A speech recognition-based telephone auto-attendant

by

GFvB van Leeuwen

Submitted in partial fulfilment of the requirements for the degree
Master of Engineering (Computer Engineering)
in the Faculty of Engineering
UNIVERSITY OF PRETORIA

September 2001

Summary

Keywords: speaker independent speech recognition, telephone auto attendant, dialogue, grammar, whole word hidden Markov modelling, Mel frequency cepstral coefficients, Viterbi search, level building algorithm, noise compensation

This dissertation details the implementation of a real-time, speaker-independent telephone auto attendant from first principles on limited quality speech data. An auto attendant is a computerized agent that answers the phone and switches the caller through to the desired person's extension after conducting a limited dialogue to determine the wishes of the caller, through the use of speech recognition technology.

The platform is a computer with a telephone interface card. The speech recognition engine uses whole word hidden Markov modelling, with limited vocabulary and constrained (finite state) grammar. The feature set used is based on Mel frequency spaced cepstral coefficients. The Viterbi search is used together with the level building algorithm to recognise speech within the utterances. Word-spotting techniques including a "garbage" model, are used. Various techniques compensating for noise and a varying channel transfer function are employed to improve the recognition rate. An Afrikaans conversational interface prompts the caller for information.

Detailed experiments illustrate the dependence and sensitivity of the system on its parameters, and show the influence of several techniques aimed at improving the recognition rate.

Samevatting

Sleutelwoorde: Spreker-onafhanklike spraakherkenning, telefoonoutomaat, dialoog, grammatika, heelwoord verskuilde Markov modellering, Mel frekwensie kepstrale koëffisiënte, Viterbi soektog, vlakbou algoritme, ruiskompensasie

Hierdie verhandeling beskryf die implementering van 'n intydse, spreker-onafhanklike telefoonoutomaat vanaf eerste beginsels. 'n Telefoonoutomaat is 'n geoutomatiseerde antwoorddiens wat die inbeller na 'n gevraagde persoon se uitbreiding deurskakel. Die outomaat bepaal met wie die inbeller wil praat deur 'n beperkte dialoog te voer en van spraakherkenning tegnologie gebruik te maak.

Die implementeringsplatform is 'n rekenaar met 'n telefoon koppelvlakkaart. Die spraakherkenner gebruik heelwoord verskuilde Markov modelle met 'n beperkte woordeskat en 'n begrensde grammatika om relatief swak kwalitiet spraak te herken. Die kenmerke van die spraak wat gebruik word vir herkenning is Mel frekwensiegespasieerde kepstrale koëffisiënte. Die Viterbi soektog saam met die vlakbou ("level building") algoritme word gebruik om die woorde te herken. Verskeie tegnieke word probeer om te kompenseer vir ruis en 'n variërende kanaal. 'n Afrikaanse dialoog (spraakkoppelvlak) lei die oproepvloei.

Eksperimentele resultate wat die stelsel se afhanklikheid en sensitiwiteit vir sy parameters beskryf, word gegee. Die resultate dien ook om die invloed van die verskillende tegnieke wat gebruik is om die herkenner te verbeter, te kwantifiseer.

Acknowledgements

I would like to thank the following people for their help and support, without which this project would not have been possible:

- Prof EC Botha
- Dr. Christoph Nieuwoudt
- Darryl Purnell
- I gratefully acknowledge the financial support of the NRF for this work.
- All the people who participated in the recording of the training data

Contents

Chapter 1

Introduction

The system implemented and studied in this dissertation is a telephone auto-attendant. The definition of an auto-attendant in this context is a system that is able to attend to incoming telephone calls without human intervention. The specific task undertaken is that of an auto-receptionist, in other words a system that, with the aid of a conversational interface, tries to determine with whom a caller wishes to speak, and puts the call through if a name was successfully negotiated. The conversational interface is implemented in the Afrikaans language spoken in South Africa.

The system is implemented on a personal computer equipped with an industry standard telephone line interface. The telephone line interface digitises audio waveforms from the analogue telephone line so that the speech waveforms can be processed by the computer. Speech recognition technology lies at the heart of the system, enabling the transformation of the digitised speech waveforms into representations the software can understand and interpret. Conversational feedback in the form of pre-recorded questions and responses is provided.

The auto-attendant was developed as a real-world implementation of leading edge speech recognition theory. The motivations for this work are:

- to determine the performance of a speech recognition system with the low bandwidth audio provided by the telephone system,
- to show that current computer hardware technology is fast enough for real-time speech processing and recognition, and
- to prove that speech recognition is a viable interface to a computer system.

Some of the characteristics of a system like this, are not only that there are in the order of 50 different vocabulary words to be recognised, but because these words are the names of people (with whom the callers are often unfamiliar), the pronunciation of the names varies greatly. A substantial amount of data is therefore necessary to train such a system. Even though many of the pronunciations are clearly incorrect, they were the natural response of the caller reading the name of the callee, and must therefore be regarded as a valid pronunciation.

Similar work has been done in the past, notably the last few years since computer hardware has started to approach acceptable performance for real-time speech recognition. Section 1.2 details these developments.

1.1 Problem statement

A system must be implemented which can interface a computer with a telephone line. This system must be able to playback and record data to and from the telephone line. A conversational interface must guide the caller into giving the name of a person he wishes to speak to (the system must thus act as a telephone auto attendant). The name must be recognised from the incoming wave data, and, if the computer has a sufficiently high confidence that the recognition was correct, the extension number of the person must be looked up, and the call must be forwarded to that number. If the recognition confidence is not high enough, the system must prompt the caller into confirming (medium confidence) or repeating (low confidence) the answer.

Experiments need to be performed off-line on pre-recorded telephone speech data to compare the performance of several techniques to enhance the speech recognition rate.

1.2 Related work

A large number of projects dealing with telephone speech recognition and the development of auto attendants exist worldwide:

Chien and Wang[1] researched methods of improving automatic speech recognition for telephone quality speech, and developed the phone-dependent channel compensated hidden Markov model (PDCC-HMM) for speech recognition. Their method involves taking into account the average channel transfer function during training so that no additional computation is necessary when the system is in the on-line mode. For this, they assume that the overall transfer function of the telephone channel does not have a large variance.

Guojun *et al.*[2] developed an automatic telephone operator that routed calls based on recognizing the callee's name. They required a fixed format *firstname surname* combination, and achieved 99% recognition in their preliminary results using a small data set.

Fraser *et al.*[3] developed a system named "Operetta" which had a more sophisticated conversational interface, allowing user friendliness of call handling. Operetta could handle eight simultaneous telephone calls using low-cost technology.

