This is to certify that the

thesis entitled

EFFICIENT DISCRETE SYMBOL HIDDEN MARKOV MODEL EVALUATION USING TRANSFORMATION AND STATE REDUCTION

presented by

Ross Kenneth Snider

has been accepted towards fulfillment of the requirements for

Master's degree in Electrical Engineering

John R. Haller Jr
Major professor

Date 9 Nov. 90

EFFICIENT DISCRETE SYMBOL HIDDEN MARKOV MODEL EVALUATION USING TRANSFORMATION AND STATE REDUCTION

By

Ross Kenneth Snider

A THESIS

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

MASTER OF SCIENCE

Department of Electrical Engineering

1990

ABSTRACT

EFFICIENT DISCRETE SYMBOL HIDDEN MARKOV MODEL EVALUATION USING TRANSFORMATION AND STATE REDUCTION

By

Ross Kenneth Snider

A procedure is developed for evaluating the likelihood of a hidden Markov model (HMM) using only $\mathcal{O}([1-\kappa]NT)$ floating point operations, where N is the number of states in the model, T is the number of observations derived from an utterance, and κ is the degree of computational complexity reduction which can be close to unity. Effective recognition of speech using conventional algorithms require $\mathcal{O}(3NT)$ (Bakis model) to $\mathcal{O}(N^2T)$ (ergodic model) operations to evaluate an HMM. Reduction in computational complexity is required for near-real-time recognition algorithms to be feasible on ordinary personal computers. The motivation for this reduction is the development of a PC-based speech recognition system for disabled individuals. This thesis will explore the tradeoff between computational complexity and recognition rate for various speakers, vocabularies, and compression techniques.

ACKNOWLEDGEMENTS

The idea for the state space formulation and transformation contained in this thesis was suggested by my advisor Dr. John Deller Jr., who also made numerous helpful suggestions in the research and also in the write up.

This work was supported by the Whitaker Foundation.

Contents

List of Tables

List of Figures

1 Introduction and Background

The general goal of this research is to assist in the development of a site within the Speech Processing Laboratory that will accept speech data from speech disabled clients at clinical centers, and return to these persons a customized speech recognition software package, implementable on a relatively ordinary personal computer (PC). The precise technical form of this recognizer will depend upon a particular individual's needs and purposes and will not be further addressed here.

The specific goal of the work presented in this thesis is to provide a recognition technique that will be used in the recognition software that is faster than the existing algorithms used in evaluating discrete symbol hidden Markov models (HMM). This research is restricted to hidden Markov modeling of isolated words, i.e. speech with deliberate pauses between words. The reason for this is that cerebral palsy speech is not well understood and using it in the continous speech case would involve more unknowns than for the isolated word case. Thus this research addresses the simpler case, though nothing restricts the techniques reported in this thesis to the isolated word case. Though the end goal of this research is to assist speech disabled individuals, to compare the technique in this thesis with existing methods of evaluating HMM's, normal speech is used.

Encouraging isolated word recognition results for speech of cerebral palsied individuals with a spectrum of articulatory abilities were recently reported [1, 2, 3]. The reason the PC is being used is to maximize accessibility of these developments to users

with disabilities. Customized architectures or mainframe computing environments are not generally available to the user of a prospective speech recognition device. Since the constraint of the PC is being used, measures must be taken to speed up existing recognition algorithms, particularly those involved in feature extraction and HMM decoding. It is the HMM search with which this thesis is primarily concerned.

The Baum-Welsh "forward-backward" recursions (see, e.g., [4]) in evaluating an ergodic HMM (a fully connected model) require $\mathcal{O}(N^2T)$ operations where N is the number of states in the model and T is the number of observations extracted from the given utterance. While this represents a remarkable improvement over the $\mathcal{O}(N^T)$ operations required for a "brute force" procedure, it still does not allow for near-real-time evaluation of a realistic number of HMM's once we have the observations representing a given utterance. For example, a floating point operation (flop)[1] requires about 40 μs on our laboratory's 16 MHz, 80386-based PC equipped with a floating point coprocessor. The evaluation of an $N = 6$ state HMM with respect to an utterance represented by $T = 100$ observations requires about $N^2T = 3600$ flops, or 0.144 s. If 100 HMM's are to be evaluated in some time slot, then the time requirement for recognition of this single word is 14.4 s which is unacceptable. Even if a $\mathcal{O}(3NT)$ method is used, the recognition would take 7.2 s. This thesis will demonstrate a method where $\mathcal{O}(N^2T)$ operations needed to evaluate an HMM can be reduced to $\mathcal{O}([1-\kappa]NT)$ where κ is the ***compression index*** and is chosen on the basis of the desired tradeoff between computational complexity and recognition rate. Thus the value of κ is chosen on an experimental basis. If, for example, κ were chosen to be 0.9, then in the above example $(\frac{1}{10}NT)$ flops would be necessary for each HMM and the time to recognize a word would be .24 s compared with 14.4 s for (N^2T), or 7.2 s for

[1] A flop is taken to be one multiplication plus one addition.

$\mathcal{O}(3NT)$, flops per model. This is "real-time," given the above vocabulary size, and further reductions are possible with faster clock speeds or other existing technologies which are quickly becoming "ordinary," or with further theoretical development.

2 Computational Complexity Reduction

2.1 Complexity Reduction by Transformation

In the following discourse it is assumed that the reader is familiar with hidden Markov modeling and how the likelihood evaluation of an HMM is computed. The reader is referred to [4, 5, 16] for more details. It should be pointed out that this thesis is dealing with *discrete symbol HMM's* as opposed to other variants such as continuous mixture density HMM's (see, e.g., [15]).

While it is not conventional to do so, the HMM is viewed here in a state-space-like formulation. Suppose a model, **M**, has N states q_i, $i = 1, 2, \ldots, N$ and M discrete observation symbols z_k, $k = 1, 2, \ldots, M$. At discrete observation time t, let us define the *state probability vector*, $\mathbf{x}(t)$, and the *observation probability vector*, $\mathbf{y}(t)$, as follows:

$$\mathbf{x}^T(t) \doteq [x_1(t) \;\; x_2(t) \;\; \ldots \;\; x_N(t)] \tag{1}$$

$$\mathbf{y}^T(t) \doteq [y_1(t) \;\; y_2(t) \;\; \ldots \;\; y_M(t)] \tag{2}$$

where, $x_i(t)$ = (the probability of being in q_i at discrete time t given the model **M**) = $P(q_i \text{ at } t \mid \mathbf{M})$, and $y_k(t)$ = (the probability of generating symbol z_k at discrete time t given **M**) = $P(z_k \text{ at } t \mid \mathbf{M})$. It is not difficult to see that the following *state equation* and *observation equation* describe the dynamics of the model:

$$\mathbf{x}(t+1) = \mathbf{A}\mathbf{x}(t) + \mathbf{u}(t)\,\delta(t) \tag{3}$$

$$\mathbf{y}(t) = \mathbf{B}\mathbf{x}(t) \tag{4}$$

where, **A** is the $N \times N$ *state transition matrix* associated with the HMM whose

(i,j) element, $a_{ij} = P(q_i \text{ at } t+1 \mid q_j \text{ at } t)$ for any t; $\mathbf{B}$ is the $M \times N$ *observation probability matrix* whose (k,j) element, $b_{kj} = P(z_k \mid q_j)$; and $\mathbf{u}(0)$ is some vector such that when $\mathbf{x}(0)$ is defined to be zero, $\mathbf{x}(1)$ takes the proper initial values, with $\mathbf{u}(t)$ arbitrary but finite $\forall t \neq 0$, and $\delta(t)$ the unit sample sequence. Note carefully that if the observation symbol at time t is z_k, then only element $y_k(t)$ of $\mathbf{y}(t)$ need be calculated.

To derive a more efficient computational structure, we first apply a diagonalizing equivalence transformation (e.g., see [6, pp.146-154]) to obtain

$$\bar{\mathbf{x}}(t+1) = \bar{\mathbf{A}}\bar{\mathbf{x}}(t) + \bar{\mathbf{u}}(t)\delta(t) \tag{5}$$

$$\mathbf{y}(t) = \bar{\mathbf{B}}\bar{\mathbf{x}}(t) \tag{6}$$

where $\bar{\mathbf{A}} = \mathbf{UPAP}^{-1}\mathbf{U}^{-1}$ with $\mathbf{P}$ the usual matrix of *normalized* eigenvectors, $\mathbf{U}$ a special diagonal matrix described below, $\bar{\mathbf{x}}(t) = \mathbf{UPx}(t)$, $\bar{\mathbf{u}}(t) = \mathbf{UPu}(t)$, and $\bar{\mathbf{B}} = \mathbf{BP}^{-1}\mathbf{U}^{-1}$. The i^{th} diagonal element of the matrix $\mathbf{U}$ is the reciprocal of the i^{th} element of the vector $\mathbf{Pu}(0)$. As a consequence of this operation each element of the excitation vector, $\bar{\mathbf{u}}(t)$, is unity at time zero. This result will be important below.

The diagonalizing transformation has reduced the computation to $\mathcal{O}(NT)$ operations per HMM since $\bar{\mathbf{A}}$ is a diagonal matrix, and since only N flops are needed to compute a relevant $y_k(t)$. Note that the systems (3) - (4) and (5) - (6) are equivalent and therefore this reduction has no impact on the recognition rate. It should be pointed out, however, that in computing the state space structure, a summing procedure is used rather than a Viterbi search (see, e.g., [16]). In the method employed here, the likelihood that the HMM produced the observation sequence (using *any* sequence of states) is computed whereas in the Viterbi decoding approach, the likelihood associated with the *best* state sequence is assigned to the model. The summing

procedure is used because it admits complexity reduction that is not possible with Viterbi decoding. A graphical representation of the diagonalizing procedure can be seen in Figures 1 and 2. Figure 1 shows how the state probability vector is computed based on equation (3) *before* the transformation is applied. The computation of the state probability vector *after* the transformation is shown in Fig. 2, which is based on equation (5). The α_i's in Fig. 2 are the eigenvalues of the matrix **A** and are thought of as the poles of the state space system describing the HMM.

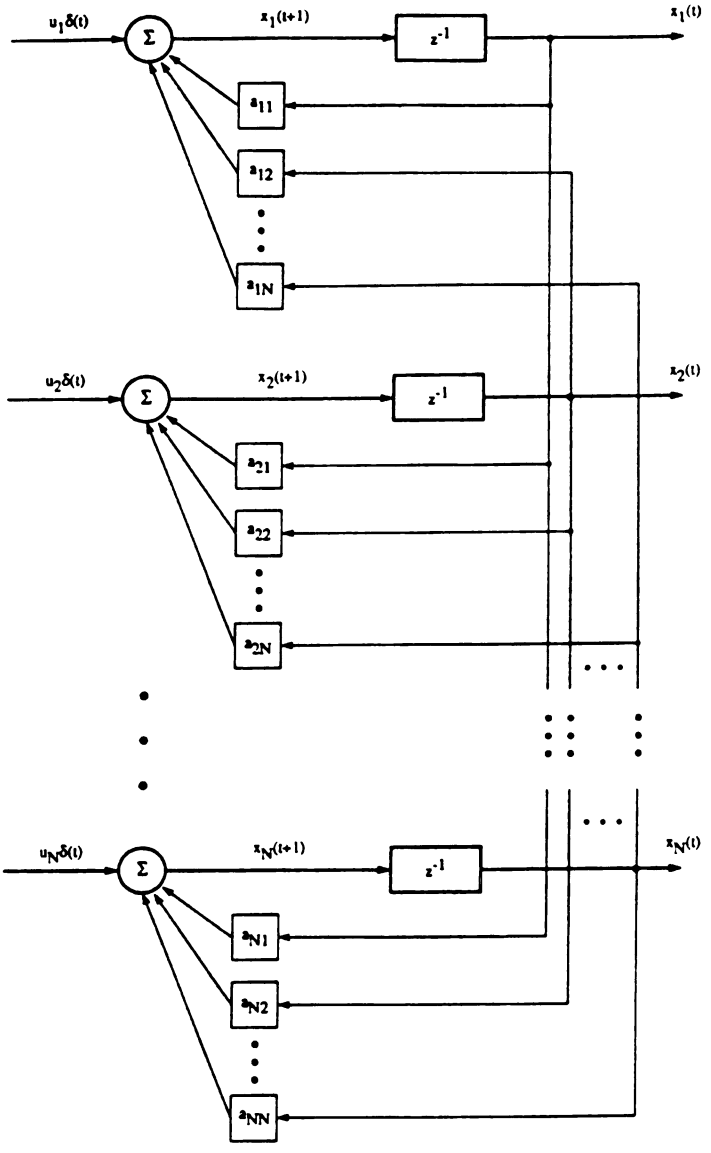

Figure 1: Computation of state probability vector *before* transformation.

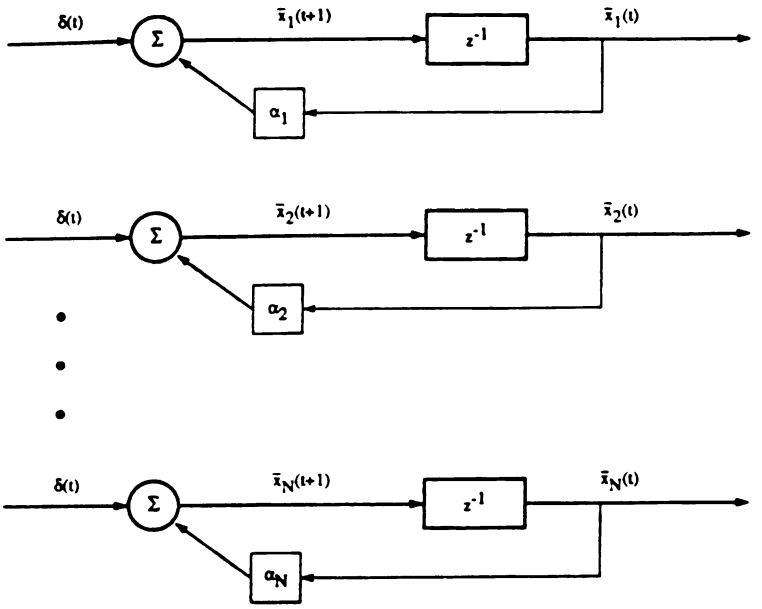

Figure 2: Computation of state probability vector *after* transformation.
The α_i's are the eigenvalues of the matrix **A**.

2.2 Complexity Reduction by Compression

To demonstrate further computational benefit, it is convenient to look at all the HMM's representing the words in the vocabulary at once. To do this the HMM's are combined into one large "state space" formulation. This is done as follows. Suppose there are W total models or words in the vocabulary. Let us construct "global" state and observation equations

$$\tilde{\mathbf{x}}(t+1) = \tilde{\mathbf{A}}\tilde{\mathbf{x}}(t) + \tilde{\mathbf{u}}(t)\delta(t) \tag{7}$$

$$\tilde{\mathbf{y}}(t) = \tilde{\mathbf{B}}\tilde{\mathbf{x}}(t) \tag{8}$$

in which $\tilde{\mathbf{x}}(t)$ is a vector of dimension NW consisting of the concatenation of the W individual state vectors, $\tilde{\mathbf{y}}(t)$ is a similarly formed MW-vector, $\tilde{\mathbf{A}}$ is a block diagonal

matrix of dimension $NW \times NW$ with the individual state transition matrices, $\bar{\mathbf{A}}$, running down its diagonal, and $\tilde{\mathbf{B}}$ is a similarly formed $NW \times MW$ matrix on the output side. $\tilde{\mathbf{u}}(t)$ is the obvious NW-vector of ones at $t = 0$ and is arbitrary otherwise.

The diagonal elements of $\tilde{\mathbf{A}}$, say $\tilde{a}_{ii}$, are the eigenvalues of the original $\mathbf{A}$ matrices or the poles of the input-output description of the HMM. Accordingly, these numbers are bounded in magnitude as $0 \leq \mid \tilde{a}_{ii} \mid \leq 1$. Empirically, we find that these numbers are often real (or have very small imaginary parts), and, if they represent "non-negligible" poles (those with significant magnitude) of the system, they are often clustered near $z = 1$ in the z-plane. This led us to consider whether, with some degree of quantization of the $\tilde{a}_{ii}$ coefficients, some of the state variables could be combined, eliminating the computational burden of nearly redundant states. If, for example, $\tilde{a}_{ii} \approx \tilde{a}_{jj}$ then, *since both state variables represent similar transient responses to a unit sample*[2], it is possible to eliminate $\tilde{x}_j(t)$, and reduce the size of the state space. The appropriate adjustment to the observation equation is to add column j of $\tilde{\mathbf{B}}$ to column i, then eliminate column j. Thus *a computational savings is realized in both the state and observation equations.* Note carefully that the merging of states may take place even if the states are from *different* HMM's! Several strategies for compression will be described in Section 3.1.

We now define the *compression index* κ. This is given as

$$\kappa = \frac{\lambda_{Eliminated}}{\lambda_{Total}} \tag{9}$$

where $\lambda_{Eliminated}$ is the number of eigenvalues merged together and λ_{Total} is the total number of eigenvalues in all the HMM's, i.e. λ_{Total} is equal to NW since one eigenvalue results from each state in each model. Notice that κ can never equal unity since if

[2]Here we see the significance of the matrix **U** above.

all the eigenvalues were merged there would still be one "eigenvalue" left, thus κ is bounded as

$$0 \leq \kappa \leq \frac{\lambda_{Total} - 1}{\lambda_{Total}} \tag{10}$$

where $\kappa = 0$ when no eigenvalues are eliminated.

Further computational efficiency is achieved by quantizing the "$\tilde{b}_{kj}$" coefficients, though the complexity reduction is much less significant than that in the quantization of the $\tilde{a}_{ii}$ coefficients. This is because the "$\tilde{b}_{kj}$" coefficients are not used at every observation whereas all of the $\tilde{a}_{ii}$ coefficients are. The "$\tilde{b}_{kj}$" coefficients are used only when the observation symbol is the k^{th} one. For formal reasons it is convenient to decompose the observation equation into individual equations for each symbol, z_k. For $k = 1, 2, \ldots, M$ we have,

$$\tilde{\mathbf{y}}_k(t) = \tilde{\mathbf{B}}_k \mathbf{D}_k \tilde{\mathbf{x}}(t) \tag{11}$$

where $\tilde{\mathbf{B}}_k$ represents the W rows of $\tilde{\mathbf{B}}$ (with the column modifications indicated above) used in computing likelihoods for z_k. These likelihoods are contained in the W-vector, $\tilde{\mathbf{y}}_k(t)$. $\mathbf{D}_k$ is a square matrix whose function will become clear below. Immediately following the state space reduction procedure, each $\mathbf{D}_k$ is simply an identity matrix with the dimension of the reduced state space. Further reduction is possible if the entries in any two columns of a particular row of $\tilde{\mathbf{B}}_k$, say in columns l and m, are the same. It is more efficient to add the corresponding state variables first, then multiply the sum by the coefficient value once. This is accomplished by changing the element (l, m) of $\mathbf{D}_k$ from zero to one, and the redundant "$\tilde{b}$" coefficient in column m of $\tilde{\mathbf{B}}_k$ to zero. The inclusion of the matrix $\mathbf{D}_k$ sets up an appropriately modified state vector.

3 Experimental Results

3.1 Speakers and Vocabularies

The speakers for the following recognition studies were two normal adult males, hereafter refered to Speaker 1 and Speaker 2. Speaker 1,2 will refer to the combined models of Speakers 1 and 2. By combining the models for Speaker 1,2 we are dealing with a speaker independent case whereas for Speaker 1 and Speaker 2 we are dealing with speaker dependent cases.

The vocabularies were chosen for the following reasons. Vocabularies 2 and 4 are the Texas Instruments "TI-20" and "TI-46" Vocabularies, respectively. These are standard vocabularies used in isolated word recognition tasks and were chosen so that results could be compared with other research using these data bases. Vocabulary 3 was chosen to see what confusion exists among the similar sounding letters of Vocabulary 4 (e.g. b,c,d,e,g,p,t,v,z). Vocabulary 5 consists of words taken from the "Grandfather passage" used by linguists and speech pathologists (see, e.g., [12]) because of the number of phonetically balanced words contained therein. Vocabulary 1 is the list of all the words contained in Vocabularies 2-5, with the exception of the word *you* which is indistinguishable from the letter *u*. Each of the compression experiments described below was performed for each of the three "speakers" on each vocabulary.

Vocabularies				
1				2
a	five	n	swiftly	one
about	four	nearly	t	two
all	frock	nine	thinks	three
an	g	ninety three	three	four
as	go	no	two	five
b	grandfather	o	u	six
beard	h	old	usually	seven
black	he	one	v	eight
buttons	help	p	w	nine
c	himself	q	well	zero
clings	i	r	wish	start
coat	in	repeat	x	stop
d	is	rubout	y	yes
dresses	j	s	years	no
e	k	seven	yes	go
eight	l	several	yet	help
enter	long	six	z	erase
erase	m	start	zero	rubout
ever	missing	still		repeat
f	my	stop		enter

Table 1: Vocabularies 1 and 2.

Vocabularies				
3	**4**		**5**	
a	one	g	about	thinks
b	two	h	all	usually
c	three	i	an	well
d	four	j	as	wish
e	five	k	beard	years
f	six	l	black	yet
g	seven	m	buttons	you
h	eight	n	clings	
i	nine	o	coat	
j	zero	p	dresses	
k	start	q	ever	
l	stop	r	frock	
m	yes	s	grandfather	
n	no	t	he	
o	go	u	himself	
p	help	v	in	
q	erase	w	is	
r	rubout	x	long	
s	repeat	y	missing	
t	enter	z	my	
u	a		nearly	
v	b		ninety three	
w	c		old	
x	d		several	
y	e		still	
z	f		swiftly	

Table 2: Vocabularies 3-5.

