. 1990.

[52] G. Ungerboeck, "Trellis coded modulation with redundant signal sets," *IEEE Commun. Mag.*, vol. 25, pp. 5–21, Feb. 1987.

[53] W. A. Finamore and W. A. Pearlman, "Optimal encoding of discrete-time continuous-amplitude memoryless sources with finite output alphabets," *IEEE Trans. Inform. Theory,* vol. IT-26, pp. 144–155, Mar. 1980.