Koumpis *et al.*[4] developed an auto attendant system for the Technical University of Crete. They used a phoneme based recogniser, and achieved 97.5% correct name retrieval for a dictionary of 350 names and services. Their system used the commercial Nuance Communications' speech recognition engine, and the dictionary of names and services was created using strings of phonemes.

Kellner *et al.*[5] from Philips developed an automatic telephone switchboard and directory information system. This system combined state-of-the-art speech recognition and language understanding to deliver a system which could understand very complex dialogues, and provide the user with a variety of information or switch the call through to a desired party. With a database of over 600 names, they achieved a dialogue success rate (i.e. the dialogue concluded successfully, often with retry attempts included) of 95% overall.

Schramm *et al.*[6], building on the previously mentioned Philips work, developed an advanced system to deal with large scale directory assistance. A two-stage approach was used, where the first stage asked for a city name (in Germany), and the second stage prompted the user for a last name and first name. The largest of the directories was for the city of Berlin, containing 1.3 million entries. They showed that for cooperative users more than 90% of all simple requests can be automated successfully. If the last name was not recognised accurately, the caller would be asked to spell out the name.

1.2.1 Similar work in the commercial sector

After this dissertation was started, computers started becoming powerful enough to warrant commercial exploitation of speech recognition. The result is that several commercial software developers' toolkits (SDK's) have been developed, enabling application developers to build telephone based speech recognisers and interactive systems with minimum effort. At least two such SDK's are currently available from Nuance [1] and Speechworks [2]. However, none of the commercially available systems currently offer Afrikaans, or any other official South African language (other than English) as a language option.

[1] www.nuance.com
[2] www.speechworks.com

1.3 Approach

Our group decided to use the hardware of an industry leader in telephone speech interfacing, to ensure reliability and quality of speech data. A Dialogic D/21H card was used together with a Pentium 133Mhz class computer for the hardware platform of our system.

A decision was made to implement the speech recognition using whole word hidden Markov models, partly because no labeled phonemic database existed for the South African accent.

Initially it was decided to use and modify (as necessary) the OGI CSLU[3] speech toolkit with the Dialogic card in the Solaris operating system. After encountering some difficulties, our group decided to implement our own object oriented speech toolkit, the Hidden Markov Toolkit for Speech Recognition (HMTSR[4]). During the course of this dissertation this toolkit was further enhanced by adding features such as support for additional wave file formats, support for creating and using finite state grammar networks and the inclusion of duration modeling. Support for various noise compensation techniques was also added.

For the purpose of this dissertation, HMTSR had to be ported to the Microsoft Windows NT platform because of limitations in the Dialogic support for Solaris[5]. Software was then written to integrate the HMTSR toolkit with the Dialogic drivers, and control the flow of the conversational interface and telephone line control with some state machines.

For the purpose of recording training data, the system was configured so that it did not perform recognition, but rather prompted callers to say the names of people. This

[3] Oregon Graduate Institute Center for Spoken Language Understanding. Toolkit available from website: `http://cslu.cse.ogi.edu/toolkit/`

[4] See website `http://www.ee.up.ac.za/ee/pattern_recognition_page/HMTSR/HMTSR-0.2.tgz`

[5] As well as the lack of support at that stage for Linux drivers.

training data was then labeled (i.e. the precise beginning and ending of each word stored along with a name) so that it could be used for training the hidden Markov model representations.

Testing of the system was performed off-line with pre-recorded utterances, so that any modifications to the system or parameters could be tested against consistent speech data.

1.4 Contributions

This dissertation makes the following contributions to the field of automatic speech recognition:

- The process of building a speech recognition system from first principles is explained in detail.
- The performance sensitivity with regard to different techniques and system parameters is shown, giving a comparative basis of the merits of each approach.
- The system is the first of its kind known to be developed in the Afrikaans language.
- The HMTSR package which was developed by our group provided the functionality to train HMM models from speech data, and perform a level-building search with an FSN grammar on a speech utterance. For the purposes of this dissertation, I added the capabilities to optionally perform Cepstral Mean Subtraction, Spectral Mean Subtraction, duration modeling (instead of using state transition probability matrix), normalisation of confidence measure (using a garbage model) and robust start/end point detection. Furthermore, I wrote a graphical program (using the Qt widget set in linux) to create and manipulate the FSN grammars,

and integrated the file format into HMTSR. I also co-authored[6] a windows program to ease labeling of the data, as well as providing the ability to compare labeled test data and the transcribed output of the recognition engine. I ported HMTSR to Microsoft Windows NT, and integrated it into software I developed to interface to the Dialogic card, which spawns the conversational interface state machine in a thread so that multiple simultaneous calls can be handled.

Several publications have also arisen from this work.

1.5 Overview

The dissertation is organised as follows:

- Background Theory

  This chapter provides the necessary technical information needed to understand the auto-attendant system. Since speech recognition is an application of pattern recognition, many of the fundamentals of pattern recognition apply. Feature extraction needs to be performed on the raw speech signal to provide distinctive information for the pattern recognition process.

  A suitable pattern comparison technique needs to be found, as well as a method for "training" the system with existing speech data.

  Several techniques for optimising system performance are also considered.

- System

  The hardware and software architecture of the system is provided, from a top-down overview to implementation details and specific parameters employed.

- Experiments

  A chapter on experimental work details the data sets used as well as experiments

[6] The other programmer was Janus Brink.

performed to optimise system parameters and to benchmark system performance. The performance sensitivity of the system to the different methods and parameters is determined from these experiments.

- Conclusion

  A conclusion is given which summarizes the work done and provides details of future work that evolves from this dissertation.

- Appendix

  The appendix contains detailed descriptions of the source code implementation.

Chapter 2

Background Theory

2.1 System overview

The framework of the telephone speech recognition engine is provided here, enabling the reader to tie up the theory with the various system components.

The input to the system is analogue speech data from a telephone line. This input data is converted into digital form by a special analogue to digital converter card adapted to a telephone connection (see Figure 2.1).

This raw digital waveform data then needs to be analysed and processed to extract only the distinguishing characteristics from the data, reducing the sheer volume of data and easing the task of the classifier in the speech recognition algorithm. This process is called *feature extraction.*

The system needs to be *trained* to recognise speech, and for this the features of a training set are provided to a training algorithm. Once the system has been trained, it can recognise utterances that are similar to the training set.