3.2 Data Handling and Model Training

To train the HMM's the follow procedure was used. First 15 utterances of each word were recorded on analog tape using a Sony F-510 microphone and a Teac W-450R tape deck with a Dolby C noise reduction system turned on. The order of the utterances was randomized for Vocabulary 5 whereas for Vocabulary 4 the utterances were given in the order as listed in Table 2. A 40 megabyte hard disk file was created and all the utterances were digitized to the file using the Metrabyte *Streamer* software and *DAS16F* sampling board after passing through an anti-aliasing filter and a NAD 3130 stereo amplifier. The utterances were sampled at 10 Khz after being bandlimited to 4.5 Khz. The transfer function of the bandlimiting filter used is found in Fig. 3. Each utterance was then viewed graphically to mark the beginning and ending point and then written to a separate file. This extraction was accomplished by the program *Wavemark* which is listed as Program 1 in Appendix A and which was developed as part of this effort. A Hamming window (see, e.g., [10]) 25.6 ms wide was used every 12.8 ms on the speech data. From each window, six mel-frequency cepstrum coefficients were computed since this parametric representation has been shown to give superior recognition performance over other types of parametric representations (see, e.g., [11]). These coefficients were computed using the program *Cepstrum* which is Program 2 in Appendix B. A codebook of 128 symbols was then produced using only the first 10 utterances of each word. A separate codebook for each speaker and also a codebook combining both speakers were produced. Program 3 in Appendix C entitled *Codebook* was used for this purpose. The codebooks were implemented in a binary tree fashion (see, e.g., [18]). The cepstrum coefficients were then quantized into symbol strings using the codebook. The program *Quantize*, which was used to do the quantization, appears as Program 4 in Appendix D.

Hidden Markov models were then trained for each word in the vocabulary using the first 10 utterances (i.e. symbol strings). For each word three models were trained, one for Speaker 1, another for Speaker 2 and the final for Speaker 1,2. The HMM's were trained using the conventional Baum-Welsh algorithm with appropriate scaling (see, e.g., [8]). The model used for training was the Bakis model [16] with six states, as opposed to the ergodic model which is fully interconnected. Ergodic models have been used in the dysarthic speech case (see, e.g., [2]). Figure 4 shows the Bakis model while Fig. 5 shows the Ergodic model. The program *Hmmtrain* was used in training the models. This appears as Program 5 in Appendix E. The software package *Matlab* (Math Works, Inc.) was then used in conjuction with the ".m files" *trnsfhmm.m* and *autotrns.m* to transform the HMM's as in Section 2.1. These ".m files" are listed under Program 6 in Appendix F. Utterances 11-15 were used to test the models. Program *Evalhmms*, listed as Program 7 in Appendix G, was used in the evaluation. The results of the evaluations are found in Section 3.3. The program to implement the compressions described in Sections 3.5 through 3.7 is entitled *Compress* and is listed as Program 8 in Appendix H.

3.3 Recognition Rates before Compression

Tables 3 through 5 list the recognition rates for the various vocabularies after the transformation and before compression. These are the baseline recognition rates since no loss in the recognition rate has occured due to compression. A comparison to a study on the recognition of digits from the literature is now given. Rabiner in [16] gives a recognition rate of 96.3% for the recognition of digits in a speaker independent case (100 speakers). Vocabulary 2 which has the digits plus 10 other words is the closest to this case. The speaker dependent cases of Vocabulary 2 have higher recognition rates

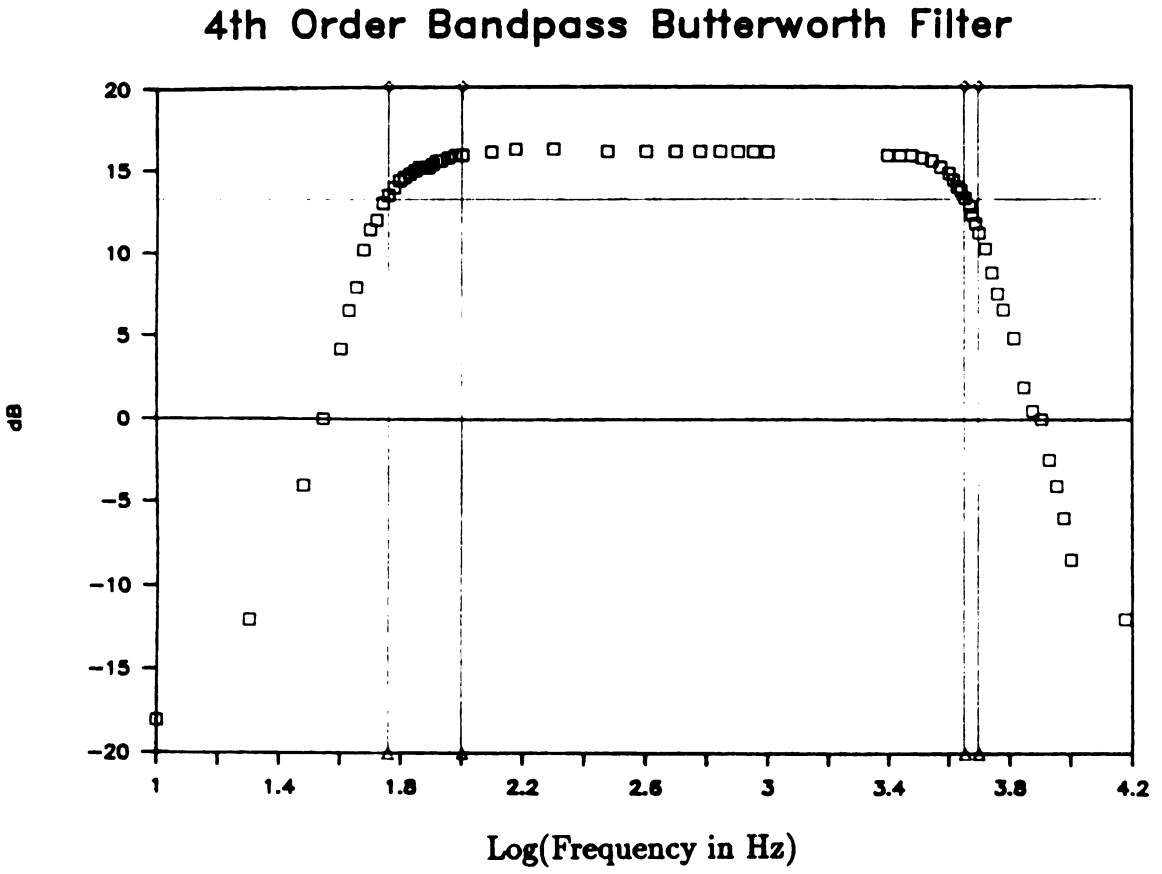

Figure 3: Transfer function of anti-aliasing filter.

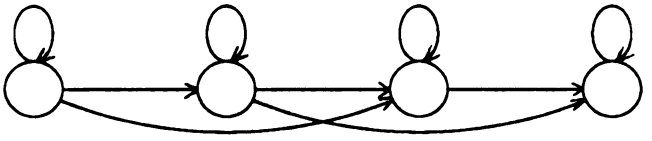

Figure 4: Four state Bakis model.

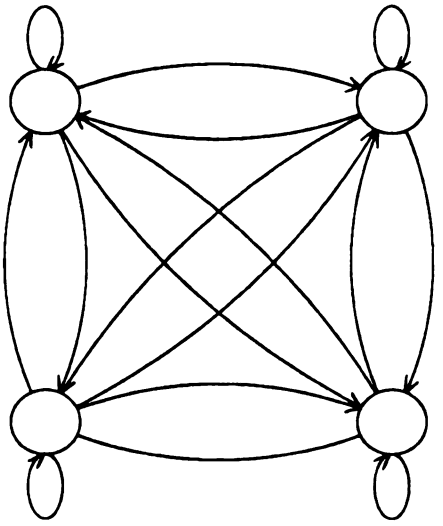

Figure 5: Four state ergodic model.

Vocabulary 1			Vocabulary 2		
Speaker 1	*Speaker 2*	*Speaker 1,2*	*Speaker 1*	*Speaker 2*	*Speaker 1,2*
91.79%	90.00%	81.67%	98.00%	97.00%	88.50%

Table 3: Recognition Rates for Vocabularies 1 and 2.

Vocabulary 3			Vocabulary 4		
Speaker 1	*Speaker 2*	*Speaker 1,2*	*Speaker 1*	*Speaker 2*	*Speaker 1,2*
90.77%	88.46%	82.31%	93.48%	89.13%	81.52%

Table 4: Recognition Rates for Vocabularies 3 and 4.

than 96.3% because in general a speaker dependent case results in better recognition than a speaker independent case. If we look at Speaker 1,2 which is the speaker independent case, the recognition rate is 88.5%, which is for the digits plus 10 other words (which results in more confusions). When the vocabulary is limited to the digits for Speaker 1,2, the recognition rate increases to 93.0%. A likely reason why this is lower then 96.3% is because there were fewer training samples for Speaker 1,2 than in [16]. The recognition rate for Speaker 1 (a speaker dependent case) on the digits was 100%. This recognition rate is expected since it is a speaker dependent case as opposed to the speaker independent case.

Vocabulary 5		
Speaker 1	*Speaker 2*	*Speaker 1,2*
96.97%	96.36%	92.42%

Table 5: Recognition Rates for Vocabulary 5.

3.4 Eigenvalues

In Fig. 6 we see the eigenvalues of the transformed HMM's for Vocabulary 1. Notice that a sixth of the eigenvalues are unity. This is a result of using a six state Bakis model which has an ending state (the final state has probability one of jumping to itself). In fact, any model with an ending state will have an eigenvalue of unity. The proof of this is as follows. A Bakis model corresponds to a matrix representation of the form

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & 0 & 0 & 0 & 0 & \dots & 0 \\ 0 & a_{11} & a_{12} & a_{13} & 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & a_{11} & a_{12} & a_{13} & 0 & 0 & \dots & 0 \\ \vdots & & & & \ddots & & & & \vdots \\ 0 & \dots & 0 & 0 & 0 & 0 & a_{n-2,n-2} & a_{n-2,n-1} & a_{n-2,n} \\ 0 & \dots & 0 & 0 & 0 & 0 & 0 & a_{n-1,n-1} & a_{n-1,n} \\ 0 & \dots & 0 & 0 & 0 & 0 & 0 & 0 & a_{nn} \end{bmatrix} \tag{12}$$

Since $\mathbf{A}$ represents the state transition probabilities, the sum of each row must equal one. Therefore $a_{nn} = 1$. The eigenvalues are determined by

$$\det(\lambda \mathbf{I} - \mathbf{A}) = 0 \tag{13}$$

Now $\det(\lambda \mathbf{I} - \mathbf{A})$ can be expressed as (see, e.g., [14])

$$\det(\lambda \mathbf{I} - \mathbf{A}) = (\lambda - a_{nn}) \det(\lambda \mathbf{I}_{nn} - \mathbf{A}_{nn}) \tag{14}$$

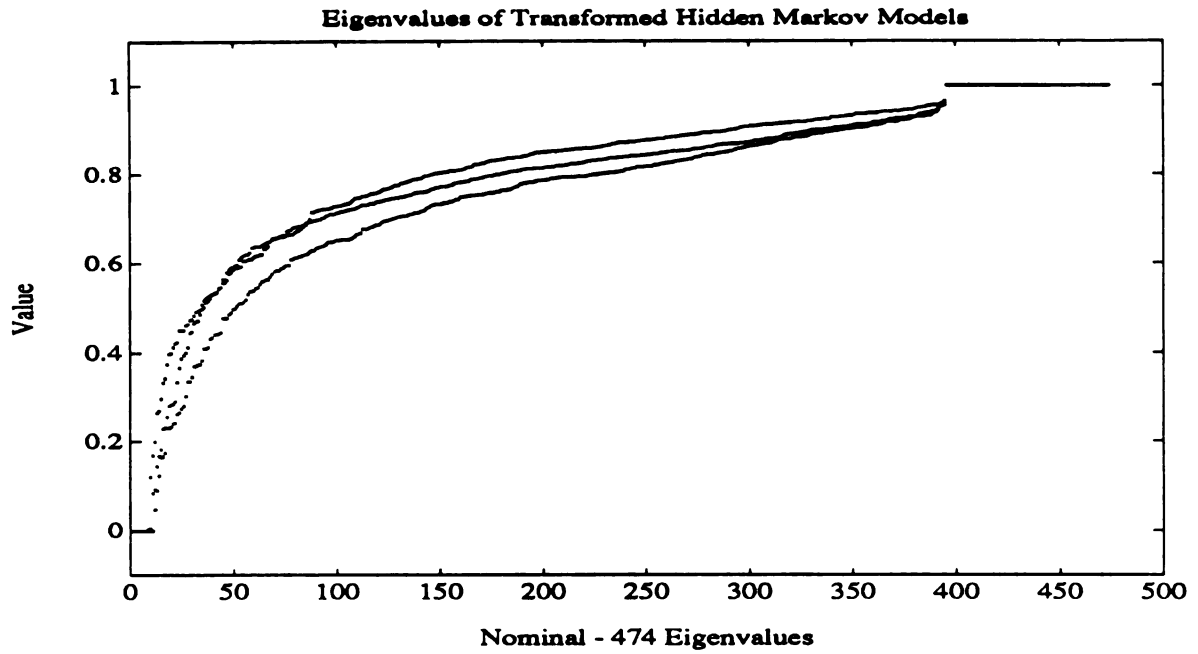

Figure 6: Eigenvalues of all transformed HMM's.
Top curve - Speaker 1,2; Middle curve - Speaker 1; Bottom curve - Speaker 2.

or (since $a_{nn} = 1$),

$$\det(\lambda \mathbf{I} - \mathbf{A}) = (\lambda - 1) \det(\lambda \mathbf{I}_{nn} - \mathbf{A}_{nn}) \tag{15}$$

where $(\lambda \mathbf{I}_{nn} - \mathbf{A}_{nn})$ is obtained by striking out row n and column n in $(\lambda \mathbf{I} - \mathbf{A})$. The eigenvalues of $\mathbf{A}$ are the roots of $\det(\lambda \mathbf{I} - \mathbf{A}) = 0$ which, from (15) clearly include $\lambda = 1$. Therefore each model has at least one unity eigenvalue for this model structure.

3.5 Nearest Neighbor Compression

As noted in Section 2.2 the eigenvalues that are close together represent similar transient responses to a unit sample input. Thus to gain further computation savings these eigenvalues need to be grouped together in some fashion. In this section we will explore what happens if we combine them together in a nearest neighbor procedure (see, e.g., [13]). By nearest neighbor we mean eigenvalues which are closest in the euclidean sense. The nearest neighbor procedure works as follows. Say for example we have a vocabulary of 100 words and are using a six state Bakis model. We then have 600 eigenvalues to compress together. So the nearest neighbor procedure would first find the two eigenvalues that are closest together out of all 600 and replace them by their mean. A recognition experiment would then take place to see what happened to the recognition rate as a result of combining these two eigenvalues. The procedure then continues by finding the next two closest eigenvalues. Notice after this point that a certain "eigenvalue" could be the mean of a previous combining. If so, then the new mean would be that of the original eigenvalues represented by this "eigenvalue" and the eigenvalue combining with it. This continues until all the eigenvalues are grouped into one value, thus the final value equals the mean of all 600 eigenvalues. The last recognition experiment is then run to see the effect of replacing all the eigenvalues by this mean.

From Section 3.4 we notice in the above example that all 100 models have an eigenvalue that equals one. The significance of this is that 100 eigenvalues have nearest neighbors of distance zero and will be combined with no effective change in value which results in no change in the recognition rate. Thus a compression of $\kappa = 0.166$ occurs with no loss in the recognition rate. Figures 7 through 11 show how the recognition rate is affected by compression using the nearest neighbor method for

the Vocabularies 1-5. The solid line represents the recognition rates for Speaker 1, the dashed line for Speaker 2, and the dotted line for Speaker 1,2.

Looking at Fig. 7 which represents Vocabulary 1, there are 468 eigenvalues (78 words with six states each) to compress. A compression of zero means nothing has been done to the eigenvalues while a compression of one means that all the eigenvalues have been replaced by the mean of the 468 eigenvalues. In terms of κ, a compression of zero implies $\kappa = 0$ and if the 468 eigenvalues are replaced by the population mean[3], $\kappa = 0.998$. The reader is refered to page 9 for the definition of κ. The most noticable feature in Fig. 7 is that the compression has a much more detrimental affect on Speaker 1,2 then on Speakers 1 or 2. All the recognition rates in Fig. 7 stay constant initially because all the identical eigenvalues are being combined as discussed earlier. The recognition rate for Speaker 2 drops about 7% after a compression of $\kappa = 0.2$ and then remains somewhat constant until a compression of $\kappa = 0.8$. Speaker 1 on the other hand just gradually degrades to the approximate 7% drop for a compression of $\kappa = 0.8$. What this means in terms of computation is that if a 7% degradation in the recognition rate is acceptable ($\kappa = 0.8$) for Speaker 2 with this 78 word vocabulary, the computation needed to recognize an utterance 1.28 seconds long (100 observation symbols) would be reduced from 140,400 flops (5.6 s on the 16 MHz PC for Bakis model before transformation) to 9360 flops (0.37 s on the PC). If a 25% drop in the recognition rate was acceptable, the compression would then be $\kappa = 0.98$. This would further reduce the computation from the 140,400 flops to 936 flops.

In Fig. 8 for Vocabulary 2 we have 120 eigenvalues to compress. The results are similar to Fig. 7 except for Speaker 1 where a compression of $\kappa = 0.9$ results in a degradation of 10% in the recognition rate. Vocabulary 3 with 156 eigenvalues shown

[3]The compression index κ approaches 1 as all the eigenvalues are replaced by the population mean and as the number of eigenvalues approaches infinity.

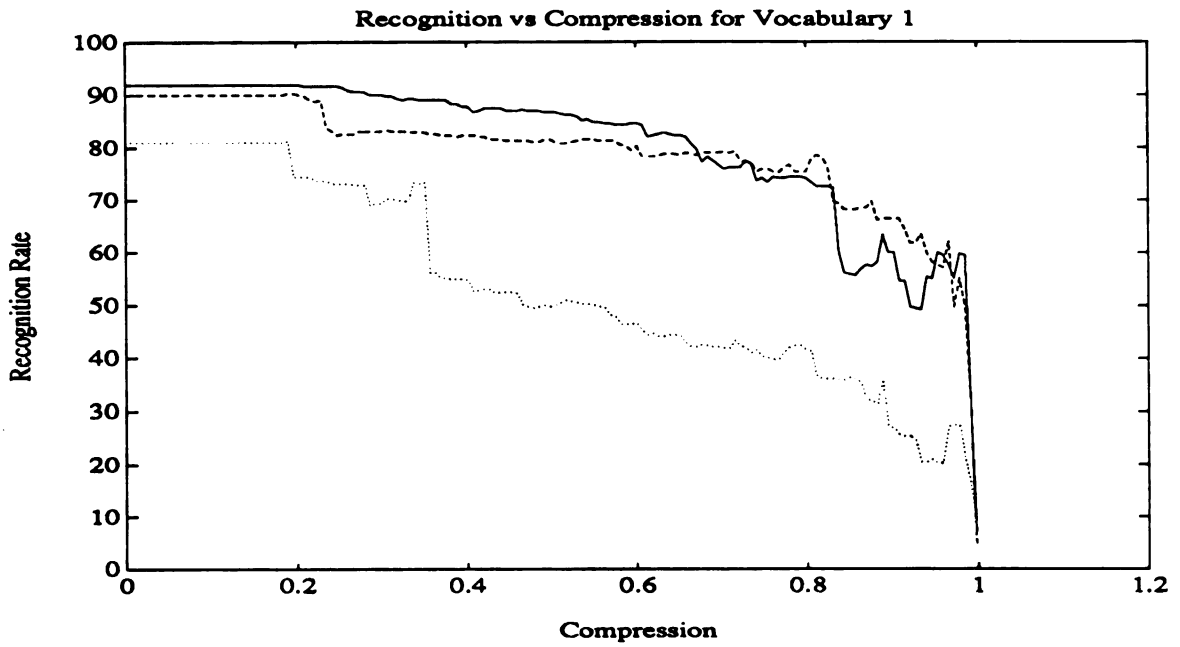

Figure 7: Nearest neighbor compression for Vocabulary 1.
Solid line - Speaker 1; Dashed line - Speaker 2; Dotted line - Speaker 1,2.

in Fig. 9 shows that the compression has a much more pronounced affect than for the previous vocabularies, especially for Speaker 2 and Speaker 1,2. The reason for this is that there are many letters (e.g. b,c,d,g,p,t,v,z) which are confused easily. Thus distinct eigenvalues are apparently important for similar words.

The results of compression for Vocabulary 4 (276 eignevalues) is seen in Fig. 10. Here we see that Speaker 1 loses only 5% in the recognition rate for a compression of $\kappa = 0.65$, but then falls off. Speaker 2 as in Fig. 9 drops off 10% but then remains somewhat constant out to a compression of $\kappa = 0.8$ where it drops off. Speaker 1,2 does not perform well with the compression greater than $\kappa = 0.3$. Vocabulary 5 (198 eigenvalues) in Fig. 11 shows that all speakers have a similar performance at a compression of $\kappa = 0.7$. This is due to the vocabulary of words that are much more dissimilar than the words in the other vocabularies. The dissimilar words are not

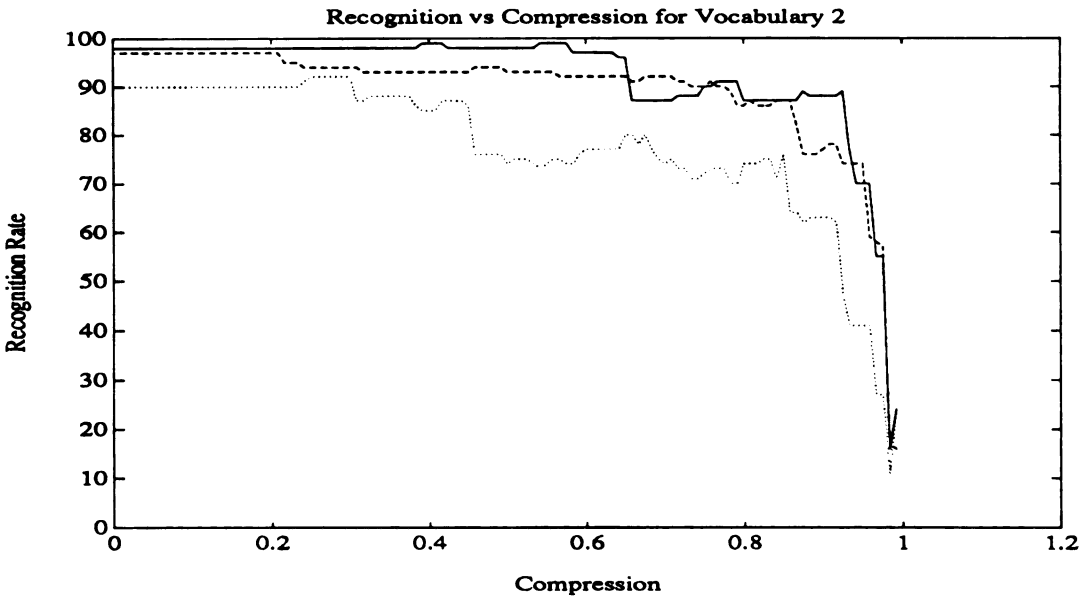

Figure 8: Nearest neighbor compression for Vocabulary 2.
Solid line - Speaker 1; Dashed line - Speaker 2; Dotted line - Speaker 1,2.

confused as easily and get better compression than, say, in Vocabulary 3 (Fig. 9). This is especially noticable for Speaker 1,2.