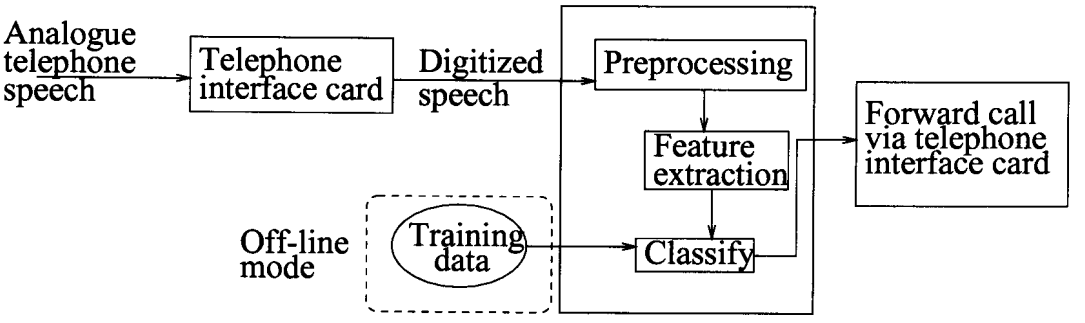

Figure 2.1: Block diagram of telephone speech recognition system

Various preprocessing techniques can be utilised to improve the feature set, e.g. filtering out noise, and a few of these techniques are discussed. A conversational interface prompts the caller into giving people's names, and a finite state grammar network assists the classifier in recognising the utterances. If a callee name is correctly recognised, the telephone call is forwarded to the appropriate telephone number via the PABX.

In off-line mode, pre-recorded training and testing data can be used to form a basis for performing experiments to compare different speech recognition techniques. Some techniques to improve the efficiency (speed and accuracy) of the training and recognition algorithms are also described in this chapter.

2.2 Telephone quality speech

The telephone system in South Africa utilises copper pairs to carry analogue signals from each telephone point to a Central Office. From here the telephone signals are usually digitised to 8 bit resolution at 8kHz sampling frequency and routed to the destination where the digital signal is converted into an analogue signal again. This digitisation process leads to two problems:

1. An effective upper signal bandwidth limit of 4kHz, based on Nyquist's theory

2. Noise due to finite voltage sampling levels (8 bit leads to 256 levels)

Furthermore, due to a combination of filters, transformers and signal transducers in various parts of the telephone system, non-linearities and a lower bandwidth limit of about 300Hz are also imposed on the signal.

The effective voice channel of the telephone system can be approximated by a non-linear bandpass filter with lower and upper cut-offs of 300Hz and 3.7kHz respectively, and with 8-bit quantisation noise.

2.3 Cepstra as features for speech recognition

Due to the local stationarity of the speech signal, short-time spectral estimates of a speech signal are used in speech recognition applications as a basis for extracting features from the continuous signal (from Rabiner *et al.* [7]). A short-time spectrum is normally obtained by placing a data window on the sample sequence, and the resultant spectral estimate is considered a snapshot of the speech characteristics at a particular instant t. This method effectively discretises the continuous speech signal. This spectral representation is denoted by $S(\omega, t)$ to emphasise the time variability of the speech. The data window slides across the signal sequence, producing a series of short-time spectral representations, $S(\omega, t)_{t=0}^{T}$.

A relatively robust, reliable feature set for speech recognition can be found by computing the cepstral coefficients of a speech window. The complex cepstrum of a signal is defined as the Fourier transform of the log of the signal spectrum. For a power spectrum (magnitude-squared Fourier transform) $S(\omega)$, which is symmetric with respect to $\omega = 0$ and is periodic for a sampled data sequence, the Fourier series representation of $logS(\omega)$ can be expressed as

$$logS(\omega) = \sum_{n=-\infty}^{\infty} c_n e^{-jn\omega} \tag{2.1}$$

where $c_n = c_{-n}$ are real and referred to as the cepstral coefficients.

The cepstral coefficients are therefore coefficients of the Fourier transform representation of the log magnitude spectrum. This representation of the speech spectrum provides a good representation of the local spectral properties of the signal for the given analysis frame. The cepstral coefficients thus give a feature vector for a given speech window. The lower index coefficients represent the spectral envelope, whereas the higher index coefficients represent the pitch and excitation signals [8].

The following equation shows that the c_0 coefficient is directly proportional to the energy underneath the log power spectrum:

$$c_0 = \int_{-\pi}^{\pi} \frac{logS(\omega)}{2\pi} d\omega. \tag{2.2}$$

It has been shown [7, p. 166] that the cepstral coefficients (except c_0) have

- zero means
- variances essentially inversely proportional to the square of the coefficient index ($E\{c_n^2\} \propto \frac{1}{n^2}$).

Spectral transitions are believed to play an important role in human speech perception. This leads to the idea [8] that an improved representation can be obtained by extending the analysis to include information about the temporal cepstral derivative. Both first and second order temporal derivatives have been investigated and found to improve the performance of speech-recognition systems. To introduce temporal order into the cepstral representation, we denote the m^{th} cepstral coefficient at time t by $c_m(t)$, where

t refers to the analysis speech frame in practice. The cepstral time derivative is approximated as follows[7, p. 116]: The time derivative of the log magnitude spectrum has a Fourier series representation of the form

$$\frac{\partial}{\partial t}[log|S(e^{j\omega},t)|] = \sum_{m=-\infty}^{\infty} \frac{\partial c_m(t)}{\partial t} e^{-j\omega m}. \tag{2.3}$$

Since $c_m(t)$ is a discrete time representation, using a first- or second-order difference equation is inappropriate to approximate the derivative, as it is too noisy. The partial derivative $\partial c_m(t)/\partial t$ can be approximated by an orthogonal polynomial fit (least squares estimate) over a finite length window:

$$\frac{\partial c_m(t)}{\partial t} = \Delta c_m(t) \approx \mu \sum_{k=-K}^{K} k c_m(t+k), \tag{2.4}$$

where μ is an appropriate normalization constant and (2K+1) is the number of frames over which the computation is performed. Typically a value of $K = 3$ has been found appropriate for computation of the first-order temporal derivative.

2.3.1 Optimal encoding of temporal information

Other research by Milner [9] indicates that a polynomial approximation for temporal derivatives does not optimally encode the temporal information. He calculates a temporal transformation matrix $\boldsymbol{H}$, so that the dot product of $\boldsymbol{M}$ (a matrix consisting of M rows of successive cepstral column feature vectors) and $\boldsymbol{H}$ gives the generalised feature matrix $\boldsymbol{V}$.

$$\boldsymbol{V} = \boldsymbol{M} \cdot \boldsymbol{H}$$

He shows that it is important that the columns of $\boldsymbol{H}$ are orthogonal to decorrelate the columns of $\boldsymbol{V}$ (thus extracting temporal information). He goes on to show several

different $\boldsymbol{H}$ matrices that can effectively perform transformations to $\boldsymbol{M}$ to extract temporal information. For example, to generate static, first and second order cepstral derivatives (based on, in his case, a regression analysis), the matrix $\boldsymbol{H}$ would be,

$$\boldsymbol{H} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & -2 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \tag{2.5}$$

with the stack width, M, set to 5. The 0^{th} column of the resulting feature matrix, $\boldsymbol{V}$, is the static cepstrum, the 1^{st} column is the first derivative, and the 2^{nd} column gives the second derivative.