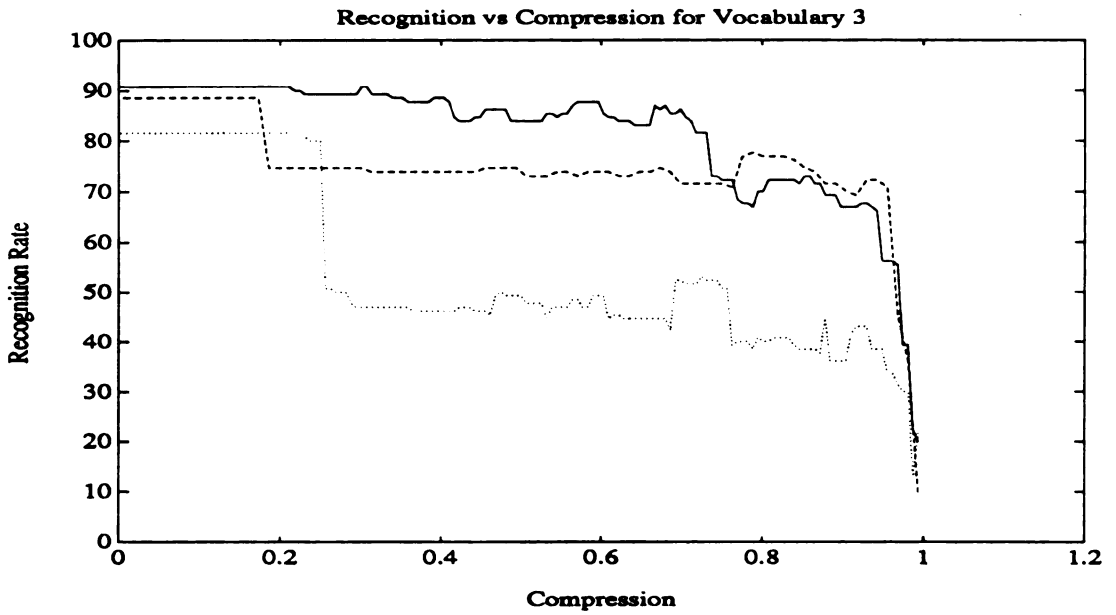

Figure 9: Nearest neighbor compression for Vocabulary 3.
Solid line - Speaker 1; Dashed line - Speaker 2; Dotted line - Speaker 1,2.

Recognition Rate

Fi
Sol

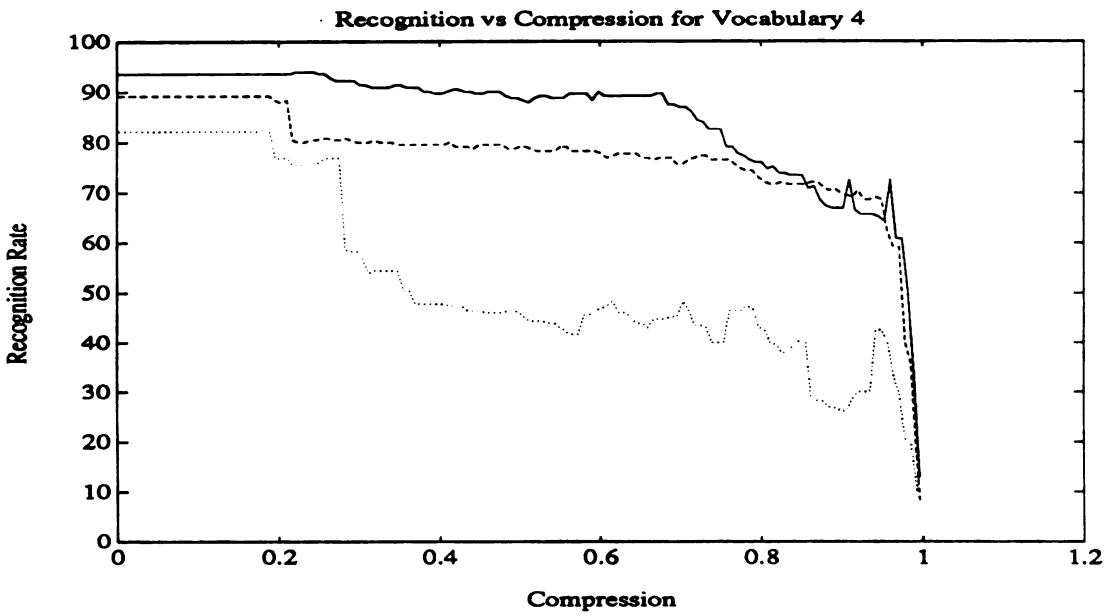

Figure 10: Nearest neighbor compression for Vocabulary 4.
Solid line - Speaker 1; Dashed line - Speaker 2; Dotted line - Speaker 1,2.

Recognition Rate

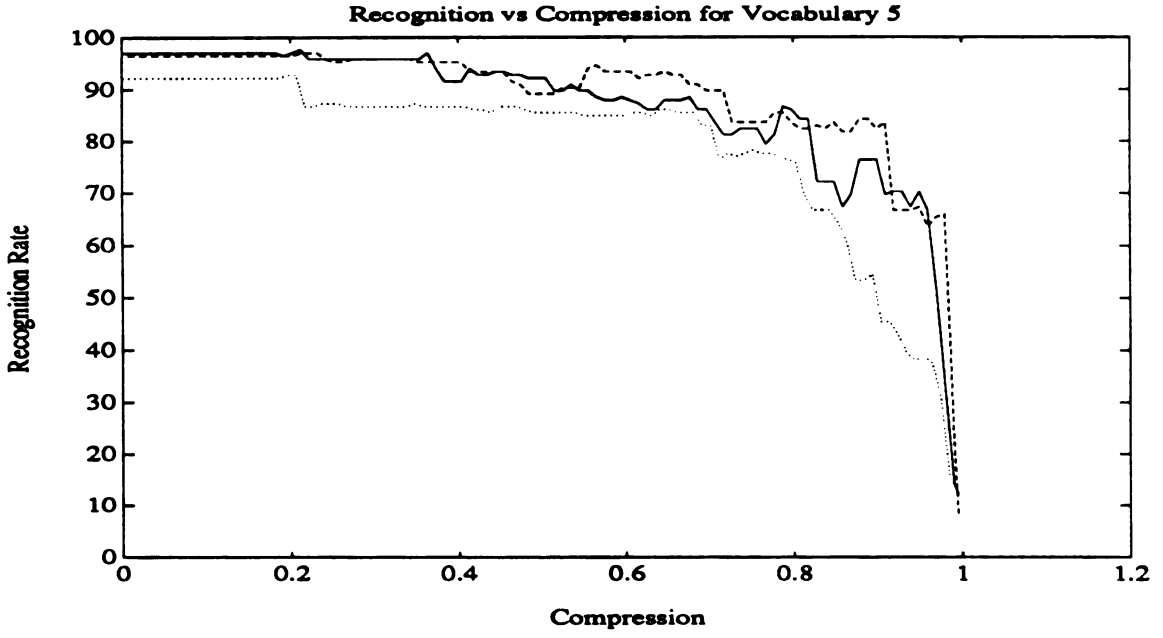

Figure 11: Nearest neighbor compression for Vocabulary 5.
Solid line - Speaker 1; Dashed line - Speaker 2; Dotted line - Speaker 1,2.

3.6 "Linearized" Nearest Neighbor Compression

If we look at Fig. 6 we can see that the values approach 0.95 in a "logarithmic" sense. Thus the eigenvalues that were combined first in the nearest neighbor compression were those close to 0.95. The question arises as to whether the eigenvalues close to 0.95 and close to each other should be combined so soon. Thus we attempt to "linearize" the eigenvalues in an attempt to combined the eigenvalues in a more uniform fashion. To do the "linearization" the following function is used:

$$\lambda_{new} = \frac{(10^{\lambda_{old}} - 1)}{9} \tag{16}$$

where λ_{new} is the modified eigenvalue. The new eigenvalues are then treated just as the eigenvalues were in Section 3.5 in the same nearest neighbor fashion with identical values combined first. The new eigenvalues are used only for the distance measure while the old eigenvalues are used to calculate the new means. Figure 12 shows the result of the "linearization".

Figures 13 through 17 show how the recognition rate is affected by compression using the "linearized" nearest neighbor method for the Vocabularies 1-5. The solid line represents the recognition rates for Speaker 1, the dashed line for Speaker 2, and the dotted line for Speaker 1,2.

The figures for the "linearized" nearest neighbor compression are very similar to the nearest neighbor case. For Speakers 1 and 2 the "linearized" method is slightly better since it has a slightly better compression for the same recognition rate compared with the nearest neighbor method. The figures look similar except that the figures in the "linearized" method are "stretched" a little and this is most noticable at the "knee," where the recognition rate plunges. For Speaker 1,2 the effect is just the opposite. There is a slightly worse performance since the recognition rate falls off a

Value

Fig
Top

bit s

gene

case

neig

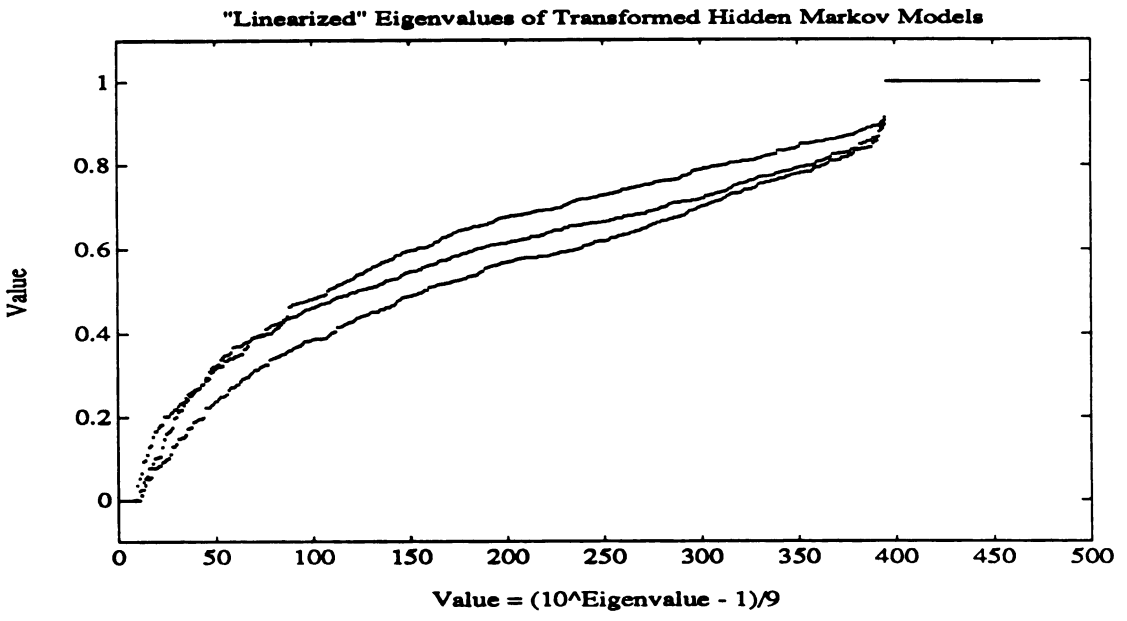

Figure 12: "Linearized" eigenvalues of transformed HMM's.
Top curve - Speaker 1,2; Middle curve - Speaker 1; Bottom curve - Speaker 2.

bit sooner than in the nearest neighbor case. To the extent that these experiments are generalizable, this method seems to work slightly better for the speaker dependent case and slightly worse for the speaker independent case compared with the nearest neighbor method without linearization.

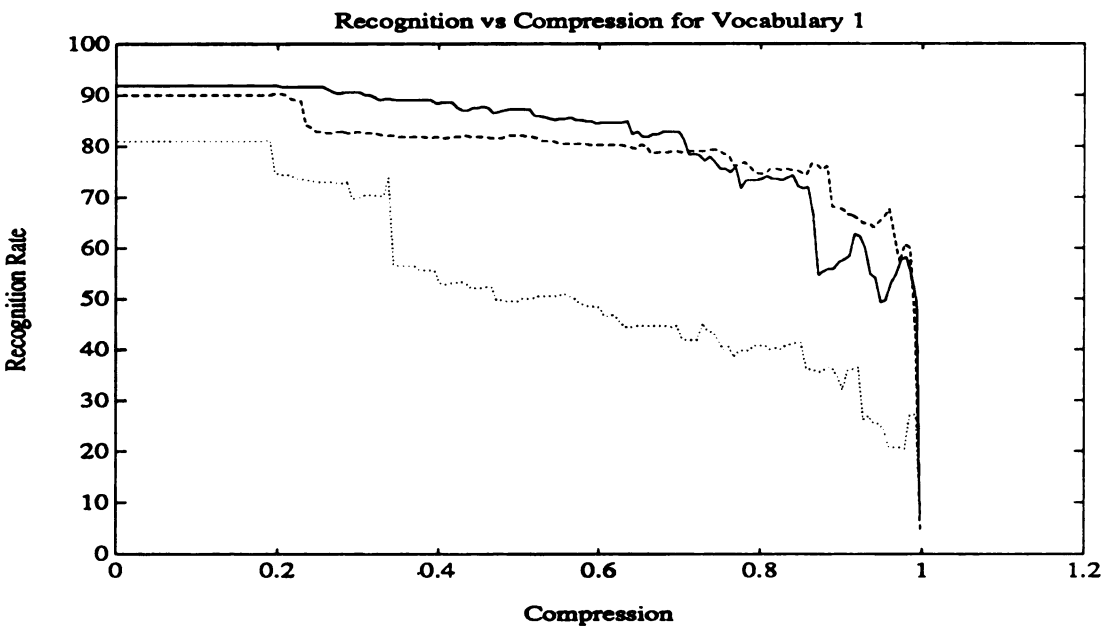

Figure 13: "Linearized nearest neighbor compression for Vocabulary 1.
Solid line - Speaker 1; Dashed line - Speaker 2; Dotted line - Speaker 1,2.

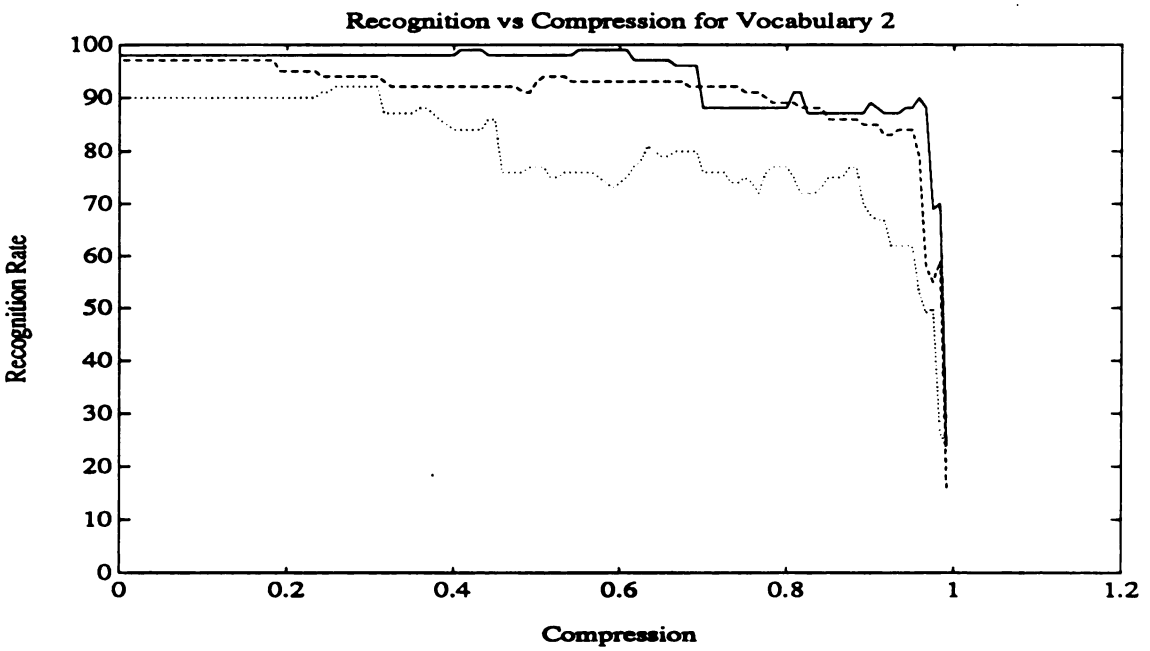

Figure 14: "Linearized nearest neighbor compression for Vocabulary 2.
Solid line - Speaker 1; Dashed line - Speaker 2; Dotted line - Speaker 1,2.

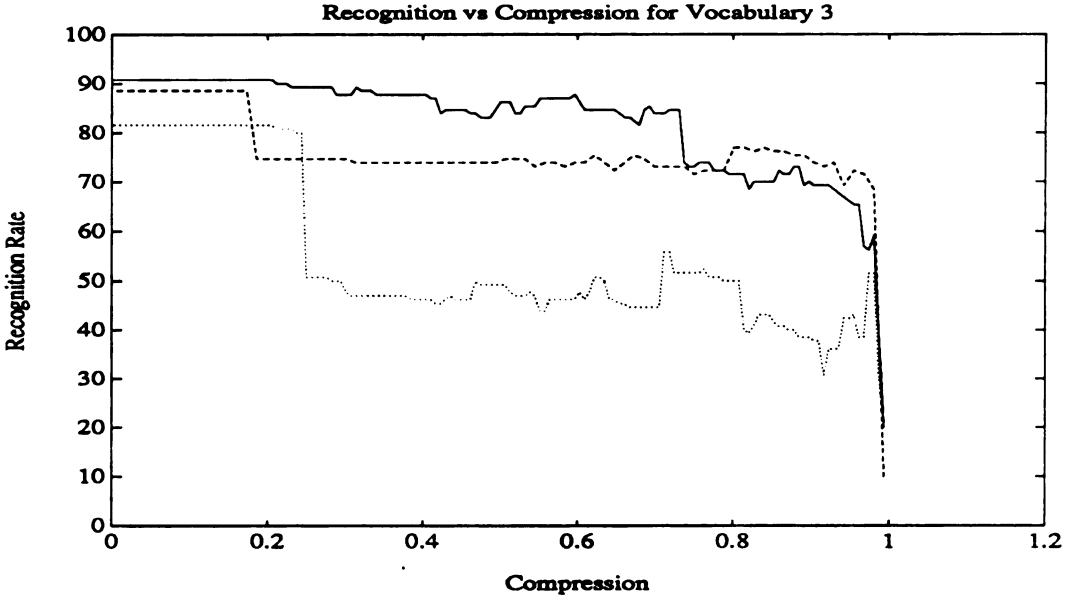

Figure 15: "Linearized nearest neighbor compression for Vocabulary 3.
Solid line - Speaker 1; Dashed line - Speaker 2; Dotted line - Speaker 1,2.

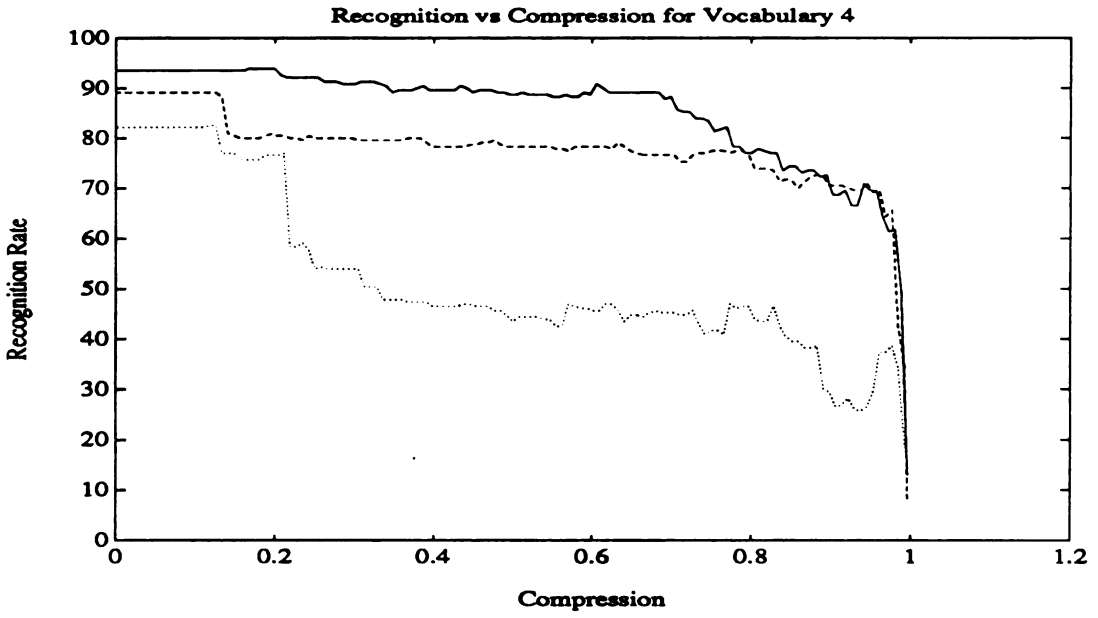

Figure 16: "Linearized nearest neighbor compression for Vocabulary 4.
Solid line - Speaker 1; Dashed line - Speaker 2; Dotted line - Speaker 1,2.