His research shows that to optimally encode the temporal information, the covariance matrix of $\boldsymbol{M}$ needs to be diagonalised, and this is achieved by the Karhunen-Loeve Transform (KLT) matrix. By pooling together many examples from the speech training data, the covariance of the stacked cepstral coefficients can be determined. The KLT transform matrix, $\boldsymbol{H}$, is given by the eigenvectors of the covariance matrix. The eigenvectors which form the KLT basis functions (columns of the $\boldsymbol{H}$ matrix) are ranked in terms of their eigenvalues (i.e. the 0^{th} KLT basis function corresponds to the eigenvector which has the largest eigenvalue).

Using this KLT transform was shown by Milner to have a performance increase over using standard first and second order temporal derivatives, but unfortunately this research was discovered after the experiments in Chapter 4 had already been completed. This method can be considered for future work. Calculation of the KLT transform matrix is a computationally intensive task, but only needs to be performed once on the training data, so it does not affect the real-time operation of the system.

2.4 Mel scale

Psychophysical studies [7, p. 183] have shown that human perception of the frequency content of sounds, either for pure tones or for speech signals, does not follow a linear scale, e.g. a 4kHz sine audio wave is not perceived as being twice as high as a 2kHz tone. This has led to the idea of defining subjective pitch of pure tones. Thus for each tone with an actual frequency, f, measured in Hz, a subjective pitch is measured on a scale called the "mel" scale. As a reference point, the pitch of a 1kHz tone, 40dB above the perceptual hearing threshold, is defined as 1000 mels. Other subjective pitch values are obtained by adjusting the frequency of a tone such that it is half or twice the perceived pitch of a reference tone (with a known mel frequency).

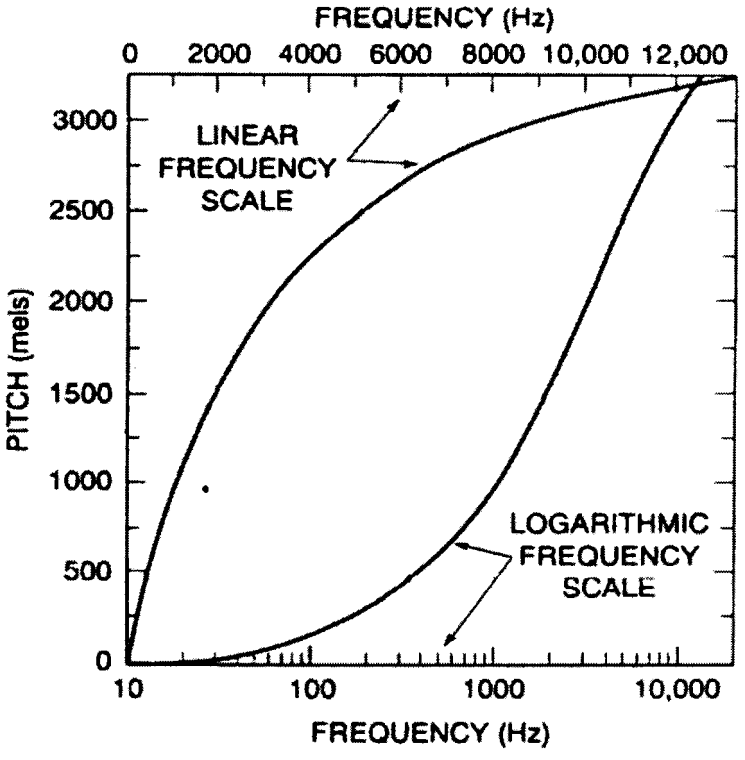

Figure 2.2: Mel scale[7, p. 184]

Figure 2.2 shows subjective pitch as a function of frequency. The upper curve shows the relationship of the subjective pitch to frequency on a linear scale; it shows that subjective pitch in mels increases less rapidly as the stimulus frequency is increased

linearly. The lower curve, on the other hand, shows the subjective pitch as a function of the logarithmic frequency; it indicates that the subjective pitch is essentially linear with the logarithmic frequency beyond about 1000Hz.

2.5 Hidden Markov Models

Hidden Markov models (HMM) [7, 10] of the acoustic feature vectors form the basis of most modern speech recognition systems. They have a number of desirable qualities which will be detailed in this section. Hidden Markov modelling finds its basis in statistical signal modelling, which is a widely used practice in pattern recognition. The HMM method provides a natural and highly reliable way of modelling and recognizing speech for a wide range of applications and integrates well into systems incorporating both task syntax and semantics. HMMs will be discussed only in the context of speech recognition, although they have much wider application, and were in fact not originally developed for speech recognition in particular.

An observable Markov model is a system that may be described as being in one of a set of N distinct states, which at regularly spaced, discrete times may undergo a change of state (possible back to the same state) according to a set of probabilities associated with each state (Figure 2.3).

The time instants associated with the state changes are denoted by $t = 1, 2, \ldots$, and the actual state at time t is given by q_t. A state in a discrete-time, *first-order* Markov chain has a probabilistic dependence only on the preceding state, i.e.,

$$P[q_t = j | q_{t-1} = i, q_{t-2} = k, \ldots] = P[q_t = j | q_{t-1} = i].$$

Only the Markov processes where this probability is independent of time are considered for speech recognition (i.e. the probability to go to the next state is only based on the current state and not on the history). Thus, the transition probabilities a_{ij} are given by