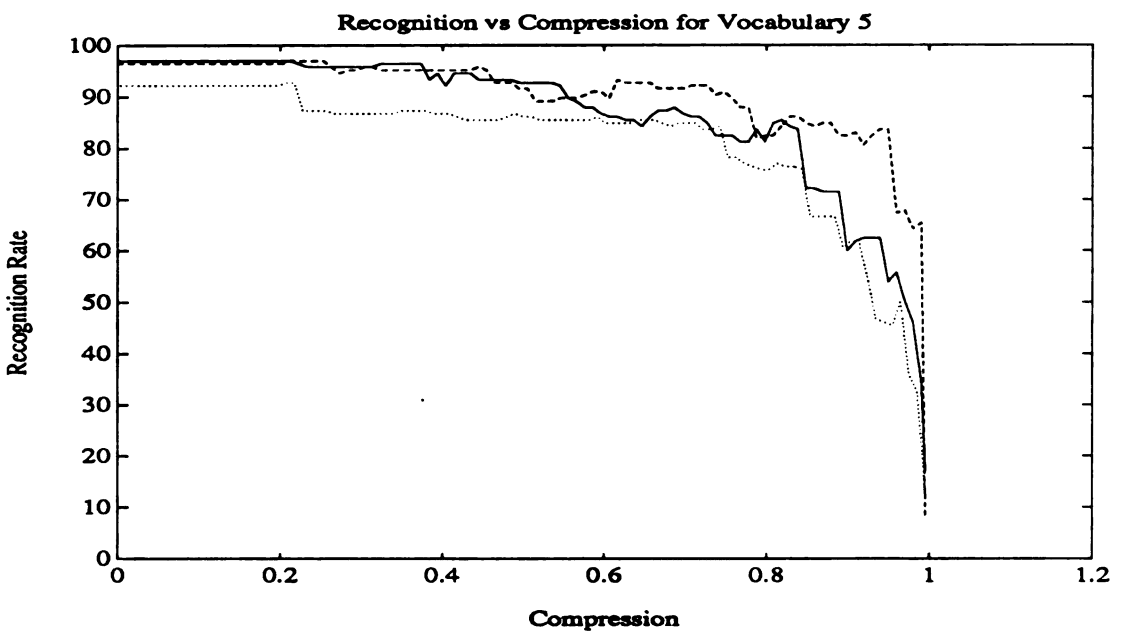

Figure 17: "Linearized nearest neighbor compression for Vocabulary 5.
Solid line - Speaker 1; Dashed line - Speaker 2; Dotted line - Speaker 1,2.

3.7 Bottom Up Compression

Bottom up compression is used to see at what point the small eigenvalues can be neglected. Bottom up compression just means that the smallest values are averaged together before the larger values, beginning with those of smallest magnitude. The only exception is that all the identical values are first compressed (i.e. all the unity eigenvalues are combined first). The bottom up compression works as follows. First all identical values are compressed together since this compression has no effect on recognition rate. Then the smallest two eigenvalues are replaced by their mean followed by a recognition test to see the effect. Next the original smallest three eigenvalues are replace by their mean and a recognition test is run to see the effect. This continues until all eigenvalues are replaced by one value representing the population mean.

Figures 18 through 22 show how the recognition rate is affected by compression using the botton up method for the Vocabularies 1-5. The solid line represents the recognition rates for Speaker 1, the dashed line for Speaker 2, and the dotted line for Speaker 1,2.

The main reason for doing the bottom up compression was to see how far the "plateau" of the initial recognition rate could be extended out before it would drop off. Thus, if say all the eigenvalues less than 0.8 did not contribute much, then the recognition rate would stay the same until the eigenvalues above 0.8 were affected. This turns out not to be the case. As can be seen in Figures 18 through 22 the recognition rate starts dropping as soon as all the identical values are merged together, the only exceptions are for Speaker 1 in Vocabularies 2 and 5, Speaker 2 in Vocabulary 3 and Speaker 1,2 in Vocabularies 2 and 3. In fact the nearest neighbor method did as well or better in extending the initial "plateau." Thus there appears to be no

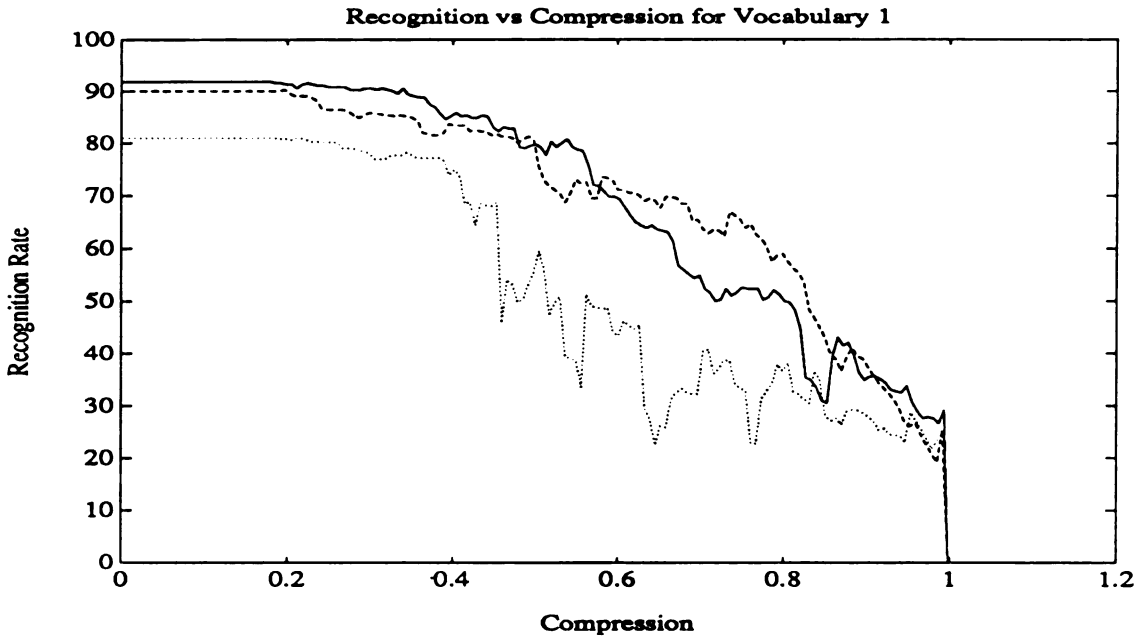

Figure 18: Bottom up compression for Vocabulary 1.
Solid line - Speaker 1; Dashed line - Speaker 2; Dotted line - Speaker 1,2.

threshold below which all the eigenvalues can be readily merged together.

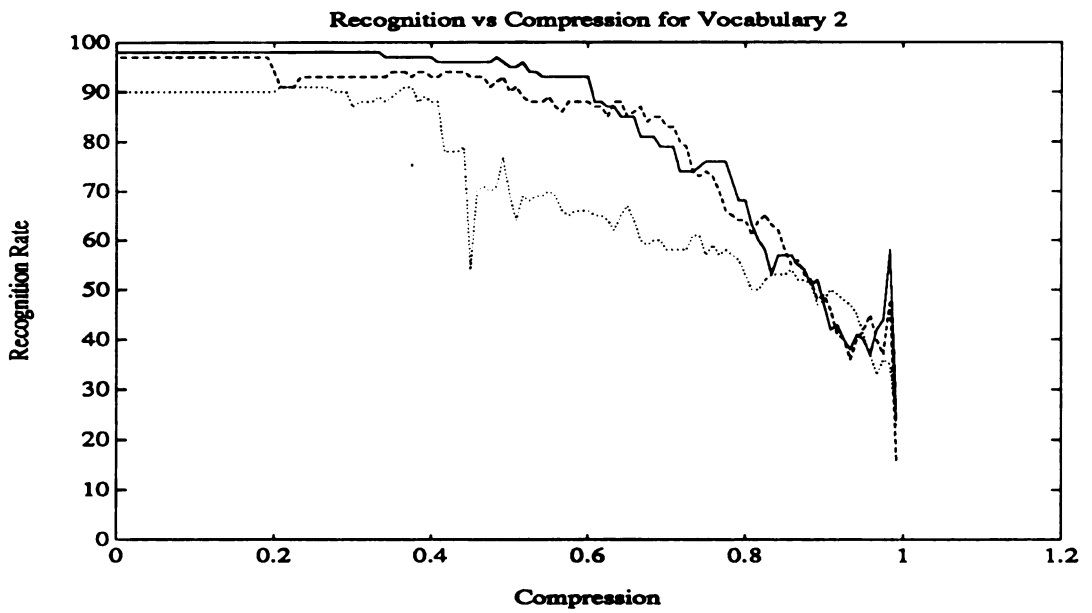

Figure 19: Bottom up compression for Vocabulary 2.
Solid line - Speaker 1; Dashed line - Speaker 2; Dotted line - Speaker 1,2.

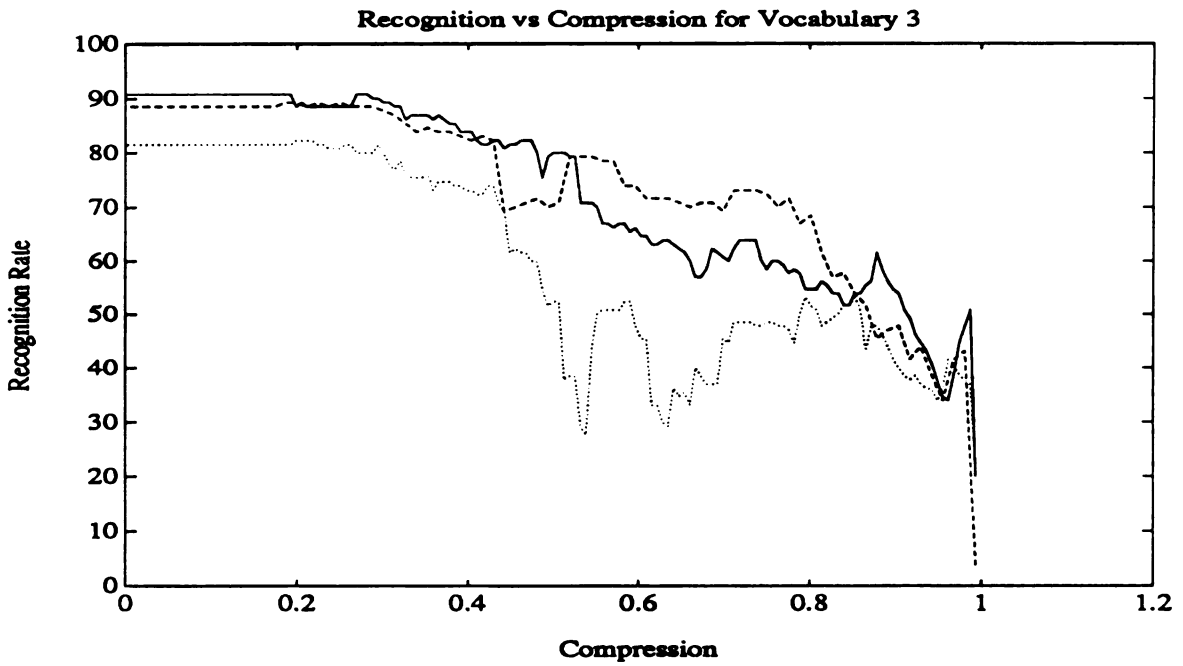

Figure 20: Bottom up compression for Vocabulary 3.
Solid line - Speaker 1; Dashed line - Speaker 2; Dotted line - Speaker 1,2.

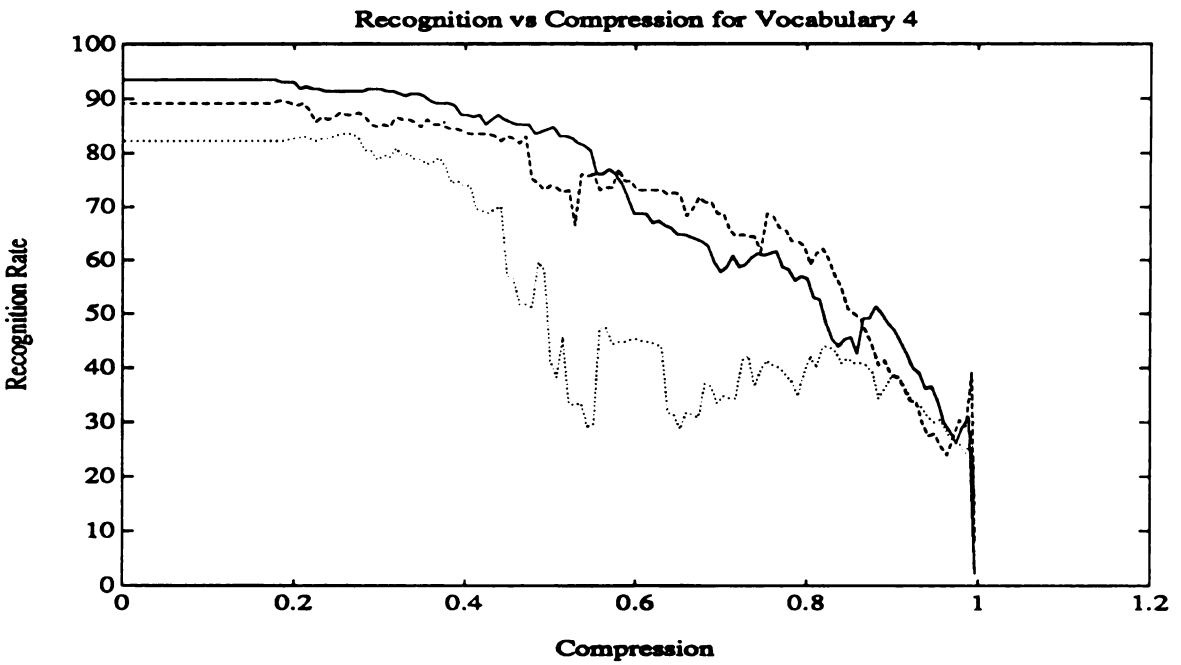

Figure 21: Bottom up compression for Vocabulary 4.
Solid line - Speaker 1; Dashed line - Speaker 2; Dotted line - Speaker 1,2.

Recognition Rate

Fi
So

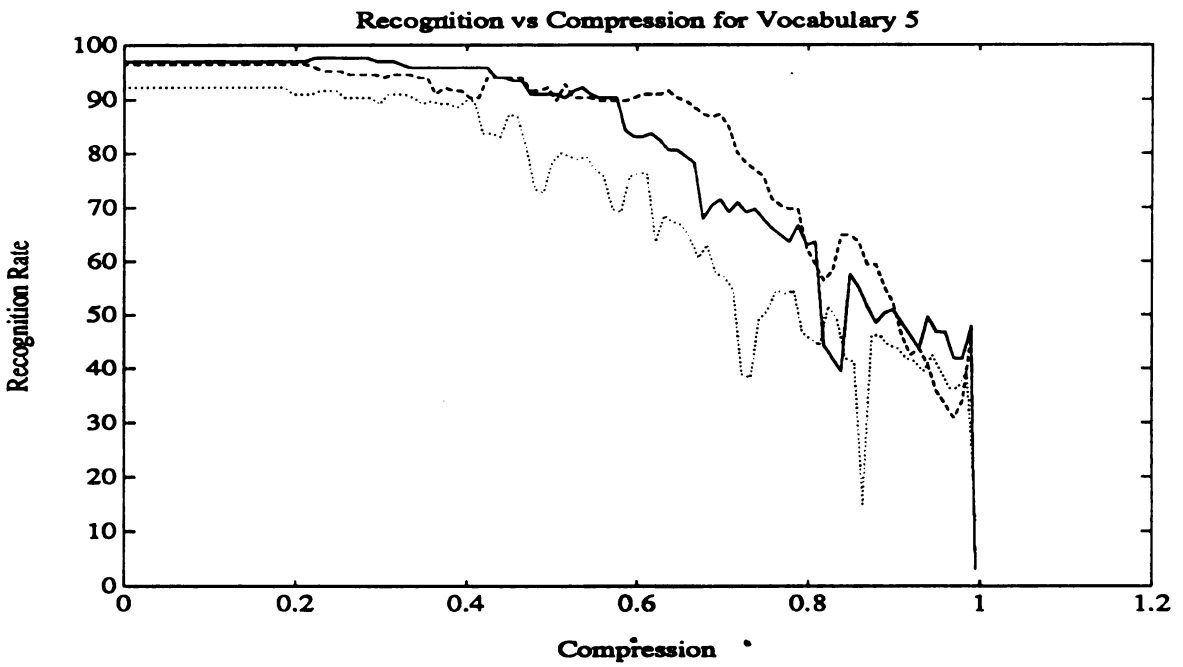

Figure 22: Bottom up compression for Vocabulary 5.
Solid line - Speaker 1; Dashed line - Speaker 2; Dotted line - Speaker 1,2.

4 Conclusions

A procedure has been presented for evaluating the likelihood of a hidden Markov model using only $\mathcal{O}([1-\kappa]NT)$ floating point operations, where N is the number of states, T is the number of observations derived from an utterance, and κ, the *compression index*, can be close to unity depending on the acceptable degradation in recognition rate. This is a reduction from the $\mathcal{O}(3NT)$ to $\mathcal{O}(N^2T)$ operations used in existing algorithms to evaluate the likelihood of an HMM. This is significant for the development of a speech recognizer to be run on a PC for speech disabled individuals.

It has been found that the compression has a greater detrimental impact on the speaker independent case than for the speaker dependent case. This is not critical since the use of the compression technique will be used for people with cerebral palsy on an individual basis. It has also been found that the vocabularies with words that are dissimilar perform better under compression than vocabularies with similar words. This is true for recognition in general. It has been observed that larger vocabularies result in greater computational savings. This is because with larger vocabularies there are more eigenvalues that are very close to each other and can be merged with negligable affect on the recognition rate.

Near-real-time recognition is possible for large vocabularies since the compression can greatly improve the speed at which recognition takes place. This can be seen in the following example. Suppose a vocabulary of 1000 words were used in conjunction with a six state Bakis model, the compression index was $\kappa = 0.9$, and an utterance contained 100 observation symbols. The computation to recognize the word would then drop from 1.8 million flops to 60 thousand flops. For the PC described earlier, this would be the difference between 72 s and 7.2 s to recognize a word. This increased speed comes at the expense of at least some recognition rate degradation however.

It should be pointed out, that though the recognition rate drops as the compression is increased, if a word is misrecognized it is usually found among the words with the highest scores in these experiments. The significance of this can be seen if one realizes that this technique is to be used in a system to help disabled individuals communicate. Thus if a word is misrecognized, the correct word can be quickly found by searching the words with the top scores in a binary fashion, as opposed to having to resort to some other method like spelling the word out.

It is the conclusion of this thesis that the reduction in computational complexity achieved by the transformation and compression of hidden Markov models is feasable for the implementation of an efficient likelihood evaluation to be implemented in a speech recognition unit for disabled individuals.

References

[1] D. Hsu and J.R. Deller, Jr., "On the use of HMM's to recognize cerebral palsy speech: Isolated word case," *Proc. ICASSP '89,* Glasgow, vol. 1, pp. 290-293, May 1989.

[2] D. Hsu, "Computer recognition of nonverbal speech using hidden Markov modelling" (Ph.D. dissertation), Northeastern University, Boston, 1988.

[3] D. Hsu and J.R. Deller, Jr., "Recognition of cerebral palsy speech using hidden Markov modelling: Isolated word case," *Biomedical Measurement, Control, and Informatics*, in review.

[4] S.E. Levinson, "Structural methods in automatic speech recognition," *Proc. IEEE,* vol. 73, pp. 1625-1650, 1985.

[5] L.R. Rabiner and B.H. Juang, "An introduction to hidden Markov models," *IEEE ASSP Magazine,* vol. 3, pp. 4-16, 1986.

[6] C.T. Chen, *Linear Systems Theory and Design,* New York: Holt, Rinehart and Winston, 1984.

[7] J.R. Deller, Jr. and D. Hsu, "An alternative adaptive sequential regression algorithm and its application to the recognition of cerebral palsy speech," *IEEE Trans. Circ. and Sys.*, vol. 34, pp. 782-787, 1987.

[8] S.E. Levinson, L.R. Rabiner, and M.M. Sondhi, "An introduction to the application of the theory of probablistic functions of a Markov process to automatic speech recognition," *Bell Sys. Tech. J.*, vol. 62, pp. 1035-1073, 1983.

[9] P.A. Divijver and J. Kittler, *Pattern Recognition: A Statistical Approach,* London: Prentice-Hall International, 1982.

[10] A.V. Oppenheim and R.W. Schafer, *Digital Signal Processing,* London: Prentice-Hall International, pp.239-250, 1975.

[11] S.A. Davis and P. Mermelstein, "Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences," *IEEE Trans. ASSP.* vol. ASSP-28, pp. 357-366, August 1985.

[12] W. Johnson, F.L. Darley, and D.C. Spriestersbach, *Diagnostic Methods in Speech Pathology,* New York: Harper & Row Publishers, 1963.

[13] A.K. Jain and R.C. Dubes, *Algorithms for Clustering Data,* London: Prentice-Hall International, pp.128-130, 1988.

APPENDICES

A Program 1 - *Wavemark*

This program allows the user to view the data sampled by the *streamer* program by Metrabyte. It is run using the following command line arguments.

Wavemark filename.dat x

where x=0 if the file was produced by *streamer*, x=1 if the file was produced by *Wavemark*, and x=2 if using the cerebral palsy data from Northeastern University.

```
#include <stdlib.h>
#include <graphics.h>
#include <conio.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <ctype.h>


long  stepsize;
int   inc,opt,b,e,M,maxx,maxy;
long  b_index,e_index;
long  FILE_SIZE,FILE_INDEX,display_size,end_val;
   /* Upper case are indexes                        */
   /* in bytes , lower case are indexes in points */
char  bufferb[900],buffere[900];
   /* there are two bytes per point (int)          */
char  outfilename[80],infilename[80];
char  PATH[80];
int   MAXVAL,INFILE_TYPE;
FILE  *infile1;

main(argc,argv)
int argc;
char *argv[];
{
int graphdriver = DETECT,graphmode;
int errorcode;
int inchar;
char *stop;
long filelength();
void change_stepsize();
void change_position();
void change_marker();
void save_points2file();
void change_fileindex();
void change_displaysize();
void change_endindex();
void change_path();
void update_screen();
void initial_screen();
void zoom_screen();
```

```
void listen_word();
void exit();

if( argc <= 1){
   printf("Usage: Wordmark <binary file name>
          <type of file>\n\n");
   printf("       For type of file enter
          <0> for streamer binary file (Metrobyte)
          or <1> for integer binary file.\n");
   printf("       Wordmark saves files in the
   integer binary file format.\n\n");
   printf("Example:  Wordmark  cpdata.dat  0\n");
   exit(0);
}

strcpy(infilename,argv[1]);
INFILE_TYPE=atoi(argv[2]);
printf("infilename=%s  INFILE_TYPE=%d\n"
       ,infilename,INFILE_TYPE);
strcpy(PATH,"\\");
FILE_INDEX=0;
display_size=50000;

if ( (infile1 = fopen(infilename, "rb")) == NULL) {
   printf("fopen failed for file %s.\n",infilename);
   exit(0);
}
printf("file %s has been opened\n",infilename);
if ( (FILE_SIZE = filelength(fileno(infile1))) != -1L) {
   printf("Size of file = %ld bytes = %ld data points\n"
          ,FILE_SIZE,FILE_SIZE/2);
}
else{
   printf("ERROR getting file size\n");
}
if( (FILE_INDEX + display_size*2) > FILE_SIZE )
   display_size = FILE_SIZE - FILE_INDEX;

MAXVAL=2048;
M=(int)ceil((double)display_size/100);
printf("M=%d  display_size=%ld fileindex=%ld
       filesize=%ld\n",M,display_size,FILE_INDEX,FILE_SIZE);
```

```
initgraph(&graphdriver,&graphmode,"c:\\tc");
errorcode = graphresult();
if( errorcode != 0 ) {
   printf("Graphics error: %s\n",grapherrormsg(errorcode));
   exit(1);
}
maxx = getmaxx();
maxy = getmaxy();

initial_screen();
change_fileindex(0);
update_screen();

inchar=0;
while( inchar != 120  )                              /* x  */
{
inchar=getch();
switch(inchar)
   {
   case 43 :  change_stepsize(1);       break;     /* +   */
   case 45 :  change_stepsize(-1);      break;     /* -   */
   case 61 :  change_marker(1);         break;     /* F3  */
   case 62 :  change_marker(2);         break;     /* F4  */
   case 63 :  change_endindex(-1);      break;     /* F5  */
   case 64 :  change_endindex(1);       break;     /* F6  */
   case 65 :  change_displaysize(-1);   break;     /* F7  */
   case 66 :  change_displaysize(1);    break;     /* F8  */
   case 67 :  change_fileindex(-1);     break;     /* F9  */
   case 68 :  change_fileindex(1);      break;     /* F10 */
   case 75 :  change_position(-1);      break;     /* <-  */
   case 77 :  change_position(1);       break;     /* ->  */
   case 102:  save_points2file();       break;     /* f   */
   case 108:  listen_word();            break;     /* l   */
   case 117:  update_screen();          break;     /* u   */
   case 122:  zoom_screen();            break;     /* z   */
   default:  break;
   }
}
closegraph();
exit(0);
}
```

```
/*********************************************************/
void save_points2file(){
char string[80],string1[80],string2[80],string3[80],ch[2];
unsigned inbuffer[100];
int inkey,inval,length,i,j,outbuffer[100];
long index_c;
FILE *outfile;

char *ltoa();
char *itoa();

setviewport(0,370,300,390,0);
setbkcolor(0);
clearviewport();
setviewport(0,0,maxx,maxy,0);
strcpy(string,"Enter output file name >>");
setcolor(60);
outtextxy(5,370,string);

strcpy(string1,"\0");
ch[1]='\0';
inkey=0;
while( inkey != 27 )   /* CR || esc */
   {
   inkey=getch();
   if( isalnum(inkey) != 0 || inkey == 46){
                          /* alpha num or "." */
      ch[0]=inkey;
      strcat(string1,ch);
      /*   length=strlen(string1);*/
      setviewport(210,370,639,390,0);
      setbkcolor(0);
      clearviewport();
      setviewport(0,0,maxx,maxy,0);
      setcolor(60);
      outtextxy(215,370,string1);
   }
   if( inkey == 8 ){
      length=strlen(string1);
      string1[length-1]='\0';
      setviewport(210,370,639,390,0);
      setbkcolor(0);
```

```
      clearviewport();
      setviewport(0,0,maxx,maxy,0);
      setcolor(60);
      outtextxy(215,370,string1);
   }
   if( inkey == 13 ){
      setviewport(0,370,639,390,0);
      setbkcolor(0);
      clearviewport();
      setviewport(0,0,maxx,maxy,0);
      setcolor(62);
      strcpy(string,"Writing points to file    ");
      strcat(string,string1);
      outtextxy(5,370,string);
      outfile = fopen(string1, "wb");
      fseek(infile1,FILE_INDEX,SEEK_SET);
      M=(int)ceil((double)display_size/100);
      index_c = 0;
      for( i=0; i<M; i++){
         fread((void *)inbuffer
               ,sizeof(unsigned),100,infile1);
         for(j=0; j<100; j++) {
            if( INFILE_TYPE == 0 ) {
               inval = inbuffer[j]>>4;
               inval -= 2048;
            }
/* File structure from Metrobyte's streamer sampling
   program. The data is in the 12 MSB's  (the 4 LSB's
   contain the channel info).  Thus the data is shifted
   4 bits to right and 2048 subtracted to get the bipolar
   format (unsigned the values range from 0-4098 for
   12 bits) */
            if( INFILE_TYPE == 1 )
               inval = inbuffer[j];
            if( INFILE_TYPE == 2 )
               inval = inbuffer[j]-2048;
            outbuffer[j] = inval;
            index_c++;
         }
         fwrite((void *)outbuffer, sizeof(int),100,outfile);
      }
      fclose(outfile);
```

```
      setviewport(0,370,639,390,0);
      setbkcolor(0);
      clearviewport();
      setviewport(0,0,maxx,maxy,0);
      break;
   }
   if( inkey == 27 ){
      setviewport(0,370,639,390,0);
      setbkcolor(0);
      clearviewport();
      setviewport(0,0,maxx,maxy,0);
      break;
   }
 }
}

/*********************************************************/
void change_stepsize(inc)
int inc;
{
char string[80];
if( inc == -1 || inc == 1 ) {
   switch(stepsize){
      case     1: if(inc == 1 ) {
                     stepsize=10;
                     setcolor(56);
                     strcpy(string,"STEP SIZE : 1    10
             100    1,000  10,000   100,000   1,000,000");
                     outtextxy(10,350,string);
                     setcolor(62);
                     strcpy(string,"STEP SIZE       10");
                     outtextxy(10,350,string);
                  }
                  break;
      case    10: if(inc == -1 ){
                     stepsize=1;
                     setcolor(56);
                     strcpy(string,"STEP SIZE : 1    10
             100    1,000   10,000   100,000   1,000,000");
                     outtextxy(10,350,string);
                     setcolor(62);
                     strcpy(string,"STEP SIZE   1");
```

```
            outtextxy(10,350,string);
         }
         if(inc == 1 ) {
            stepsize=100;
            setcolor(56);
            strcpy(string,"STEP SIZE : 1    10
      100    1,000    10,000    100,000    1,000,000");
            outtextxy(10,350,string);
            setcolor(62);
            strcpy(string,"STEP SIZE         100");
            outtextxy(10,350,string);
         }
         break;
case   100: if(inc == -1 ){
            stepsize=10;
            setcolor(56);
            strcpy(string,"STEP SIZE : 1    10
      100    1,000    10,000    100,000    1,000,000");
            outtextxy(10,350,string);
            setcolor(62);
            strcpy(string,"STEP SIZE        10");
            outtextxy(10,350,string);
         }
         if(inc == 1 ){
            stepsize=1000;
            setcolor(56);
            strcpy(string,"STEP SIZE : 1    10
      100    1,000    10,000    100,000    1,000,000");
            outtextxy(10,350,string);
            setcolor(62);
            strcpy(string,"STEP SIZE       1,000");
            outtextxy(10,350,string);
         }
         break;
case  1000: if(inc == -1 ) {
            stepsize=100;
            setcolor(56);
            strcpy(string,"STEP SIZE : 1    10
      100    1,000  10,000    100,000    1,000,000");
            outtextxy(10,350,string);
            setcolor(62);
            strcpy(string,"STEP SIZE         100");
```

```
                outtextxy(10,350,string);
             }
             if(inc == 1 ){
                stepsize=10000;
                setcolor(56);
                strcpy(string,"STEP SIZE : 1    10
         100    1,000    10,000   100,000   1,000,000");
                outtextxy(10,350,string);
                setcolor(62);
                strcpy(string,"STEP SIZE     10,000");
                outtextxy(10,350,string);
             }
             break;
 case 10000: if(inc == -1 ){
                stepsize=1000;
                setcolor(56);
                strcpy(string,"STEP SIZE : 1    10
         100    1,000   10,000   100,000   1,000,000");
                outtextxy(10,350,string);
                setcolor(62);
                strcpy(string,"STEP SIZE      1,000");
                outtextxy(10,350,string);
             }
             if(inc == 1 ){
                stepsize=100000;
                setcolor(56);
                strcpy(string,"STEP SIZE : 1    10
         100    1,000    10,000   100,000   1,000,000");
                outtextxy(10,350,string);
                setcolor(62);
                strcpy(string,"STEP SIZE     100,000");
                outtextxy(10,350,string);
             }
             break;
case 100000: if(inc == -1 ){
                stepsize=10000;
                setcolor(56);
                strcpy(string,"STEP SIZE : 1    10
         100    1,000    10,000   100,000   1,000,000");
                outtextxy(10,350,string);
                setcolor(62);
                strcpy(string,"STEP SIZE     10,000");
```

```
                    outtextxy(10,350,string);
                  }
                  if(inc == 1 ){
                    stepsize=1000000;
                    setcolor(56);
                    strcpy(string,"STEP SIZE : 1    10
            100    1,000    10,000   100,000   1,000,000");
                    outtextxy(10,350,string);
                    setcolor(62);
                    strcpy(string,"STEP SIZE  1,000,000");
                    outtextxy(10,350,string);
                  }
                  break;
    case  1000000: if(inc == -1 ){
                    stepsize=100000;
                    setcolor(56);
                    strcpy(string,"STEP SIZE : 1    10
            100    1,000   10,000   100,000   1,000,000");
                    outtextxy(10,350,string);
                    setcolor(62);
                    strcpy(string,"STEP SIZE    100,000");
                    outtextxy(10,350,string);
                   }
                   break;
       default: break;
      }
   }
}
/*********************************************************/
void change_position(nxt)
int nxt;
{
char string1[10];
int oldb,olde;

char *ltoa();

if( opt == 1 || nxt == 0)
   {
   oldb=b;
   b_index = b_index + stepsize*nxt;
   if( b_index < FILE_INDEX/2 )
```

```
      b_index = FILE_INDEX/2;
   if( b_index > e_index)
      b_index = e_index - display_size/maxx;
   b=(b_index - FILE_INDEX/2)*maxx/display_size;
   setviewport(200,330,300,340,0);
   setbkcolor(0);
   clearviewport();
   setviewport(0,0,maxx,maxy,0);
   ltoa(b_index,string1,10);
   setcolor(63);
   outtextxy(200,330,string1);
   putimage(oldb,10,bufferb,COPY_PUT);
   getimage(b,10,b,225,bufferb);
   setcolor(63);
   line(b,10,b,225);
   }

if( opt == 2 || nxt == 0)
   {
   olde=e;
   e_index = e_index + stepsize*nxt;
   if( e_index < b_index )
      e_index = b_index + display_size/maxx;
   if( e_index > FILE_INDEX/2 + display_size )
      e_index = FILE_INDEX/2 + display_size;
   e=(e_index - FILE_INDEX/2)*maxx/display_size;
   setviewport(470,330,570,340,0);
   setbkcolor(0);
   clearviewport();
   setviewport(0,0,maxx,maxy,0);
   ltoa(e_index,string1,10);
   setcolor(57);
   outtextxy(470,330,string1);
   putimage(olde,10,buffere,COPY_PUT);
   getimage(e,10,e,225,buffere);
   setcolor(57);
   line(e,10,e,225);
   }

}

/*********************************************************/
```

```
void change_marker(mrk)
int mrk;
{
if ( mrk == 1 )
   opt = 1;
if ( mrk == 2 )
   opt = 2;

}

/*************************************************************/
void change_fileindex(nxt)
int nxt;
{
char string[40],string1[40];
FILE_INDEX=FILE_INDEX+stepsize*2*nxt;
if ( FILE_INDEX < 0 )
   FILE_INDEX=0;
if ( FILE_INDEX > FILE_SIZE - 20)
   FILE_INDEX=FILE_SIZE-20;
ltoa(FILE_INDEX/2,string1,10);
setviewport(10,280,110,290,0);
setbkcolor(0);
clearviewport();
setviewport(0,0,maxx,maxy,0);
setcolor(58);
outtextxy(10,280,string1);

end_val = FILE_INDEX/2 + display_size;
if ( end_val > FILE_SIZE/2 )
   {
   end_val = FILE_SIZE/2;
   display_size = FILE_SIZE/2 - FILE_INDEX/2;
   }
ltoa(end_val,string1,10);
setviewport(540,280,639,290,0);
setbkcolor(0);
clearviewport();
setviewport(0,0,maxx,maxy,0);
setcolor(58);
outtextxy(540,280,string1);
```

```
ltoa(display_size,string1,10);
strcpy(string,"<-- ");
strcat(string,string1);
strcat(string," -->");
setviewport(250,280,390,290,0);
setbkcolor(0);
clearviewport();
setviewport(0,0,maxx,maxy,0);
setcolor(58);
outtextxy(250,280,string);

}
/*********************************************************/
void change_endindex(nxt)
int nxt;
{
char string[40],string1[40];

end_val = end_val + stepsize*nxt;
if ( end_val > FILE_SIZE/2 )
   end_val = FILE_SIZE/2;
if ( end_val < FILE_INDEX/2 )
   end_val = FILE_INDEX/2 + 10;
display_size = end_val - FILE_INDEX/2;
ltoa(end_val,string1,10);
setviewport(540,280,639,290,0);
setbkcolor(0);
clearviewport();
setviewport(0,0,maxx,maxy,0);
setcolor(58);
outtextxy(540,280,string1);

ltoa(display_size,string1,10);
strcpy(string,"<-- ");
strcat(string,string1);
strcat(string," -->");
setviewport(250,280,390,290,0);
setbkcolor(0);
clearviewport();
setviewport(0,0,maxx,maxy,0);
setcolor(58);
outtextxy(250,280,string);
```

```
}

/*********************************************************/
void change_displaysize(nxt)
int nxt;
{
char string[40],string1[40];
display_size = display_size + stepsize*nxt;
if ( display_size < 10 )
   display_size=10;
if ( display_size > FILE_SIZE/2 - FILE_INDEX/2 )
   display_size=FILE_SIZE/2 - FILE_INDEX/2;
ltoa(display_size,string1,10);
strcpy(string,"<-- ");
strcat(string,string1);
strcat(string," -->");
setviewport(250,280,390,290,0);
setbkcolor(0);
clearviewport();
setviewport(0,0,maxx,maxy,0);
setcolor(58);
outtextxy(250,280,string);

end_val = FILE_INDEX/2 + display_size;
if ( end_val > FILE_SIZE/2 )
   {
   end_val = FILE_SIZE/2;
   display_size = FILE_SIZE/2 - FILE_INDEX/2;
   }
ltoa(end_val,string1,10);
setviewport(540,280,639,290,0);
setbkcolor(0);
clearviewport();
setviewport(0,0,maxx,maxy,0);
setcolor(58);
outtextxy(540,280,string1);

}
/*********************************************************/
void update_screen()
{
```

```
char string[80],string1[40];
int x1,x2,y1,y2,inchar;
int i,j;
long index_1;
long index_2,inc_value;
unsigned buffer[100];
int inval;

void change_fileindex();

setviewport(0,10,639,225,0);
setbkcolor(0);
clearviewport();
setviewport(0,0,maxx,maxy,0);
setcolor(58);

fseek(infile1,FILE_INDEX,SEEK_SET);
M=(int)ceil((double)display_size/100);
if( M < 2000 ){
   MAXVAL=0;
   for( i=0; i<M; i++) {
      fread((void *)buffer, sizeof(unsigned),100,infile1);
      for(j=0; j<100; j++)
   {
   if( INFILE_TYPE == 0 ){
      inval = buffer[j]>>4;
      inval -= 2048;
   }
   if( INFILE_TYPE == 1 )
      inval = buffer[j];
   if( INFILE_TYPE == 2 )
      inval = buffer[j]-2048;
   if( MAXVAL < abs(inval) )
      MAXVAL = abs(inval);
   }
}
}
else
   MAXVAL=2048;
itoa(MAXVAL,string1,10);
strcpy(string,"Amplitude of Waveform");
setviewport(220,295,390,315,0);
```

```
setbkcolor(0);
clearviewport();
setviewport(0,0,maxx,maxy,0);
setcolor(2);
outtextxy(220,295,string);
outtextxy(290,305,string1);

setcolor(58);
x1=0;
y1=120;
fseek(infile1,FILE_INDEX,SEEK_SET);
index_1=0;
M=(int)ceil((double)display_size/100);
for( i=0; i<M; i++)
   {
   if( kbhit() != 0 ) /* abort drawing if esc is entered */
      {
      inchar=getch();
      if( inchar == 27  )                  /* esc = 27 */
         break;
      }
   fread((void *)buffer, sizeof(unsigned),100,infile1);
   for(j=0; j<100; j++)
      {
      if( INFILE_TYPE == 0 ){
         inval = buffer[j]>>4;
         inval -= 2048;
      }
      if( INFILE_TYPE == 1 )
         inval = buffer[j];
      if( INFILE_TYPE == 2 )
         inval = buffer[j]-2048;
      x2=floor((double)maxx
               /(double)display_size*(double)index_1);
      y2=floor(-(double)inval/(double)MAXVAL*100+120);
      line(x1,y1,x2,y2);
      x1=x2;
      y1=y2;
      index_1++;
      }
   }
```

```
b_index = FILE_INDEX/2;
e_index = FILE_INDEX/2 + display_size;
change_position(0);

setcolor(63);
line(0,235,639,235);
line(0,0,639,0);
for( i=0; i<11; i++)
   {
   x1=i*64-1;
   if( x1 < 0 )
      x1=0;
   line(x1,230,x1,235);
   line(x1,0,x1,5);
   }
setviewport(0,240,639,260,0);
setbkcolor(0);
clearviewport();
setviewport(0,0,maxx,maxy,0);
setcolor(63);
inc_value = display_size/10;
for( i=1; i<10; i++)
   {
   index_2 = floor(FILE_INDEX/2 + inc_value*i);
   x1=i*64-1;
   ltoa(index_2,string1,10);
   if ( i%2 == 0 )
      outtextxy(x1,250,string1);
   else
      outtextxy(x1,240,string1);
   }
}

/*********************************************************/
void initial_screen()
{
char string[80],string1[80];

setbkcolor(56);
setcolor(2);
ltoa(FILE_SIZE/2,string1,10);
strcpy(string,"Total points");
```

```
outtextxy(540,295,string1);
outtextxy(540,305,string);
setcolor(3);
strcpy(string,"  Begin Marker <F3 > <F4 > End Marker");
outtextxy(5,400,string);
strcpy(string,"   - End Point <F5 > <F6 > + End Point");
outtextxy(5,410,string);
strcpy(string,"- Display Size <F7 > <F8 > + Display Size");
outtextxy(5,420,string);
strcpy(string," - Start Point <F9 > <F10> + Start Point");
outtextxy(5,430,string);
strcpy(string,"   - Step Size < - > < + > + Step Size");
outtextxy(5,440,string);
strcpy(string,"   Move Marker <<- > < ->> Move Marker");
outtextxy(5,450,string);
strcpy(string,"<u>  Update Screen");
outtextxy(380,400,string);
strcpy(string,"<z>  Zoom Screen");
outtextxy(380,410,string);
strcpy(string,"<f>  Save Marked points to FILE");
outtextxy(380,420,string);
strcpy(string,"<l>  Listen to Marked Portion");
outtextxy(380,430,string);
strcpy(string,"<x>  Exit");
outtextxy(380,440,string);
/*strcpy(string,"<c>  Change directory Path");
outtextxy(380,450,string);*/
setcolor(56);
strcpy(string,"STEP SIZE : 1   10   100   1,000
  10,000   100,000   1,000,000");
outtextxy(10,350,string);
setcolor(62);
strcpy(string,"STEP SIZE                  1,000");
outtextxy(10,350,string);
stepsize=1000;
opt=1;
b_index = FILE_INDEX/2;
b=(b_index - FILE_INDEX/2)*maxx/display_size;
e_index = FILE_INDEX/2 + display_size;
e=(e_index - FILE_INDEX/2)*maxx/display_size;
getimage(b,10,b,225,bufferb);
getimage(e,10,e,225,buffere);
```

```
ltoa(b_index,string1,10);
strcpy(string,"Beginning point = ");
setcolor(2);
outtextxy(50,330,string);
setcolor(63);
outtextxy(200,330,string1);
ltoa(e_index,string1,10);
strcpy(string,"Ending point = ");
setcolor(2);
outtextxy(340,330,string);
setcolor(57);
outtextxy(470,330,string1);
}

/*************************************************************/
void zoom_screen()
{
FILE_INDEX = b_index*2;
display_size = e_index - b_index;
update_screen();
change_fileindex(0);
}

/*************************************************************/
void change_path()
{
int inkey;
int length;
char string[80],ch[2];

void delay();

setviewport(0,370,300,390,0);
setbkcolor(0);
clearviewport();
setviewport(0,0,maxx,maxy,0);
strcpy(string,"Enter directory path   >>");
setcolor(60);
outtextxy(5,370,string);

strcpy(PATH,"\0");
ch[1]='\0';
```

```
inkey=0;
while( inkey != 27 )   /* CR */
   {
   inkey=getch();
   if( isalnum(inkey) != 0 || inkey == 58 || inkey == 92 )
                              /* alpha num or ":" or "\" */
      {
      ch[0]=inkey;
      strcat(PATH,ch);
      setviewport(410,370,639,390,0);
      setbkcolor(0);
      clearviewport();
      setviewport(0,0,maxx,maxy,0);
      setcolor(60);
      outtextxy(415,370,PATH);
      }
   if( inkey == 8 )
      {
      length=strlen(PATH);
      PATH[length-1]='\0';
      setviewport(410,370,639,390,0);
      setbkcolor(0);
      clearviewport();
      setviewport(0,0,maxx,maxy,0);
      setcolor(60);
      outtextxy(415,370,PATH);
      }
   if( inkey == 13 )
      {
      setviewport(0,370,639,390,0);
      setbkcolor(0);
      clearviewport();
      setviewport(0,0,maxx,maxy,0);
      strcpy(string,"Path name entered >>    ");
      strcat(string,PATH);
      setcolor(62);
      outtextxy(5,370,string);
      delay(1000);
      setviewport(0,370,639,390,0);
      setbkcolor(0);
      clearviewport();
      setviewport(0,0,maxx,maxy,0);
```

```
      break;
      }
   if( inkey == 27 )    /* esc */
      {
      setviewport(0,370,639,390,0);
      setbkcolor(0);
      clearviewport();
      setviewport(0,0,maxx,maxy,0);
      break;
      }
   }

}
/*************************************************************/
void  listen_word()
{
long i,j;
int port;
unsigned inbuffer[1];
int inval,outnum;
long delay,total,k;
char string[80];

void outport();

port=820;  /*  816 + 4 */  /* 816 = 330H */
port=772;  /*  768 + 4 */  /* 768 = 300H */
if( INFILE_TYPE == 0 )
   delay=6;
if( INFILE_TYPE == 1 )
   delay=5;
if( INFILE_TYPE == 2 )
   delay=7;

total = e_index - b_index;
fseek(infile1,b_index*2,SEEK_SET);
for( i=0; i<total; i++)
   {
   fread((void *)inbuffer, sizeof(unsigned),1,infile1);
   if( INFILE_TYPE == 0 )
      {
      inval = inbuffer[0]>>4;
```

```
      outnum = inval<<4;
      }
   if( INFILE_TYPE == 1 )
      outnum = inbuffer[0]+2048;
   if( INFILE_TYPE == 2 )
      outnum = inbuffer[0];
   for( k=0; k<delay; k++);
   outport(port,outnum);
   }
}
```