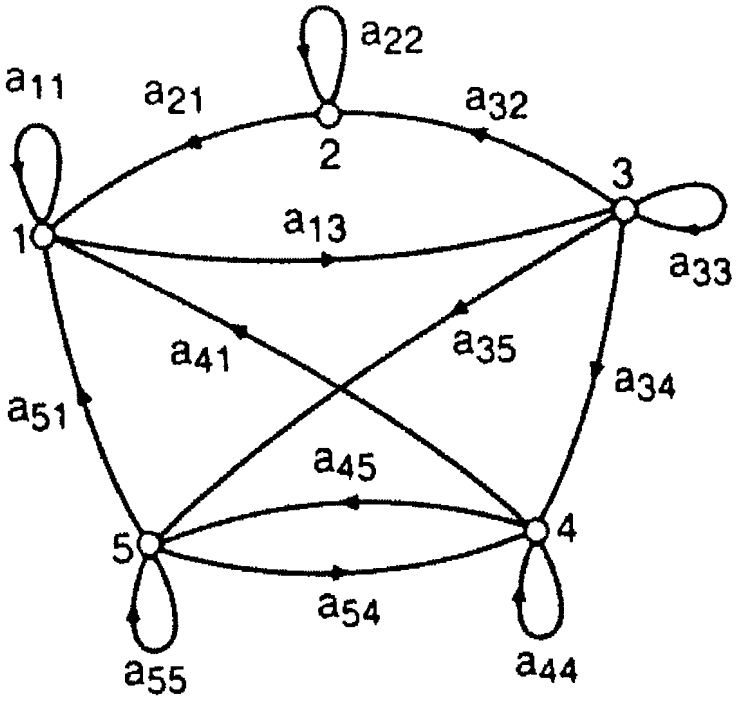

Figure 2.3: Hidden Markov Model

$$a_{ij} = P[q_t = j | q_{t-1} = i], \qquad 1 \leq i, j \leq N$$

where

$$a_{ij} \geq 0, \qquad \forall j, i \tag{2.6}$$

$$\sum_{j=1}^{N} a_{ij} = 1, \qquad \forall i \tag{2.7}$$

$$\tag{2.8}$$

to conform to standard stochastic constraints. The model described above is an observable Markov process because the output of the process is the set of states at each instant of time, where each state corresponds to an observable event. An example of such a Markov process is a simple traffic light, where the observable states are [green, orange, red],[1] and a possible transition matrix A could be given by

$$A = \{a_{ij}\} = \begin{bmatrix} 0.6 & 0.4 & 0 \\ 0 & 0.4 & 0.6 \\ 0.5 & 0 & 0.5 \end{bmatrix}$$

indicating that green cannot go to red, orange cannot go to green, and red cannot go to orange. Given this model, we can determine things such as the probability of a certain sequence of events, or what the probability is of a system staying in a certain state for a specified time.

Hidden Markov models extend *observable* Markov models by including the case in which the observation is a probabilistic function of the state instead of a directly observable event. That is, the resulting model is a doubly embedded stochastic process with an underlying stochastic process that is *not* directly observable (it is hidden), but can only be observed through another set of stochastic processes that produce the sequence of

[1] We could extend this to include flashing red for error, or blank for failure

observations. An example related to the speech recognition problem at hand can be given as follows: imagine that every possible sound that can be made by humans can be described by a finite set of numbers, each number corresponding to a single sound. [2] A hidden Markov model for a word ideally consists of as many states as are required to describe each transition from one sound to another through the word from beginning to end. The observation sequence produced by this model is a series of equally time spaced numbers, e.g.

4 4 4 4 4 7 7 7 2 1 1 1 1.

To match a particular word model to an observation sequence entails finding the probability of observing that sequence given a model with observation output probability at each state, and a given state transition matrix. Note that, as in this example, most HMM based speech recognizers use a strict left to right transition model (also called a Bakis model). This left-right model allows for auto-transitions and transitions to the next sequential state only. This configuration matches the left-right temporal property of a spoken word.

The formal elements of a discrete HMM can now be defined as follows:

- N, the number of states in the model. The number of states in a model cannot necessarily be deduced from the observation sequence by looking at the number of transitions. The states are labeled $\{1, 2, \ldots, N\}$, and the state at time t is denoted by q_t.
- M, the number of distinct observation symbols per state, or, in other words, the discrete alphabet size. The individual symbols are labeled $V = \{\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_m\}$.
- The state-transition probability distribution $A = \{a_{ij}\}$, where

$$a_{ij} = P[q_{t+1} = j | q_t = i], \qquad 1 \leq i, j \leq N,$$

[2] Note that this can be approximated in reality by using a vector quantizer codebook, although this does result in serious performance degradation compared to continuous densities.

$$a_{ij} = 0, \qquad j < i, j > i+1 \text{(Bakis model)}.$$

This describes a left-right model without skipping transitions, i.e. the state can only go to the next sequentially numbered state, or stay within itself, as is commonly used in speech recognition. As will be seen in Section 2.9, a transition matrix is not optimal for speech recognition.

- The observation symbol probability distribution, $B = \{b_j(k)\}$, where

$$b_j(k) = P[\boldsymbol{o}_t = \boldsymbol{v}_k | q_t = j], \qquad 1 \leq k \leq M$$

defines the symbol distribution in state j. $\boldsymbol{o}_t$ is the observation vector at time t.

- An initial state distribution $\pi = \{\pi_i\}$ in which

$$\pi_i = \begin{cases} 0, & i \neq 1 \\ 1, & i = 1 \end{cases} \tag{2.9}$$

for a strict left-right model, to ensure that the model starts in the first state. (The probability for a model to be in state i *initially* is given by π_i.)

As has previously been mentioned, observations with a discrete probability density lead to a serious performance degradation for speech recognition, since we are dealing with a continuous signal from which feature vectors can be extracted. It would be advantageous to use HMMs with continuous observation densities to model the continuous signal representations directly.

To use a continuous observation density, a probability density function (pdf) must be chosen which has parameters that can be estimated consistently from the training data. The most general representation of the pdf, for which the parameter estimation equations can be expressed analytically, is a finite mixture of Gaussian pdf's of the form

$$b_j(\boldsymbol{o}) = \sum_{k=1}^{M} c_{jk} \Psi(\boldsymbol{o}, \boldsymbol{\mu}_{jk}, \boldsymbol{U}_{jk}), \qquad 1 \leq j \leq N \tag{2.10}$$

where $\boldsymbol{o}$ is the observation vector being modelled, c_{jk} is the mixture coefficient for the kth mixture in state j and Ψ is a Gaussian probability density function with mean vector $\boldsymbol{\mu}_{jk}$ and covariance matrix $\boldsymbol{U}_{jk}$ for the kth mixture component in state j. The mixture gains c_{jk} satisfy the stochastic constraint

$$\sum_{k=1}^{M} c_{jk} = 1, \qquad 1 \leq j \leq N$$

$$c_{jk} \geq 0, 1 \leq j \leq N, 1 \leq k \leq M$$

so that the pdf is normalized, that is

$$\int_{-\infty}^{\infty} b_j(\boldsymbol{o}) d\boldsymbol{o} = 1, \qquad 1 \leq j \leq N. \tag{2.11}$$

The pdf in Eq. (2.10) can be used to approximate with arbitrary precision any finite continuous density function, making it suitable for the speech recognition problem.