B Program 2 - *Cepstrum*

This program computes the mel-cepstrum coefficients of an input speech data file produced by *Wavemark*.

```
#include <stdio.h>
#include <ctype.h>
#include <string.h>
#include <time.h>
#include <math.h>
#define W_SIZE                      256
        /* Hamming window size */
#define CEP_SIZE                      6
        /* number of cepstral coefficients */
#define FFT_SIZE                   4096
        /* number of points transformed real & imag */
#define PI 3.14159265358979


int count;
double mel[21];                 /* mel_frequency energy */
double mf[22];                  /* mel frequency scale  */
double wf[FFT_SIZE];
       /* 256 samples plus zero padding to 4096 pts */
double cep_coef[CEP_SIZE];
double fftmag[1024];
int sdata[128],buffer[128],stemp;
FILE *outfile;

int main(){
   int i,h,s,t;
   long S,T;
   int numread;
   char infilename1[80],infilename2[80],outfilename[80];
   char instring[40],numstring[5];
   FILE *infile,*file;
   void exit();
   void mel_cepstrum();
   void mel_freq();
   void mel_energy();
   void window();
   void fft();

   mel_freq();

   strcpy(infilename1,"dwrds.dat");
   if ( (infile = fopen(infilename1, "r")) == NULL){
```

```
   printf("fopen failed for  %s.\n",infilename1);
   exit(0); }
S=T=0;
while( fscanf(infile,"%s\n",instring) != EOF){
   for(h=1; h<16; h++){
      strcpy(infilename2,"d:\\sdata\\");
      strcat(infilename2,instring);
      strcat(infilename2,".d");
      switch(h){
         case 1  : strcpy(numstring,"01");
                   break;
         case 2  : strcpy(numstring,"02");
                   break;
         case 3  : strcpy(numstring,"03");
                   break;
         case 4  : strcpy(numstring,"04");
                   break;
         case 5  : strcpy(numstring,"05");
                   break;
         case 6  : strcpy(numstring,"06");
                   break;
         case 7  : strcpy(numstring,"07");
                   break;
         case 8  : strcpy(numstring,"08");
                   break;
         case 9  : strcpy(numstring,"09");
                   break;
         case 10 : strcpy(numstring,"10");
                   break;
         case 11 : strcpy(numstring,"11");
                   break;
         case 12 : strcpy(numstring,"12");
                   break;
         case 13 : strcpy(numstring,"13");
                   break;
         case 14 : strcpy(numstring,"14");
                   break;
         case 15 : strcpy(numstring,"15");
                   break;
         default : break;
      }
      strcat(infilename2,numstring);
```

```
      if ( (file = fopen(infilename2, "rb")) == NULL){
         printf("fopen failed for infilename2 %s\n"
                ,infilename2);
         exit(0); }
      printf("reading in data from file %s\n"
             ,infilename2);
      strcpy(outfilename,"c:\\cdata2\\");
      strcat(outfilename,instring);
      strcat(outfilename,".j");
      strcat(outfilename,numstring);
      outfile = fopen(outfilename, "wb");
      printf("Writing cepstral coeffs. to file %s\n"
             ,outfilename);
      s=t=0;
      numread=fread((void*)buffer,sizeof(int),128,file);
      t+=numread;
      do {
         for(i=0; i<127; i++){
         sdata[i] = buffer[i]; }
         numread = fread((void *)buffer
                         ,sizeof(int),128,file);
         t+=numread;
         if( numread < 128 ){
         for(i=numread; i<128; i++){
            buffer[i]=0;
         }
      }
      window();
      fft(wf-1,2048,1);  /* -1 for inverse fft */
      mel_energy();
      mel_cepstrum();
      s++;
      fwrite((void *)cep_coef
             ,sizeof(double),6,outfile);
      if( s%10 == 0 ){
      printf("          %d\n",s);
   }
/*    for(i=0; i<6; i++){
        printf("cep_coef[%d]=%f\n",i,cep_coef[i]);
   }*/
   }while( numread == 128 );
   fclose(file);
```

```
      fclose(outfile);
      T+=t;
      S+=s;
      printf("%u  data points read from file %s\n"
             ,t,infilename2);
      printf("    %ld global points read so far\n",T);
      printf("%u  cepstral vectors (%u coeffs)
             computed and written to file %s\n"
             ,s,s*6,outfilename);
      printf(" %ld global vectors computed so far\n",S);
      }
   }
   fclose(infile);
}

/*************************************************************
 * This function takes 256 points from sampled data using *
 * Hamming window and puts zeros in remaining positions   *
 * to implement a 2048 point FFT                          *
 *************************************************************/
void window() {
   int i;

   for (i=0; i<256; i++){
      if( i < 128 ){
         wf[i*2]=sdata[i]*(0.54-0.46*cos(2*PI*i/255));
         wf[i*2+1]=0;
      }else{
         wf[i*2]=buffer[i-128]*(0.54-0.46*cos(2*PI*i/255));
         wf[i*2+1]=0;
      }
   }
   for (i=512; i<4096; i++){
      wf[i]=0; }
}

/*************************************************************/
/* This routine implements a radix-2  FFT                    */
/*************************************************************/
#define SWAP(a,b) tempr=(a);(a)=(b);(b)=tempr

void fft(data,nn,isign)
```

```
double data[];
int nn,isign;
{
   int n,mmax,m,j,istep,i;
   double wtemp,wr,wpr,wpi,wi,theta;
   double tempr,tempi;

   n=nn << 1;
   j=1;
   for (i=1;i<n;i+=2) {
      if (j > i) {
         SWAP(data[j],data[i]);
         SWAP(data[j+1],data[i+1]);
      }
      m=n >> 1;
      while (m >= 2 && j > m) {
         j -= m;
         m >>= 1;
      }
      j += m;
   }
   mmax=2;
   while (n > mmax) {
      istep=2*mmax;
      theta=6.28318530717959/(isign*mmax);
      wtemp=sin(0.5*theta);
      wpr = -2.0*wtemp*wtemp;
      wpi=sin(theta);
      wr=1.0;
      wi=0.0;
      for (m=1;m<mmax;m+=2) {
         for (i=m;i<=n;i+=istep) {
            j=i+mmax;
            tempr=wr*data[j]-wi*data[j+1];
            tempi=wr*data[j+1]+wi*data[j];
            data[j]=data[i]-tempr;
            data[j+1]=data[i+1]-tempi;
            data[i] += tempr;
            data[i+1] += tempi;
         }
         wr=(wtemp=wr)*wpr-wi*wpi+wr;
         wi=wi*wpr+wtemp*wpi+wi;
```

```
      }
      mmax=istep;
   }
}

/*************************************************************
 * This routine computes the MEL-frequencies, then computes
 * the critical and energy and put these values in the same
 * array.
 *************************************************************/
void mel_energy(){
   int i;
   int bindex;

   for(i=0; i<1024; i++){
    fftmag[i]=log10(sqrt(pow(wf[i*2],2)+pow(wf[i*2+1],2)));
   }
   for(i=1; i<21; i++){
      bindex = floor( mf[i-1]*0.2048 );
      mel[i]=0;
      while( bindex*4.8828125 < mf[i+1] ){
      if( bindex*4.8828125 >= mf[i-1] && bindex*4.8828125
         < mf[i] ){
         mel[i] += fftmag[bindex]
              *(bindex*4.8828125-mf[i-1])/(mf[i]-mf[i-1]);
      }
      if( bindex*4.8828125 >= mf[i] && bindex*4.8828125
         < mf[i+1] ){
         mel[i] += fftmag[bindex]
             *(mf[i+1]-bindex*4.8828125)/(mf[i+1]-mf[i]);
      }
      bindex++;
      }
   }
}
/*************************************************************
* This routine the computes MEL-based cepstral coefficients
*************************************************************/
void mel_cepstrum()
{
   int i,k;
```

```
   for(i=0; i<CEP_SIZE; i++){
      cep_coef[i]=0.0;
      for(k=1; k<=20; k++){
         cep_coef[i]+=mel[k]*cos(i*(k-0.5)*PI/20);
      }
   }
}
/*************************************************************
 * This routine computes the MEL-frequencies.              *
 *************************************************************/
void mel_freq() {
   int i;

   mf[0]=0;
   mf[21]=5000;
   for(i=1; i<21; i++){
      mf[i]=1000*(pow(10.0,0.03705482*i)-1);
   }
}
```

C Program 3 - *Codebook*

This program produces the binary tree codebook used by *Quantize*. The program requires the Phar Lab Dos extender supplied with the MicroWay C compiler because of the large amount of data involved. The program is run using the following command.

run386 -vm <path> vmmdrv codebook

The -vm command means that the program is running in virtual memory (i.e. the hard disk is being used as RAM) and vmmdrv is the virtual memory driver contained in the directory specifed by <path>.

```
#include <stdio.h>
#include <ctype.h>
#include <string.h>
#include <time.h>
#include <math.h>

#define NUM        40000
      /* maximum number of cepstral vectors            */
#define MSIZE         30
      /* maximum number of characters in input         */
#define MO             6
      /* Number of Cepstral coefficients               */
#define dist_thresh    0.00000001
      /* threshold for separation of old and new means */
#define NLEVELS        7
      /* number of levels in binary codebook           */
#define TWOPOWER     128
      /* 2^NLEVELS                                     */
#define CONVTHRESH   100
      /* max iteration per node                        */



int     quant[NUM];
double  s_vector[NUM][MO];
long    total_count;
long    tot1,oldtot1,tot2,oldtot2;
double  vector1[MO],vector2[MO],old_vector1[MO];
double  old_vector2[MO];
double  vector[NLEVELS+1][TWOPOWER][MO];
FILE    *outfile,*outfile2;

main()
{
char  outfilename[80];
void  initialize();
void  vectorize();

strcpy(outfilename,"cdbkjdif.dat");
outfile = fopen(outfilename,"w");
outfile2 = fopen("cdbkjdif.log","w");
printf("Start initialization...\n");
```

```
initialize();
vectorize(NLEVELS);
fclose(outfile);
fclose(outfile2);
}

/*********************************************************/

void vectorize(nlevels)
int nlevels;
{
int    Level,Level2,nt;
register int i,k;
long   j;
double old_dist1,old_dist2;
double distance(),dist1,dist2;
long   input_data();
void   separate();
void   compute_vector1();
void   compute_vector2();
void   perturb();
printf("Start  vectorize\n");

total_count = input_data();
printf("total_count=%ld\n",total_count);
for(Level=0;Level<nlevels;Level++)
   {
   for(j=0;j<total_count;j++)
      quant[j]*=2;
   Level2=(int)pow((double)2,(double)Level);
   for(i=0;i<Level2;i++)
      {
      printf("\n");
      dist1=dist2=0;
      perturb(vector[Level][i]);
      nt=0;
up:   separate(i);
      nt++;
      old_dist1 = dist1;
      old_dist2 = dist2;
      dist1=distance(vector1,old_vector1);
      dist2=distance(vector2,old_vector2);
```

```
      printf("level=%u  node=%u  nt=%u\n",Level,i,nt);
      printf("    dist1=%g  old_dist1=%g  dif=%g\n",dist1,
            old_dist1,fabs(dist1-old_dist1));
      printf("    dist2=%g  old_dist2=%g  dif=%g\n",dist2,
            old_dist2,fabs(dist2-old_dist2));
      fprintf(outfile2,"lvl=%u nd=%u nt=%u dst1=%g
             dst2=%g\n\015",Level,i,nt,dist1,dist2);
      if( fabs(dist1-old_dist1) >= dist_thresh && nt
         < CONVTHRESH ){goto up;}
      if( fabs(dist2-old_dist2) >= dist_thresh && nt
         < CONVTHRESH ){goto up;}
      for(k=0;k<MO;k++){
         vector[Level+1][2*i][k]=vector1[k];
         vector[Level+1][2*i+1][k]=vector2[k];
      }
   }
}
for(i=1;i<=nlevels;i++)
   {
   fprintf(outfile,"Level  %u .......\n",i);
   for(j=0;j<(int)pow((double)2,(double)i);j++)
      {
      for(k=0;k<MO;k++)
         fprintf(outfile,"% 14.10f ",vector[i][j][k]);
      fprintf(outfile,"\n");
      }
   }
}

/*********************************************************/

void separate(i)
   long i; {
   register int  j,k;
   long    n;
   double  dist1,dist2;
   double  sum1[MO],sum2[MO];
   double  distance();

   tot1=tot2=0;
   for(j=0; j<MO; j++){
      sum1[j]=0;
```

```
      sum2[j]=0;
   }
   for(n=0; n<total_count; n++){
      if(quant[n]==i*2 || quant[n]==i*2+1){
         dist1=distance(vector1,s_vector[n]);
         dist2=distance(vector2,s_vector[n]);
      if( dist1 < dist2 ){
         for(k=0; k<M0; k++)
            sum1[k]+=s_vector[n][k];
         tot1++;
         quant[n]=i*2;
      }else{
         for(k=0; k<M0; k++)
            sum2[k]+=s_vector[n][k];
         tot2++;
         quant[n]=i*2+1;
      }
   }
}
   for(k=0; k<M0; k++){
      old_vector1[k]=vector1[k];
      old_vector2[k]=vector2[k];
   }
   if(tot1!=0)
      for(k=0; k<M0; k++)
         vector1[k]=sum1[k]/tot1;
   if(tot2!=0)
      for(k=0; k<M0; k++)
         vector2[k]=sum2[k]/tot2;
   printf("tot1=%ld   tot2=%ld\n",tot1,tot2);
   fprintf(outfile2,"tot1=%ld   tot2=%ld\n\015",tot1,tot2);
}

/*************************************************************/

double distance(vector_a,vector_b)
   double vector_a[M0],vector_b[M0];{
   int k;
   double  dist;

   dist=0.0;
   for(k=0; k<M0; k++){
```

```
      dist+=pow((vector_a[k]-vector_b[k]),2.0);
   }
   dist=sqrt(dist);
   return(dist);
}

/*************************************************************/
void perturb(vectora)
   double vectora[MO];{
   int j;

   for(j=0; j<MO; j++){
      vector1[j]=vectora[j]*1.01;
      vector2[j]=vectora[j]*0.99;
   }
}

/*************************************************************/
void initialize(){
   long i;

   for(i=0; i<MO; i++){
      vector1[i]=0.0;
      vector2[i]=0.0;
   }
   for(i=0; i<NUM; i++)
      quant[i]=0;
}

/*************************************************************/
long input_data(){
   int  i,h,t;
   long count;
   char filename[80],infilename[80];
   char instring[20],numstring[10];
   FILE *infile,*file;
   int numread;
   double buffer[6];
   void exit();

   count=0;
   strcpy(infilename,"dwrds.dat");
```

```
printf("Cepstral vector input  -
       infilename = %s\n",infilename);
if ( (infile = fopen(infilename, "r")) == NULL){
   printf("fopen failed for
          infilename %s.\n",infilename);
   exit(0);
}
while( fscanf(infile,"%s\n",instring) != EOF){
   for(h=1; h<11; h++){
      strcpy(filename,"c:\\cdata2\\");
      strcat(filename,instring);
      strcat(filename,".j");
      switch(h){
      case 1  : strcpy(numstring,"01");
                break;
      case 2  : strcpy(numstring,"02");
                break;
      case 3  : strcpy(numstring,"03");
                break;
      case 4  : strcpy(numstring,"04");
                break;
      case 5  : strcpy(numstring,"05");
                break;
      case 6  : strcpy(numstring,"06");
                break;
      case 7  : strcpy(numstring,"07");
                break;
      case 8  : strcpy(numstring,"08");
                break;
      case 9  : strcpy(numstring,"09");
                break;
      case 10 : strcpy(numstring,"10");
                break;
      case 11 : strcpy(numstring,"11");
                break;
      case 12 : strcpy(numstring,"12");
                break;
      case 13 : strcpy(numstring,"13");
                break;
      case 14 : strcpy(numstring,"14");
                break;
      case 15 : strcpy(numstring,"15");
```

```
                    break;
            default : break;
        }
        strcat(filename,numstring);
        _pmode = 0x8000;    /* binary mode for NDPC*/
        if ( (file = fopen(filename, "r")) == NULL){
            printf("fopen failed for
                    filename %s\n",filename);
            exit(0);
        }
        printf("reading in data from
                file %s .....\n",filename);
        t=0;
        do{
            numread = fread((void *)buffer,
                            sizeof(double),6,file);
        if(numread == 0)
            break;
        for(i=0; i<6; i++){
            s_vector[count][i] = buffer[i];
            vector1[i] += buffer[i];
        }
        count++;
        t++;
    }while( numread == 6 );
    fclose(file);
    printf("%ld  vectors (%ld coeffs.) read from file
            %s\n",t,t*6,filename);
    printf("      %ld  total vectors (%ld coeffs.)
            read so far\n",count,count*6);
    }
}
    for(i=0; i<M0; i++){
        vector[0][0][i]=vector1[i]/count;
    }
    return count;
}
```

D Program 4 - *Quantize*

This program is used to quantize the mel-cepstrum coefficients into symbols that are then used in program *Hmmtrain*. The program is run using the following command:

Quantize codebook.file inputwords.dat

where codebook.file is the codebook produced by program *Codebook* and inputwords.dat contains the words of the vocabulary being quantized.

```
#include <stdio.h>
#include <ctype.h>
#include <string.h>
#include <time.h>
#include <math.h>
#define MSIZE         50
      /* maximum number of characters in input */
#define MO             6
      /* number of cepstral coefficients       */
#define NLEVELS        7
      /* number of levels in binary codebook   */
#define LEVELINDX    126
      /* 2^NLEVELS-2                           */
#define TOTVECT      254
      /* number of vectors in codebook         */

int count;
double in_vector[MO];
double codebook[TOTVECT][MO];
double buffer[6];
char outfilename[80];
char codefile[80];
char words[MSIZE];
FILE *outfile;

int main(argc,argv)
   int argc;
   char *argv[];{
   int i,j,k;
   int numread;
   char infiles[80],infilename1[80];
   char wrdfile1[80],wrdfile2[80],wrdnum[10];
   FILE *infile1,*infile2;
   int count;
   void codebook_entry();
   void vector_quantize();

   if( argc <= 1 ){
      printf("*****  After program name
             enter two file names  *****\n");
      printf("1.   The first file name is the
             name of the codebook.\n");
```

```
   printf("2.   The second is the
       name of the file that contains the paths and\n");
   printf("     names of all the data files to be
          qauntized.\n\n");
   printf("Example:   ");
   printf("lpc2 codebook.dat allfiles.dat\n",argv[0]);
   exit(0);
}
strcpy(codefile,argv[1]);
strcpy(infiles,argv[2]);

codebook_entry();

for(j=0; j<M0; j++)
   printf("% 10.8f ",codebook[253][j]);
printf("\n");

if ( (infile1 = fopen(infiles, "r")) == NULL){
   printf("fopen failed for file %s.\n",infilename1);
   exit(0);
}
while( fscanf(infile1,"%s\n",words) != EOF){
   for(i=1; i<16; i++){
      strcpy(wrdfile1,"c:\\cdata2\\");
      strcat(wrdfile1,words);
      strcat(wrdfile1,".j");
      itoa(i,wrdnum,10);
      if( i < 10 )
         strcat(wrdfile1,"0");
      strcat(wrdfile1,wrdnum);
      if ( (infile2 = fopen(wrdfile1, "rb")) == NULL){
         printf("fopen failed for file %s.\n",wrdfile1);
         exit(0);
      }
      strcpy(wrdfile2,"c:\\qdata2\\");
      strcat(wrdfile2,words);
      strcat(wrdfile2,".q");
      itoa(i,wrdnum,10);
      if( i < 10 )
         strcat(wrdfile2,"0");
      strcat(wrdfile2,wrdnum);
      if ( (outfile = fopen(wrdfile2, "w")) == NULL){
```

```
            printf("fopen failed for file %s.\n",wrdfile2);
         exit(0);
      }
      count=0;
      printf("reading in data from
             file %s .....\n",wrdfile1);
      do{
         numread = fread((void *)buffer
                         ,sizeof(double),6,infile2);
         if(numread == 0)
            break;
         vector_quantize();
         count++;
      }while( numread == 6 );
      fclose(infile2);
      fclose(outfile);
      printf("%u  data points read from file %s\n"
             ,count,wrdfile1);
      printf("     %u Quantized vectors written to
             file  %s\n",count,wrdfile2);
      }
   }
}

/*************************************************************/
/* This routine vector quantizes the computed lpc vector */
/* with respect to the given codebook.                       */
/*************************************************************/
void vector_quantize()
{
int i,index1,index2,vq;
double idm1,idm2;
double distance();
int level_index();

index1 = 0;
index2 = 1;
idm1 = distance(codebook[index1],buffer);
idm2 = distance(codebook[index2],buffer);
/*printf("index1=%d   index2=%d\n",index1,index2);*/
/*printf("    idm1=%f   idm2=%f\n",idm1,idm2);*/
if( idm1 > idm2 )
```

```
   index1 = index2;

for(i=1; i<NLEVELS; i++)
   {
   index1 = (index1 - level_index(i))*2 + level_index(i+1);
   index2 = index1 + 1;
   idm1 = distance(codebook[index1],buffer);
   idm2 = distance(codebook[index2],buffer);
/*printf("index1=%d   index2=%d\n",index1,index2);*/
/*printf("    idm1=%f   idm2=%f\n",idm1,idm2);*/
   if( idm1 > idm2 )
      index1 = index2;
   }
vq = index1 - LEVELINDX;
fprintf(outfile,"%d\n",vq);
/*printf(" %d\n",vq);*/
}

/*************************************************************/
/*  This routine calculates which vector to compare      */
/*  next in the codebook once a vector index in the      */
/*  previous level is given                              */
/*************************************************************/
int level_index(k)
int k;
{
int num;
num = (int)pow((double)2,(double)k) - 2;
return num;
}
/*************************************************************/

double distance(vector_a,vector_b)
   double vector_a[MO],vector_b[MO];{
   int k;
   double  dist;

   dist=0.0;
   for(k=0; k<MO; k++){
      dist+=pow((vector_a[k]-vector_b[k]),2.0);
   }
   dist=sqrt(dist);
```

```
   return(dist);
}

/*************************************************************/
void codebook_entry(){
   FILE *infile3;
   int i,j,k,m;
   char input1[80],input2[80],input3[80],fp[6][20];

   infile3 = fopen(codefile,"r");

   printf("\nReading %s\n",codefile);
   m=0;
   for(i=1; i<=NLEVELS; i++){
      fscanf(infile3,"%s %s %s\n",input1,input2,input3);
      printf("%s %s %s\n",input1,input2,input3);
      for(j=0; j<(int)pow((double)2,(double)i); j++){
         fscanf(infile3,"%s %s %s %s %s %s\n"
                ,fp[0],fp[1],fp[2],fp[3],fp[4],fp[5]);
         for(k=0; k<6; k++){
            codebook[m][k]=atof(fp[k]);
         }
         m++;
      }
   }
   fclose(infile3);
}
```

E Program 5 - *Hmmtrain*

This program is used to train the hidden Markov models and uses the files containing the symbols produced by program *Quantize.* It is run using the command:

run386 Hmmtrain

Certain files have to be set up before *Hmmtrain* can be used. these are set up using the programs *Makew8 and Makew9. Makew8* is used to set up the paths to the data files and *Makew9* is used to set up the batch file containing the command found above. The listing of these programs are found following *Hmmtrain.*

```
#include <stdio.h>
#include <ctype.h>
#include <string.h>
#include <time.h>
#include <errno.h>
#include <math.h>
#define NUM       5000
    /* total allowable number of vq indices for training */
#define SYM        128
    /* number of symbols in B matrix                     */
#define TNUM       500
    /* number of vq indices in each utterance            */
#define MORDER       6
    /* model order of the A matrix                       */
#define MAXITER    150
    /* maximum number of training iterations             */
#define WNUM        20
    /* number of training files                          */
#define INIT         5
    /* number of initial tries                           */
#define INITLIMIT    5
   /* number of iterations for each try                  */
#define PTH          0.02
#define EPSILON 1.0e-8
#define EPSLN   1.0e-8
  /*threshold for bw[][], used to keep beta from overflow*/

int vq_data[NUM],length[WNUM],ind1[SYM],ind2[SYM];
int M;
int W,W1;
int indx[WNUM+1];
int iteration;
int num_vq_indices;
int scalenum;
double a[MORDER][MORDER],b[MORDER][SYM];
double bw[MORDER][SYM],p[MORDER];
double ia[INIT][MORDER][MORDER],ibw[INIT][MORDER][SYM];
double alpha[WNUM][TNUM][MORDER],beta[WNUM][TNUM][MORDER];
double ct[WNUM][TNUM];
double prb[WNUM],prb_old[WNUM];
double probsum,prob_oldsum;
double initprob[INIT];
```

```
char input2[80];

main(argc,argv)
int argc;
char *argv[];
{
int i,j,k,n;
int prbindx;
double smallest;
int input_vq_indices();
void order_vq_indices();
void initialize_ab();
void alpha_beta_computation();
void reestimate_ab();
void ab_matrix_output();

if( argc <= 2)
   {
   printf("\nCommand line arguments should be two file
          names.\n");
   printf("     1.  The first file name is for a file
          that contains\n");
   printf("         the names and paths of the
          quantized data files.\n");
   printf("     2.  The second file name is the
          name of the HMM to be calculated.\n");
   printf("Example:  hmm128lr qfiles.dat hmm2.dat\n\n");
   exit(0);
   }


num_vq_indices=input_vq_indices(argv[1]);
printf("total vq indices read in is: %d\n",num_vq_indices);
printf("Start function order_vq_indices()\n");
order_vq_indices();
printf("Start function initialize_ab()\n");
initialize_ab();

for(n=0; n<INIT; n++)
   {
   for(i=0; i<MORDER; i++)
      {
```

```
      for(j=0; j<MORDER; j++)
         a[i][j] = ia[n][i][j];
      for(k=0; k<M; k++)
         bw[i][k] = ibw[n][i][k];
      }
   for(i=0; i<WNUM; i++)
      prb[i]=prb_old[i]=0;
   for(k=0; k<INITLIMIT; k++)
      {
      prob_oldsum=probsum;
      for(j=0; j<WNUM; j++)
         prb_old[j]=prb[j];
      printf("\niteration %u of trial = %u\n",k,n);
      alpha_beta_computation();
/*      printf("Start function reestimate_ab();\n");*/
      reestimate_ab();
      printf("old:  ");
      for(j=0; j<WNUM; j++)
         printf("% 6.1f",prb_old[j]);
      printf("\n");
      printf("new:  ");
      for(j=0; j<WNUM; j++)
         printf("% 6.1f",prb[j]);
      printf("\n");
      printf("prob_oldsum =% 12.7f\n",prob_oldsum);
      printf("probsum     =% 12.7f\n",probsum);
      }
   for(i=0; i<MORDER; i++)
      {
      for(j=0; j<MORDER; j++)
         ia[n][i][j] = a[i][j];
      for(k=0; k<M; k++)
         ibw[n][i][k] = bw[i][k];
      }
   initprob[n] = probsum;
   }
for(n=0; n<INIT; n++)
   printf("initprob[%u]=% 12.7f\n",n,initprob[n]);
smallest = initprob[0];
prbindx = 0;
for(n=1; n<INIT; n++)
   if( initprob[n] < smallest )
```

```
      {
      prbindx = n;
      smallest = initprob[n];
      }
   printf("smallest  = % 12.7f\n",smallest);
   printf("Training with trial %u  prob was % 12.7f\n"
          ,prbindx,smallest);
   for(i=0; i<MORDER; i++){
      for(j=0; j<MORDER; j++)
         a[i][j] = ia[prbindx][i][j];
      for(k=0; k<M; k++)
         bw[i][k] = ibw[prbindx][i][k];
   }
printf("Starting training...\n");

iteration = 0;
probsum = smallest;
for(i=0; i<WNUM; i++)
   prb[i]=prb_old[i]=0;
for(i=0; i<MAXITER; i++)
   {
   prob_oldsum=probsum;
   for(j=0; j<WNUM; j++)
      prb_old[j]=prb[j];
   printf("\niteration = %d\n",iteration);
/*   printf("Start function alpha_beta_computation()\n");*/
   alpha_beta_computation();
/*   printf("Start function reestimate_ab();\n");*/
   reestimate_ab();
   printf("old:  ");
   for(j=0; j<WNUM; j++)
      printf("% 6.1f",prb_old[j]);
   printf("\n");
   printf("new:  ");
   for(j=0; j<WNUM; j++)
      printf("% 6.1f",prb[j]);
   printf("\n");
   printf("prob_oldsum =% 12.7f\n",prob_oldsum);
   printf("probsum     =% 12.7f\n",probsum);
   printf("difference  = %g\n",probsum-prob_oldsum);
   if( fabs(probsum - prob_oldsum) < PTH && iteration > 15)
      break;
```

```
   iteration++;
   }
printf("threshold reached.....\n");
ab_matrix_output(argv[2]);
}

/*********************************************************
This function inputs the training lpc quantized vectors *
from their respective files corresponding to each       *
training utterance and puts them in an array vq_data[]  *
*********************************************************/
int input_vq_indices(infilename)
char infilename[];
{
char input3[15];
FILE *infile1,*infile2;
int j,k,t;
int symbnum[WNUM];

W1=t=indx[0]=0;
if ( (infile1 = fopen(infilename, "r")) == NULL)
   {
   printf("fopen failed for file %s.\n",infilename);
   exit(0);
   }
while( fscanf(infile1,"%s\n",input2) != EOF)
   {
   if ( (infile2 = fopen(input2, "r")) == NULL)
      {
      printf("fopen failed for file %s.\n",input2);
      exit(0);
      }
   printf("reading in data from file %s .....\n",input2);
   W1++;
   k=0;
   while( fgets(input3,15,infile2) != NULL)
      {
      vq_data[t] = atoi(input3);
      t++;
      k++;
      }
   indx[W1]=t;
```

```
   fclose(infile2);
   symbnum[W1-1]=k;
   printf("    %u symbols read in\n",k);
   printf("    indx[%u]=%u\n",W1,indx[W1]);
   }
fclose(infile1);
scalenum=symbnum[0];
for(j=1; j<W1; j++)
   if( symbnum[j] <= scalenum )
      scalenum = symbnum[j];
printf("The smallest file contains %u
       lpc vectors\n",scalenum);
W=W1;
return t;
}

/*************************************************************/
/*  This function outputs the evaluated A and B matrices */
/*************************************************************/
void ab_matrix_output(outfilename)
char outfilename[];
{
int i,j,k,l;
double b[MORDER][SYM];
double bsum;
FILE *outfile;

printf("writing hmm to file %s\n",outfilename);
outfile = fopen(outfilename,"w");

for(i=0; i<MORDER; i++)
   {                    /* A is being transposed to file */
   for(j=0; j<MORDER; j++)
      fprintf(outfile,"%10.8f  ",a[j][i]);
   fprintf(outfile,"\n");
   }
fprintf(outfile,"\n");

for(j=0; j<MORDER; j++)
   for(k=0; k<SYM; k++)
      b[j][k] = 0.0;
```

```
for(k=0; k<M; k++)
   for(j=0; j<MORDER; j++)
      b[j][ind1[k]] = bw[j][k];

for(j=0; j<MORDER; j++)
   {
   l=0;
   bsum = 0.0;
   for(k=0; k<SYM; k++)
      {
      if( b[j][k] < EPSILON )
         l = l + 1;
      else
         bsum = bsum + b[j][k];
      }
   for(k=0; k<SYM; k++)
      {
      if( b[j][k] < EPSILON )
         b[j][k] = EPSILON;
      else
         b[j][k] = (1.0 - (double)l*EPSILON)*b[j][k]/bsum;
      }
   }

for(i=0; i<SYM; i++)
   {                    /* B is being transposed to file */
   for(j=0; j<MORDER; j++)
      fprintf(outfile,"%10.8f  ",b[j][i]);
   fprintf(outfile,"\n");
   }

fclose(outfile);
}

/*************************************************************/
/*  This function reestimates the A and B matrices      */
/*  using the Baum - Welch reestimation formulas.       */
/*************************************************************/
void reestimate_ab()
{
int i,j,k,l,t,s;
int iw;
```

```
double bsum;
double xi[WNUM][MORDER][MORDER];
double bt[WNUM][MORDER][SYM];
double gamma[WNUM][MORDER];
double temp1,temp2;
double a2[MORDER],b2[MORDER],tempsum;
double end;

for(iw=0; iw<W; iw++)
   for(i=0; i<MORDER; i++)
      {
      gamma[iw][i] = 0.0;
      for(j=0; j<MORDER; j++)
         xi[iw][i][j] = 0.0;
      for(j=0; j<M; j++)
         bt[iw][i][j] = 0.0;
      }
for(iw=0; iw<W; iw++){
   for(i=0; i<MORDER; i++){
      for(j=0; j<MORDER; j++){
         t=0;
         for(k=indx[iw]; k<indx[iw+1]-1; k++){
            xi[iw][i][j] += alpha[iw][t][i]*a[i][j]
              *bw[j][ind2[vq_data[k+1]]]*beta[iw][t+1][j];
            t++;
         }
      }
      t=0;
      for(k=indx[iw]; k<indx[iw+1]-1; k++){
         gamma[iw][i] += alpha[iw][t][i]
            *beta[iw][t][i]/ct[iw][t];
         t++;
      }
   }
}

/*for(iw=0; iw<W; iw++)
   for(i=0; i<MORDER; i++)
      {
      tempsum=0;
      for(j=0; j<MORDER; j++)
         tempsum += xi[iw][i][j];
```

```
      printf("gamma[%u][%u]=%g summed over j
         xi[%u][%u]=%g\n",iw,i,gamma[iw][i],iw,i,tempsum);
      }*/


/*printf("Begin A calculations\n");*/
for(i=0; i<MORDER; i++)
   {
   a2[i]=0;
   for(j=0; j<MORDER; j++)
      {
      temp1=temp2=0;
      for(iw=0; iw<W; iw++){
         temp1 += xi[iw][i][j];
         temp2 += gamma[iw][i];
      }
      a[i][j] = temp1/temp2;
      a2[i] += a[i][j];
      printf("%6.5f ",a[i][j]);
      }
   printf("\n");
   }
for(i=0; i<MORDER; i++)
   printf("a2[%u]=%g  ",i,a2[i]);
printf("\n");
for(i=0; i<MORDER; i++)
   for(j=0; j<MORDER; j++)
      a[i][j] = a[i][j]/a2[i];

for(iw=0; iw<W; iw++)
   {
   end = indx[iw+1]-indx[iw]-1;
   for(j=0; j<MORDER; j++)
      {
      gamma[iw][j] += alpha[iw][end][j]
         *beta[iw][end][j]/ct[iw][end];
      }
   for(j=0; j<MORDER; j++)
      for(k=0; k<M; k++){
         t=0;
         for(l=indx[iw]; l<indx[iw+1]; l++){
            if( ind1[k] == vq_data[l] ){
```

```
                bt[iw][j][k] += alpha[iw][t][j]
                    *beta[iw][t][j]/ct[iw][t];
            }
            t++;
         }
      }
   }
/*printf("Begin B calculations\n");*/
for(j=0; j<MORDER; j++)
   {
   b2[j]=0;
   for(k=0; k<M; k++)
      {
      temp1=temp2=0;
      for(iw=0; iw<W; iw++){
         temp1 += bt[iw][j][k];
         temp2 += gamma[iw][j];
      }
      bw[j][k] = temp1/temp2;
      b2[j] += bw[j][k];
      }
   printf("b2[%u]=%g  ",j,b2[j]);
   }
printf("\n");

printf("%s\n",input2);

for(j=0; j<MORDER; j++)
   for(k=0; k<M; k++)
      bw[j][k] = bw[j][k]/b2[j];

for(j=0; j<MORDER; j++)
   {
   l=0;
   bsum = 0.0;
   for(k=0; k<M; k++)
      {
      if( bw[j][k] < EPSLN )
         l = l + 1;
      else
         bsum = bsum + bw[j][k];
      }
```

```
   for(k=0; k<SYM; k++)
      {
      if( bw[j][k] < EPSLN )
         bw[j][k] = EPSLN;
      else
         bw[j][k] = (1.0 - (double)1*EPSLN)*bw[j][k]/bsum;
      }
   }
}

/*********************************************************/
/*  This function calculates alpha and beta of the   */
/*  forward-backward procedure and does it for       */
/*  each training utterance.                         */
/*********************************************************/
void alpha_beta_computation()
{
int i,j,k,m,t,t2;
int iw;
double sumalpha;
for(iw=0; iw<W; iw++)
   {
   for(j=0; j<MORDER; j++)            * Computing alpha  */
      alpha[iw][0][j]= p[j]*bw[j][ind2[vq_data[indx[iw]]]];
   sumalpha = 0;
   for(j=0; j<MORDER; j++)
      sumalpha += alpha[iw][0][j];
   if( sumalpha == 0 )
      {
/*      printf("sumalph = 0  iw=%u t=0\n",iw);*/
      ct[iw][0]=1;
      }
   else
      ct[iw][0] = 1/sumalpha;
   for(j=0; j<MORDER; j++)
      {
      alpha[iw][0][j] *= ct[iw][0];
      }

   t = 0;
   for(k=indx[iw]; k<indx[iw+1]-1; k++)
      {
```

```
      for(j=0; j<MORDER; j++){
         alpha[iw][t+1][j] = 0.0;
      for(i=0; i<MORDER; i++)
         alpha[iw][t+1][j] +=  alpha[iw][t][i]*a[i][j];
         alpha[iw][t+1][j] *= bw[j][ind2[vq_data[k+1]]];
      }
      sumalpha = 0;
      for(j=0; j<MORDER; j++)
         sumalpha += alpha[iw][t+1][j];
      if( sumalpha == 0 ){
/*       printf("sumalpha = 0  iw=%u t+1=%u\n",iw,t+1);*/
         ct[iw][t+1] = 1;
      }
      else
         ct[iw][t+1] = 1/sumalpha;
      for(j=0; j<MORDER; j++) {
         alpha[iw][t+1][j] *= ct[iw][t+1];
      }
      t++;
      }
   t2=t;
   for(i=0; i<MORDER-1; i++)
      beta[iw][indx[iw+1]-1][i] = 0.0;
   beta[iw][indx[iw+1]-indx[iw]-1][MORDER-1] = 1.0;
   beta[iw][indx[iw+1]-indx[iw]-1][MORDER-1] *= ct[iw][t2];
   t2--;
   t=indx[iw+1]-indx[iw]-2;
   for(k=indx[iw+1]-2; k>=indx[iw]; k--)
      {
      for(i=0; i<MORDER; i++){
         beta[iw][t][i] = 0.0;
         for(j=0; j<MORDER; j++){
            beta[iw][t][i] += a[i][j]
               *bw[j][ind2[vq_data[k+1]]]*beta[iw][t+1][j];
         }
      }
      for(i=0; i<MORDER; i++){
         beta[iw][t][i] *= ct[iw][t2];
      }
      t--;
      t2--;
      }
```

```
   }
for(iw=0; iw<W; iw++)
   prb[iw]=0;
probsum=0;
for(iw=0; iw<W; iw++)
   {
   for(j=0; j<indx[iw+1]-indx[iw]; j++)
      prb[iw] += log10(ct[iw][j]);
   probsum += prb[iw];
   }
}

/*********************************************************/
/*   This function finds and orders the vq indices that  */
/*   occur in the training phase so that only these vq   */
/*   indices get any type of probability of occuring,    */
/*   so when we randomize the B matrix only the vq       */
/*   indices that have occured received random values.   */
/*   The others are set to value EPSILON later           */
/*   in the program                                      */
/*********************************************************/
void order_vq_indices()
{
int f,i,j,k;

M=1;
ind1[0]=vq_data[0];
for(i=1; i<num_vq_indices; i++)
   {
   f=0;
   for(j=0; j<M; j++)
      {
      if( vq_data[i] < ind1[j] ){
         for(k=(M-1); k>=j; k--)
            ind1[k+1]=ind1[k];
         ind1[j] = vq_data[i];
         M=M+1;
         f=1;
         break;
      }
      if( vq_data[i] == ind1[j] ){
         f=1;
```

```
            break;
         }
      }
      if( f == 0 )
         {
         ind1[M]=vq_data[i];
         M=M+1;
         }
      }
printf("distinct vq indices: M=%d\n",M);
for(i=0; i<M; i++)
   ind2[ind1[i]]=i;
   /* this "compresses" the indices from 0 to M */
   /* instead of using the symbols as indexes   */
}


/*****************************************************/
/*   This function sets the values of the A and B */
/*   matrices to initial random values and then   */
/*   normalizes the rows of A and B               */
/*****************************************************/
void initialize_ab()
{
int i,j,k,n;
double suma,sumb,sump;

for(n=0; n<INIT; n++)
   for(i=0; i<MORDER; i++)
      /* Randomizing the A and B matrices */
      {
      for(j=0; j<MORDER; j++){
         if( j>=i && j<i+3 )
            ia[n][i][j]=(double)rand();
         else
            ia[n][i][j]=0;
         }
         for(k=0; k<M; k++)
            ibw[n][i][k]=(double)rand();
      }
p[0]=1;
for(i=1;i<MORDER;i++) /* state 0 to 1 since for this hmm */
```

```
   p[i]=0;            /* model we only start at state 0  */
sump = 0.0;
for(i=0;i<MORDER;i++) /*Normalizing p (not really needed */
   sump = sump + p[i];  /* but included for generality)  */
for(i=0; i<MORDER; i++)
   p[i]=p[i]/sump;
for(n=0; n<INIT; n++)
   for(i=0; i<MORDER; i++)      /* Normalizing the rows  */
      {                         /* of A and B            */
      suma=sumb=0;
      for(j=0; j<MORDER; j++)
         suma = suma + ia[n][i][j];
      for(k=0; k<M; k++)
         sumb = sumb + ibw[n][i][k];
      for(j=0; j<MORDER; j++)
         ia[n][i][j] = ia[n][i][j]/suma;
      for(k=0; k<M; k++)
         ibw[n][i][k] = ibw[n][i][k]/sumb;
      }
/*for(i=0; i<MORDER; i++)
   {
   for(k=0; k<M; k++)
      printf("%g  ",bw[i][k]);
   printf("\n");
   }*/
}
```