Three problem formulations exist with regard to HMMs that are of interest to the speech recognition problem:

1. Given an observation sequence $\boldsymbol{O} = (\boldsymbol{o}_1 \ldots \boldsymbol{o}_T)$ and a number of models, how does one compute the probability of each model, so that the most likely model to give that observation sequence can be found? This has direct application in *recognizing a feature sequence of a spoken word.*

2. Given the observation sequence and a model, how does one choose a state sequence $\boldsymbol{q} = (q_1 q_2 \ldots q_T)$ that best explains the observation (i.e. the optimal state sequence)? There is no "correct" state sequence, so an optimimality criterion is used to solve this problem. This has application in continuous speech recognition to find optimal state sequences, and is discussed in Section 2.7, and forms part of both the training and the recognition.

3. How does one adjust the model parameters to maximise the probability of an observation sequence given the model $P(\boldsymbol{O}|\lambda)$? The observation sequence used

to adjust the model parameters is called a "training" sequence. This is an important problem for speech recognition since it allows one to optimally adapt model parameters to observed training data, giving the best model for a real speech signal during training. Section 2.6 shows how the "expectation-maximization" method can be used to iteratively select these model parameters, given one or more training sequences.

2.6 Expectation-Maximisation

The most difficult problem in HMMs is to determine a method to adjust the model parameters to satisfy a certain optimization criterion. There is no analytical way to solve for these parameters to maximize the probability of the observation sequence, but an iterative procedure known as the Baum-Welch [7] method can be used to choose the maximum likelihood (ML) parameters.

We define $\xi(i,j)$ as the probability of being in state i at time t, and state j at time $t+1$, given the model and observation sequence, and can be written as:

$$\xi(i,j) = P(q_t = i, q_{t+1} = j|\boldsymbol{O}, \lambda) \tag{2.12}$$

$$= \frac{\alpha_t(i)a_{ij}b_j(\boldsymbol{o}_{t+1})\beta_{t+1}(j)}{\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_t(i)a_{ij}b_j(\boldsymbol{o}_{t+1})\beta_{t+1}(j)} \tag{2.13}$$

where $\alpha_t(i)$ is defined as the probability of the partial observation sequence $\boldsymbol{o}_1\boldsymbol{o}_2\cdots\boldsymbol{o}_t$ (until time t) in state i at time t, and $\beta_t(i)$ is defined as the probability of the partial observation sequence from $t+1$ to the end, given state i at time t.

If we define $\gamma_t(i)$ as the probability of being in state i at time t given the entire observation sequence and the model, we can relate $\gamma_t(i)$ to $\xi_t(i,j)$ by summing over j, giving

$$\gamma_t(i) = \sum_{j=1}^{N} \xi_t(i,j) \tag{2.14}$$

If we sum $\gamma_t(i)$ over the time index t, we get a quantity that can be interpreted as the expected (over time) number of transitions made from state i (if we exclude $t = T$). Similarly, summation of $\xi_t(i,j)$ over $t = 1$ to $t = T-1$ can be interpreted as the expected number of transitions from state i to state j. Using this knowledge, we can get a set of reasonable reestimation formulas for π, A, and B:

$$\bar{\pi}_j = \gamma_1(i) \tag{2.15}$$

$$\bar{a}_{ij} = \frac{\text{expected number of transitions from state i to state j}}{\text{expected number of transitions from state i}} \tag{2.16}$$

$$= \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)} \tag{2.17}$$

$$\bar{b}_j(k) = \frac{\text{expected number of times in state j and observing symbol } \boldsymbol{v_k}}{\text{expected number of times in state j}} \tag{2.18}$$

$$= \frac{\sum_{\text{t=1 } o_t=v_k}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)} \tag{2.19}$$

If we define the current model as $\lambda = (A, B, \pi)$ and use that to compute the right hand side of Eqs. (2.15)-(2.18), and we define the reestimated model as $\bar{\lambda} = (\bar{A}, \bar{B}, \bar{\pi})$ as determined from the left-hand side of Eqs. (2.15)-(2.18), then it has been proven by Baum *et al.* that either the initial model λ defines a critical point of the likelihood function, in which case $\bar{\lambda} = \lambda$, or model $\bar{\lambda}$ is more likely than model λ in the sense that $P(\boldsymbol{O}|\bar{\lambda}) > P(\boldsymbol{O}|\lambda)$; i.e. we have found a new model from which the observation sequence is more likely to have been produced. If we iteratively use this procedure, we can improve the probability of $\boldsymbol{O}$ being observed from the model until some limiting

point is reached. The final result of this reestimation procedure is a maximum likelihood estimate of the HMM.

The reestimation formulas for continuous observation densities in HMMs are of the form:

$$\bar{c}_{jk} = \frac{\sum_{t=1}^{T} \gamma_t(j,k)}{\sum_{t=1}^{T} \sum_{k=1}^{M} \gamma_t(j,k)} \tag{2.20}$$

$$\bar{\boldsymbol{\mu}}_{jk} = \frac{\sum_{t=1}^{T} \gamma_t(j,k) \boldsymbol{o}_t}{\sum_{t=1}^{T} \gamma_t(j,k)} \tag{2.21}$$

$$\bar{\boldsymbol{U}}_{jk} = \frac{\sum_{t=1}^{T} \gamma_t(j,k)(\boldsymbol{o}_t - \boldsymbol{\mu}_{jk})(\boldsymbol{o}_t - \boldsymbol{\mu}_{jk})'}{\sum_{t=1}^{T} \gamma_t(j,k)} \tag{2.22}$$

where $\gamma_t(j,k)$ is the probability of being in state j at time t with the kth mixture component accounting for $\boldsymbol{o}_t$, i.e.

$$\gamma_t(j,k) = \left[\frac{\alpha_t(j)\beta_t(j)}{\sum_{j=1}^{N} \alpha_t(j)\beta_t(j)}\right]\left[\frac{c_{jk}\eta(\boldsymbol{o}_t, \boldsymbol{\mu}_{jk}, \boldsymbol{U}_{jk})}{\sum_{m=1}^{M} c_{jm}\eta(\boldsymbol{o}_t, \boldsymbol{\mu}_{jk}, \boldsymbol{U}_{jk})}\right] \tag{2.23}$$

These equations can be interpreted as follows: The reestimation formula for c_{jk} is the ratio between the expected number of times the system is in state j using the k^{th} mixture component, and the expected number of times the system is in state j. The reestimation formula for the mean vector $\boldsymbol{\mu}_{jk}$ weights each numerator term by the observation, thereby giving the expected value of the portion of the observation vector accounted for by the k^{th} mixture component. The covariance matrix $\boldsymbol{U}_{jk}$ is also weighted by the observation vector.