Listing of program *Makew8*

```
#include <stdio.h>
#include <string.h>
#include <math.h>
main(argc,argv)
int argc;
char *argv[];
{
int k,m,s;
FILE *outfile,*infile;
char infilename[80];
char numstring[5];
char prefix[30],prefix1[30];
char words[192][30];

strcpy(infilename,argv[1]);
printf("infilename=%s\n",infilename);
if ( (infile = fopen(infilename, "r")) == NULL)
   {
   printf("fopen failed for file %s.\n",infilename);
   exit(0);
   }
while( fscanf(infile,"%s\n",prefix1) != EOF)
   {
   strcpy(prefix,"c:\\fdata\\");
   strcat(prefix,prefix1);
   strcat(prefix,".dat");
   outfile = fopen(prefix,"w");
   printf("%s\n",prefix);
   for( m=1; m<11; m++)
      {
      strcpy(prefix,"c:\\qdata\\");
      strcat(prefix,prefix1);
      strcat(prefix,".q");
      itoa(m,numstring,10);
      if( m < 10 )
         strcat(prefix,"0");
      strcat(prefix,numstring);
      fprintf(outfile,"%s\n",prefix);
      strcpy(prefix,"c:\\qdatars\\");
      strcat(prefix,prefix1);
      strcat(prefix,".q");
```

```
      if( m < 10 )
         strcat(prefix,"0");
      strcat(prefix,numstring);
      fprintf(outfile,"%s\n",prefix);
      }
   fclose(outfile);
   }
}
```

Listing of program *Makew9*

```
#include <stdio.h>
#include <string.h>
#include <math.h>
main(argc,argv)
int argc;
char *argv[];
{
int i,k,m,n,t1,t2,s;
FILE *outfile,*infile;
char outfilename[80],infilename[80],conv1[1],conv2[2],input[40];
char number[10],prefix2[45],prefix3[45];
char prefix[75],prefix1[30];
char filename1[20],filename2[20],filename3[20];


strcpy(infilename,argv[1]);
strcpy(outfilename,argv[2]);

if ( (infile = fopen(infilename, "r")) == NULL)
   {
   printf("fopen failed for file %s.\n",infilename);
   exit(0);
   }
outfile = fopen(outfilename,"w");
s=0;
   while( fscanf(infile,"%s\n",prefix1) != EOF){
      strcpy(prefix,"run386 hmm128lr c:\\fdata\\");
      strcat(prefix,prefix1);
      strcat(prefix,".dat  ");
      strcat(prefix," c:\\hmmcomb\\");
      strcat(prefix,prefix1);
      strcat(prefix,".hmm");
      fprintf(outfile,"%s\n",prefix);
   }
fclose(infile);
fclose(outfile);
}
```

F Program 6 - *".m files"*

These ".m files" are used in conjuction with the software program *Matlab* (Math Works Inc.) to diagonalize the HMM's. The ".m file" *trnsfhmm.m* does the actual transformation. The ".m file" *autotrns.m* reads in the HMM's, uses the ".m file" *trnsfhmm.m* to do the transformation, and then saves the transformed HMM. The listing of *autotrns.m* gives an example for transforming the HMM's (a, about, all, & an).

Listing of ".m file" *trnsfhmm.m*

```
function [a1,a2,c2]=trnsfhmm(a,c);
[q,a2]=eig(a);
a1=eig(a);
n=[1;zeros(5,1)];
b=a*n;
b1=inv(q)*b;
c1=c*q;
d=eye(6);
for i=1:6
   d(i,i)=b1(i);
end
c2=c1*d;
```

Listing of ".m file" *autotrns.m*

```
load c:\hmmcomb\a.hmm
   [a1,a2,c2]=trnsfhmm(a(1:6,:),a(7:134,:));
   hmm=[a2;c2];
   eigval=[eigval;a1];
   save c:\hmmcomb\a.hmt hmm /ascii /double
   clear a a1 a2 c2 i hmm
load c:\hmmcomb\about.hmm
   [a1,a2,c2]=trnsfhmm(about(1:6,:),about(7:134,:));
   hmm=[a2;c2];
   eigval=[eigval;a1];
   save c:\hmmcomb\about.hmt hmm /ascii /double
   clear about a1 a2 c2 i hmm
load c:\hmmcomb\all.hmm
   [a1,a2,c2]=trnsfhmm(all(1:6,:),all(7:134,:));
   hmm=[a2;c2];
   eigval=[eigval;a1];
   save c:\hmmcomb\all.hmt hmm /ascii /double
   clear all a1 a2 c2 i hmm
load c:\hmmcomb\an.hmm
   [a1,a2,c2]=trnsfhmm(an(1:6,:),an(7:134,:));
   hmm=[a2;c2];
   eigval=[eigval;a1];
   save c:\hmmcomb\an.hmt hmm /ascii /double
   clear an a1 a2 c2 i hmm
save eigvalcm.dat eigval /ascii /double
```

G Program 7 - *Evalhmms*

This program is used in the Recognition experiments to evaluate the likelihoods of the HMMs.

```
#include <stdio.h>
#include <ctype.h>
#include <string.h>
#include <time.h>
#include <errno.h>
#include <math.h>
#define SYM         128
   /* number of symbols in B matrix            */
#define TNUM        200
   /* number of vq indices in each utterance   */
#define MORDER        6
   /* model order of the A matrix              */
#define WNUM         79



int data[TNUM];
double hmma[WNUM][MORDER][MORDER],hmmb[WNUM][MORDER][SYM];
char words[WNUM][10];
int W,mt;

int main(argc,argv)
   int argc;
   char *argv[];{
   char infilename[80];
   char outfilename[80];
   char input3[15];
   char prefix[80],prefix9[80];
   FILE *outfile,*infile2;
   int i,j,k,hmm,p,n,n1,n2,t,tot;
   double prob[WNUM];
   double computation();
   double greatest[8];
   int great_index[8];
   void input_vq_symbols();
   void input_hmms();
   void exit();
   int atoi();


   strcpy(infilename,argv[1]);
```

```
strcpy(outfilename,argv[2]);

input_hmms(infilename);
for(j=0; j<W; j++)
   printf("%s\n",words[j]);

p=tot=0;
outfile=fopen(outfilename,"w");
for(n=0; n<W; n++){
   for(i=11; i<16; i++){
      switch(i){
         case 11 : strcpy(prefix9,"11"); break;
         case 12 : strcpy(prefix9,"12"); break;
         case 13 : strcpy(prefix9,"13"); break;
         case 14 : strcpy(prefix9,"14"); break;
         case 15 : strcpy(prefix9,"15"); break;
         default : break;
      }
      strcpy(prefix,"c:\\qdata\\");
      strcat(prefix,words[n]);
      strcat(prefix,".q");
      strcat(prefix,prefix9);
      if ( (infile2 = fopen(prefix, "r")) == NULL){
         printf("fopen failed for file %s.\n",prefix);
         exit(0);
      }
      t=0;
      while( fgets(input3,15,infile2) != NULL){
         data[t] = atoi(input3);
         t++;
      }
      mt=t;
      fclose(infile2);
      for(hmm=0; hmm<W; hmm++){
         prob[hmm] = computation(hmm);
      }
      for(n1=0; n1<8; n1++){
         greatest[n1] = prob[n1];
         great_index[n1] = n1;
      }
      for(k=1; k<W; k++){
         for(n1=0; n1<8; n1++){
```

```
            if( greatest[n1] < prob[k] ){
               for(n2=8; n2>n1; n2--){
                  greatest[n2] = greatest[n2-1];
                  great_index[n2] = great_index[n2-1];
               }
               greatest[n1] = prob[k];
               great_index[n1] = k;
               break;
            }
         }
      }
      if( n == great_index[0] ){
         printf("%8s - %2u  --->  %8s\n"
                ,words[n],i-11,words[great_index[0]]);
         fprintf(outfile,"%8s - %2u  --->  %8s\n"
                 ,words[n],i-11,words[great_index[0]]);
         p++;
      }else{
         printf("%8s - %2u  --->  %8s MISCLASSIFIED\n"
                ,words[n],i-11,words[great_index[0]]);
         fprintf(outfile,"%8s - %2u  --->  %8s
                 MISCLASSIFIED\n",words[n],i-11
                 ,words[great_index[0]]);
         for(n1=0; n1<8; n1++){
            printf("      prob[%8s]=%f\n"
                 ,words[great_index[n1]],greatest[n1]);
            fprintf(outfile,"prob[%8s]=%f\n"
                 ,words[great_index[n1]],greatest[n1]);
         }
      }
      tot++;
   }
}
printf("recognition rate = % 5.2f percent\n"
       ,(double)p/(double)tot*100);
fprintf(outfile,"recognition rate = % 5.2f percent\n"
        ,(double)p/(double)tot*100);
fclose(outfile);
}
/*************************************************************
This function inputs the training lpc quantized vectors   *
from their respective files corresponding to each         *
```

```
training utterance and puts them in an array data[]       *
/*********************************************************/
void input_hmms(infilename)
   char infilename[];{
   char input2[80];
   char prefix[80],in[6][40];
   FILE *infile1,*infile2;
   int i,j,n;
   double extractword();

/*   strcpy(infilename,"ti20.dat"); */
   if ( (infile1 = fopen(infilename, "r")) == NULL){
      printf("fopen failed for file %s.\n",infilename);
      exit(0);
   }
   n=0;
   while( fscanf(infile1,"%s\n",input2) != EOF){
      strcpy(words[n],input2);
      strcpy(prefix,"c:\\hmmjd\\");
      strcat(prefix,input2);
      strcat(prefix,".hmt");
      if ( (infile2 = fopen(prefix, "r")) == NULL){
         printf("fopen failed for file %s.\n",input2);
         exit(0);
      }
      printf("reading data from file %s\n",prefix);
      for(i=0; i<MORDER; i++){
         fscanf(infile2,"%s %s %s %s %s %s\n"
                ,in[0],in[1],in[2],in[3],in[4],in[5]);
         for(j=0; j<MORDER; j++){
            hmma[n][j][i] = atof(in[j]);
         }
      }
      for(i=0; i<SYM; i++){
         fscanf(infile2,"%s %s %s %s %s %s\n"
                ,in[0],in[1],in[2],in[3],in[4],in[5]);
      for(j=0; j<MORDER; j++)
         hmmb[n][j][i] = atof(in[j]);
      }
      fclose(infile2);
      n++;
      if( n == W ){
```

```
            fclose(infile1);
            break;
         }
      }
      fclose(infile1);
      W=n;
}

/*************************************************************/
double computation(h1)
   int h1;{
   int j,t;
   double ct,sum;
   double x[MORDER];

   for(j=0; j<MORDER; j++)
      x[j]=1;
   sum = 0;
   ct=0;
   for(t=0; t<mt; t++){
      for(j=0; j<MORDER; j++){
         x[j] *= hmma[h1][j][j];
      }
      ct=0;
      for(j=0; j<MORDER; j++){
         ct += hmmb[h1][j][data[t]]*x[j];
      }
      sum += log10(ct);
   }
   return sum;
}
```

H Program 8 - *Compress*

This program is used in combining the eigenvalues to implement the compression techniques.

```
#include <stdio.h>
#include <ctype.h>
#include <string.h>
#include <time.h>
#include <errno.h>
#include <math.h>
#define SYM         128
/* number of symbols in B matrix          */
#define TNUM        100
/* number of vq indices in each utterance  */
#define MORDER        6
/* model order of the A matrix            */
#define WNUM         79



int ldata[WNUM][10];
int qdata[WNUM][10][TNUM];
double hmma[WNUM][MORDER],hmmb[WNUM][MORDER][SYM];
double eigsum[WNUM*MORDER],eigval[WNUM*MORDER];
int  eigindx[WNUM*MORDER],eigcnt[WNUM*MORDER];
int eigvalndx[WNUM*MORDER];
char words[WNUM][10];
char infilename[80];
int W,mt,T;

int main(){
   FILE *outfile;
   int i,j,c;
   double rate;
   double computation();
   void input_vq_symbols();
   void input_hmms();
   void input_qdata();
   void qs();
   void exit();
   void quantize();
   void initialize();
   int atoi();
   double recognition();
```

```
   strcpy(infilename,"dwrdsnu.dat");
   outfile=fopen("outcm1bu.log","w");

   input_hmms();
   input_qdata();
   initialize();
   for(j=0; j<W; j++)
      printf("%s\n",words[j]);
   qs(eigsum,eigindx,0,T-1);
   qs(eigval,eigvalndx,0,T-1);
   rate=recognition();
   printf("%3d  rate=%f\n",T,rate);
   fprintf(outfile,"%3d  %f\n",T,rate);
   c=0;
   for(i=0; i<T-1; i++){
      if( eigsum[i] == eigsum[i+1] ){
         c++;
      }
   }
   for(i=0; i<c; i++){
      printf("%3d  rate=%f\n",T-i-1,rate);
      fprintf(outfile,"%3d  %f\n",T-i-1,rate);
   }
   quantize();
   while( c < T-1 ){
      quantize();
      quantize();
      quantize();
      rate=recognition();
      c=0;
      for(i=0; i<T-1; i++){
         if( eigsum[i] == eigsum[i+1] ){
            c++;
         }
      }
      printf("%3d  rate=%f\n",T-c,rate);
      fprintf(outfile,"%3d  %f\n",T-c,rate);
   }
}

/*********************************************************/
```

```
double recognition() {
   int i,k,hmm,p,n,tot;
   double rate;
   double prob[WNUM];
   double computation();
   double greatest;
   int great_index;
   void input_vq_symbols();
   void input_hmms();
   void qs();
   void exit();
   int atoi();
   p=tot=0;
   for(n=0; n<W; n++){
      for(i=0; i<5; i++){
         for(hmm=0; hmm<W; hmm++){
            prob[hmm] = computation(hmm,n,i);
         }
         greatest = prob[0];
         great_index=0;
         for(k=1; k<W; k++){
            if( greatest < prob[k] ){
               greatest = prob[k];
               great_index=k;
            }
         }
         if( n == great_index ){
            p++;
         }
         tot++;
      }
   }
   rate=(double)p/(double)tot*100;
/*   printf("recognition rate = % 5.2f percent\n",rate);*/
   return rate;
}
/*************************************************************/
void input_qdata(){
   char input3[15];
   char prefix[80],prefix9[10];
   FILE *infile2;
   int i,k,n,t;
```

```
int atoi();
for(n=0; n<W; n++){
   k=0;
   for(i=11; i<16; i++){
      switch(i){
         case 11 : strcpy(prefix9,"11"); break;
         case 12 : strcpy(prefix9,"12"); break;
         case 13 : strcpy(prefix9,"13"); break;
         case 14 : strcpy(prefix9,"14"); break;
         case 15 : strcpy(prefix9,"15"); break;
         default : break;
      }
      strcpy(prefix,"c:\\qdata\\");
      strcat(prefix,words[n]);
      strcat(prefix,".q");
      strcat(prefix,prefix9);
      if ( (infile2 = fopen(prefix, "r")) == NULL){
         printf("fopen failed for file %s.\n",prefix);
         exit(0);
      }
      printf("reading in data from
             file %s .....\n",prefix);
      t=0;
      while( fgets(input3,15,infile2) != NULL){
         qdata[n][k][t] = atoi(input3);
         t++;
      }
      ldata[n][k]=t;
      fclose(infile2);
      k++;
      strcpy(prefix,"c:\\qdatars\\");
      strcat(prefix,words[n]);
      strcat(prefix,".q");
      strcat(prefix,prefix9);
      if ( (infile2 = fopen(prefix, "r")) == NULL){
         printf("fopen failed for file %s.\n",prefix);
         exit(0);
      }
      printf("reading in data from
             file %s .....\n",prefix);
      t=0;
      while( fgets(input3,15,infile2) != NULL){
```

```
            qdata[n][k][t] = atoi(input3);
            t++;
         }
         ldata[n][k]=t;
         fclose(infile2);
         k++;
      }
   }
}

/*************************************************************/
void quantize(){
   int i,j,n,indx;
   double tempsum,dist,prvdist;
   int tempcnt;

   dist=0;
   for(i=0; i<T-1; i++){
      prvdist = dist;
      dist = eigsum[i+1]/eigcnt[i+1] - eigsum[i]/eigcnt[i];
      if( dist != 0 && prvdist == 0){
         tempsum = eigsum[i+1] + eigsum[i];
         tempcnt = eigcnt[i+1] + eigcnt[i];
         indx=i;
         break;
      }
   }
   for( i=0; i<indx+2; i++){
      eigsum[i]=tempsum;
      eigcnt[i]=tempcnt;
   }
   for(i=0; i<T; i++){
      n=(int)floor((double)eigindx[i]/6.0);
      j=eigindx[i]-n*6;
      hmma[n][j]=eigsum[i]/eigcnt[i];
   }
}

/*************************************************************/
/* Alternate quantization procedure

void quantize(){
```

```
    int i,j,n,indx;
    double tempsum,dist,prvdist;
    int tempcnt;

    dist=0;
    for(i=0; i<T-1; i++){
       prvdist = dist;
       dist = eigsum[i+1]/eigcnt[i+1] - eigsum[i]/eigcnt[i];
       if( dist != 0 && prvdist == 0){
          tempsum = eigsum[i+1] + eigsum[i];
          tempcnt = eigcnt[i+1] + eigcnt[i];
          indx=i;
          break;
       }
    }
    for( i=0; i<indx+2; i++){
       eigsum[i]=tempsum;
       eigcnt[i]=tempcnt;
    }
    for(i=0; i<T; i++){
       n=(int)floor((double)eigindx[i]/6.0);
       j=eigindx[i]-n*6;
       hmma[n][j]=eigsum[i]/eigcnt[i];
    }
}
*/
/*************************************************************/
void qs(item,itemindx,left,right)
double item[MORDER*WNUM];
int itemindx[MORDER*WNUM];   /* added */
int left,right;
{
register int i,j;
double x,y;
int h;   /* added */
i=left;
j=right;
x=item[(left+right)/2];
do {
   while(item[i]<x && i<right) i++;
   while(x<item[j] && j>left) j--;
   if(i<=j){
```

```
      y=item[i];
      item[i]=item[j];
      item[j]=y;
      h=itemindx[i];
      itemindx[i]=itemindx[j];
      itemindx[j]=h;
      i++;
      j--;
    }
 } while(i<=j);
if(left<j) qs(item,itemindx,left,j);
if(i<right) qs(item,itemindx,i,right);
}
/*************************************************************/
void initialize(){
    int i;
    for(i=0; i<WNUM*MORDER; i++){
       eigcnt[i] = 1;
    }
}



/*************************************************************
This function inputs the training lpc quantized vectors  *
from their respective files corresponding to each        *
training utterance and puts them in an array data[]      *
*************************************************************/
void input_hmms(){
    char input2[80];
    char prefix[80],in[6][40];
    FILE *infile1,*infile2;
    int i,j,n,s;
    double extractword();

    if ( (infile1 = fopen(infilename, "r")) == NULL){
       printf("fopen failed for file %s.\n",infilename);
       exit(0);
    }
    n=s=0;
    while( fscanf(infile1,"%s\n",input2) != EOF){
       strcpy(words[n],input2);
```

```
      strcpy(prefix,"c:\\hmmcomb\\");
      strcat(prefix,input2);
      strcat(prefix,".hmt");
      if ( (infile2 = fopen(prefix, "r")) == NULL){
         printf("fopen failed for file %s.\n",input2);
         exit(0);
      }
      printf("reading data from file %s\n",prefix);
      for(i=0; i<MORDER; i++){
         fscanf(infile2,"%s %s %s %s %s %s\n",in[0]
                ,in[1],in[2],in[3],in[4],in[5]);
         printf("in[%d]=%s  atof(in[%d])=%f  n=%d  "
                ,i,in[i],i,atof(in[i]),n);
         hmma[n][i] = atof(in[i]);
         printf("hmma[%d][%d]=%f\n",n,i,hmma[n][i]);
         eigval[s]=(pow(10.0,hmma[n][i])-1)/9;
         eigvalndx[s]=s;
         eigsum[s]=hmma[n][i];
         eigindx[s]=s;
         s++;
      }
      for(i=0; i<SYM; i++){
         fscanf(infile2,"%s %s %s %s %s %s\n",in[0]
                ,in[1],in[2],in[3],in[4],in[5]);
         for(j=0; j<MORDER; j++)
            hmmb[n][j][i] = atof(in[j]);
      }
      fclose(infile2);
      n++;
      if( n == W ){
         fclose(infile1);
         break;
      }
   }
   fclose(infile1);
   W=n;
   T=s;
}

/*********************************************************/
double computation(h1,n,i)
   int h1,n,i;{
```

```
    int j,t;
    double ct,sum;
    double x[MORDER];

    for(j=0; j<MORDER; j++)
       x[j]=1;
    sum = 0;
    ct=0;
    for(t=0; t<ldata[n][i]; t++){
       for(j=0; j<MORDER; j++){
          x[j] *= hmma[h1][j];
       }
       ct=0;
       for(j=0; j<MORDER; j++){
          ct += hmmb[h1][j][qdata[n][i][t]]*x[j];
       }
       sum += log10(ct);
    }
    return sum;
}
```