As previously mentioned, left-right or Bakis models are commonly used for speech recognition. Due to the transient nature of the states within a model of this form, only a small amount of observations are available for any state. Therefore, to have

sufficient data to make reliable estimates of all model parameters, one has to use multiple observation sequences.

2.6.1 Initial Estimates of HMM parameters

Since the reestimation formulas rely on previous values of the HMM parameters, a question that arises is "How does one choose initial estimates of the HMM parameters so that the local maximum obtained after reestimation is equal to or as close as possible to the global maximum of the likelihood function?"

Experience has shown that either random (subject to the stochastic and nonzero restraints) or uniform initial estimates of the π and A parameters are adequate for giving estimates of these parameters in almost all cases. However, for the B parameters, experience has shown that good initial estimates are essential in the continuous-distribution case [7, p. 370].

A procedure for providing good initial estimates of the B parameters has been devised and is called the segmental K-means training procedure [7, p. 382]. This iterative procedure also requires an initial model estimate but this can be chosen either randomly or based on any available model appropriate to the data.

The set of training observation sequences is segmented in states, based on the current model λ. This sequence is achieved by finding the optimal state sequence, via the Viterbi algorithm, and then backtracking along the optimal path. The result of segmenting each of the training sequences is, for each of the N states, a maximum likelihood estimate of the set of the observations that occur within each state j according to the current model.

Since we are using continuous observation densities, a segmental K-means procedure is used to cluster the observation vectors within each state j into a set of M clusters using a Euclidean distortion measure. Each cluster represents one of the M mixtures

of the $b_j(\boldsymbol{o}_t)$ density. From the clustering, a set of updated model parameters is derived as follows:

$$\begin{aligned}
\hat{c}_{jm} &= \text{number of vectors classified in cluster } m \text{ of state } j \text{ divided by} \\
&\quad \text{the number of vectors in state } j \\
\hat{\boldsymbol{\mu}}_{jm} &= \text{sample mean of the vectors classified in cluster } m \text{ of state } j \\
\hat{\boldsymbol{U}}_{jm} &= \text{sample covariance matrix of the vectors classified in} \\
&\quad \text{cluster } m \text{ of state } j
\end{aligned}$$

Based on this state segmentation, when using a transition matrix, updated estimates of the a_{ij} coefficients can be obtained by counting the number of transitions from state i to state j and dividing it by the number of transitions from state i to any state (including itself).

An updated model $\hat{\lambda}$ is obtained from the new model parameters, and the formal reestimation procedure (Equations (2.20), (2.21) and (2.22)) is used to reestimate all model parameters. The resulting model is then compared to the previous model by computing a distance score that reflects the statistical similarity of the HMMs. If the model distance exceeds a threshold, then the old model λ is replaced by the reestimated model and the overall training loop is repeated. If the model distance falls below the threshold, then model convergence is assumed and the final model parameters are saved.

2.7 Viterbi search

The way in which the "optimal" state sequence associated with any given observation sequence is calculated is dependent on the definition of the optimal state sequence, i.e.

an optimality criterion needs to be specified. The most widely used criterion in speech recognition is to maximize $P(\boldsymbol{q}|\boldsymbol{O},\lambda)$, which is equivalent to maximizing $P(\boldsymbol{q},\boldsymbol{O}|\lambda)$. A formal technique called the Viterbi algorithm exists for finding this single best state sequence, and is formulated as follows [7]:

To find the single best state sequence, $\boldsymbol{q} = (q_1 q_2 \dots q_T)$, for the given observation sequence $\boldsymbol{O} = (\boldsymbol{o}_1 \boldsymbol{o}_2 \dots \boldsymbol{o}_T)$, we need to define the quantity

$$\delta_t(i) = \max_{q_1,q_2,\dots,q_{t-1}} P[q_1 q_2 \dots q_{t-1}, q_t = i, \boldsymbol{o}_1 \boldsymbol{o}_2 \dots \boldsymbol{o}_t|\lambda]. \tag{2.24}$$

that is, $\delta_t(i)$ is the highest probability along a single path at time t which accounts for the first t observations and ends in state i. By induction we get

$$\delta_{t+1}(j) = [\max_i \delta_t(i) a_{ij}] b_j(\boldsymbol{o}_{t+1}). \tag{2.25}$$

To retrieve the actual state sequence, we need to keep track of the argument that maximized Eq. (2.25) for each t and j. We can do this via the array $\psi(j)$. The complete procedure for finding the best sequence can be stated as follows (using the log to reduce the number of multiplications required):

1. Preprocessing

$$\begin{aligned} \tilde{\pi}_i &= log(\pi_i), 1 \leq i \leq N \\ \tilde{b}_i(\boldsymbol{o}_t) &= log[b_i(\boldsymbol{o}_t)], 1 \leq i \leq N, 1 \leq t \leq T \\ \tilde{a}_{ij} &= log(a_{ij}), 1 \leq i, j \leq N \end{aligned} \tag{2.26}$$

2. Initialization

$$\begin{aligned} \tilde{\delta}(i) = log(\delta_1(i)) &= \tilde{\pi}_i + \tilde{b}_i(\boldsymbol{o}_1), 1 \leq i \leq N \\ \psi_1(i) &= 0, 1 \leq i \leq N \end{aligned} \tag{2.27}$$

3. Recursion

$$\begin{aligned}\tilde{\delta}_t(j) &= \max_{1\leq i\leq N}[\tilde{\delta}_{t-1}i(i) + \tilde{a}_{ij}] + \tilde{b}_j(\boldsymbol{o}_t) \\ \psi(j) &= arg \max_{1\leq i\leq N}[\tilde{\delta}_{t-1}i(i) + \tilde{a}_{ij}], 2 \leq t \leq T, 1 \leq j \leq N\end{aligned} \tag{2.28}$$

4. Termination

$$\begin{aligned}\tilde{P}^* &= \max_{1\leq i\leq N}[\tilde{\delta}_T(i)] \\ q_T^* &= arg \max_{1\leq i\leq N}[\tilde{\delta}_T(i)]\end{aligned} \tag{2.29}$$

5. Backtracking

$$q_t^* = \psi_{t+1}(q_{t+1}^*), t = T-1, T-2, \ldots, 1$$

The preprocessing step needs to be performed only once and saved, thus the cost is negligible. The rest of the calculations required are in the order of N^2T additions.

2.8 Level building algorithm

The Viterbi method on its own works well for doing isolated word recognition, i.e. matching a single spoken word to one of several word models. For many applications, notably those referred to as "command-and-control" applications, this approach works well and is appropriate. However, for the purpose of this dissertation, the recognition engine needs to be able to cope with a sequence of words from a limited recognition vocabulary, and thus using the Viterbi method alone will not suffice.

To formulate the problem at hand we need to make some definitions [7]. Assume that we are given the sequence of feature vectors (extracted from the speech signal, e.g. Mel-scaled cepstra) $T = \boldsymbol{t}(1), \boldsymbol{t}(2), \ldots, \boldsymbol{t}(M)$, and we also have the HMM word models $(R_i, i = 1..v)$ for each of V vocabulary words.

$$R_i = \boldsymbol{r}_i(1), \ldots, \boldsymbol{r}_i(N_i)$$

where N_i is the number of states in the i^{th} word reference pattern.

The connected word-recognition problem can be summarised by the question: Given a fluently spoken sequence of words, how can we determine the *optimum* match in terms of concatenation of word models? That is, how do we find R^* (the best possible sequence of reference patterns) that best matches T (the test pattern)?

The following problems arise when dealing with connected-word recognition:

- The number of words, L, in the string is not usually known (although often upper and lower bounds exist).
- The word utterance boundaries are unknown, except for the beginning of the first word and the end of the last word.[3]
- The word boundaries are fuzzy and not uniquely identifiable. It is difficult to automatically find accurate word boundaries because of sound coarticulation, i.e. the ending sound of one word is "morphed" into the beginning sound of the next word when spoken fluently.
- For a set of V word models, and for a given value of L (the number of words in the string, if known), there are V^L possible combinations of composite matching patterns; for anything but extremely small values of V and L the exponential number of composite matching patterns implies that the connected word-recognition problem cannot be solved by exhaustive means.

A non-exhaustive matching algorithm is required to solve the connected word-recognition problem efficiently. Fortunately algorithms exist with which to do this. In particular, the level building algorithm will be discussed in this text.

[3]The beginning and end of the utterance could be silence or noise, but that can also be classified as a word model.

Assume there are L word patterns in R^*, where L varies between the minimum and maximum possible values. The best possible sequence of reference patterns R^* is given by a concatenation of L reference patterns:

$$R^* = R_{q^*(1)} \oplus R_{q^*(2)} \oplus \ldots \oplus R_{q^*(L)}$$

where each index $q^*(l)$ is in the range [1,V].

An approach to calculate R^* efficiently, called the Level-Building algorithm, has been devised [7, pp. 400-422].The basic approach is as follows: At the first "level", a Viterbi alignment is performed with each reference model to the test sequence. The range of starting frames (initial state) is 1 to $T - N_v$, where N_v is the number of states in each reference model, and T is the number of frames in the test sequence. The range of ending frames (final state) for each Viterbi alignment is given by N_v to T. These parallelogram style ranges result from the strict left-right limitation of the Bakis model HMMs. This first level results in probability estimates at the final state for each model at test frames N_i to T. Over all the models, a minimum starting frame for the next level is determined by the model with the smallest number of states, say N_m. The model with the highest probability at each point in the range N_m to T is stored along with the probability value, and this vector is the result of the first "level." At the next level, the calculation is picked up starting from time point N_m where the previous level ended. Each reference model is again aligned with the rest of the test sequence, with the probabilities accumulating from the best models at each point from the previous level. After the second level, the new starting point will be $2N_m$. At the end of each level, the best scores and models are stored, as well as a backpointer indicating the source point that value was calculated from. After the iteration has gone through all the levels, the highest total accumulated probability at frame T over all levels is found (since the best matching sequence can also be less than L concatenated word models). By using the backpointer, the best reference frame sequence which resulted in this score can be found.

Figure 2.4 shows the resulting trellis shaped grid of results obtained for a particular reference model during the level building computation.

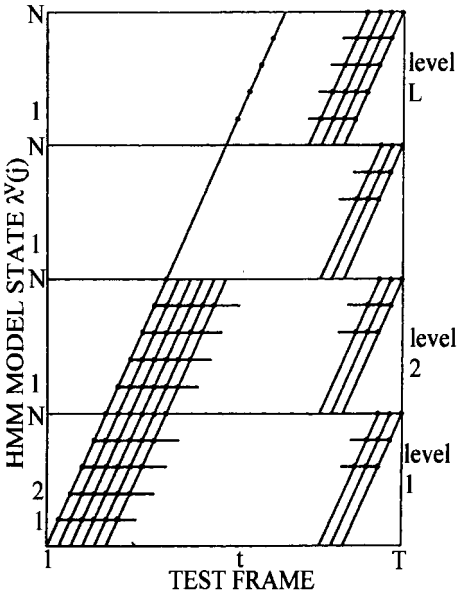

Figure 2.4: Level building algorithm applied to HMMs, from [7, p.424].

2.8.1 Frame synchronous approach

The standard level-building approach requires that the entire set of feature vectors T be known before recognition can start, which implies that it is not possible to start the recognition before the speaker has finished the utterance. An alternative approach is to use a method known as the frame synchronous level-building approach [11]. As its name suggests, this method can do most of the calculations frame by frame, making it useful for real-time implementations. For the case of HMMs using Viterbi alignment to eventually find the optimum path through the trellis of output observation probability densities, this approach can be simplified to merely calculating the output probability density for each incoming test frame for every state of each model, and storing the partially calculated accumulated Viterbi score. These calculations form the bulk of the computational cost of this speech recognition approach, and can be done with each

incoming frame, assuming enough computational power exists in the processor. At the end of the utterance the relatively low computational cost of backtracking and finding the highest accumulated scores on each "level" can be done to find the best overall matching sequence of concatenated word models.

2.9 Duration modelling

A major weakness of conventional HMMs is that they implicitly model state durations by a Geometric distribution, which is usually inappropriate. Erroneous recognition with HMM based systems is often associated with unreasonable occupancy durations of the (phoneme-like) states from which the model is built up.

Each state in a left-right (Bakis) HMM has an auto-transition probability a_{ii}, and a probability to go to the next state. The sum of these two probabilities is necessarily 1. Assuming that the model stays in this state, on average, for a duration τ, the Geometric distribution $p(\tau)$ is given by $p(\tau) = a_{ii}^{\tau}(1 - a_{ii})$. Figure 2.5 shows the relative decay over time for two cases, one with a high auto-transition probability, and one with a low auto-transition probability. It is obvious that for a high auto-transition probability, the decay is relatively slow, and this is in direct contrast with the actual probability of durations of acoustic events, which drop rapidly to zero when the duration is either too short or too long.

This exponential durational model imposes little limitation on how long an utterance can occupy a certain state in the model. This freedom often creates false alarms and/or substitutions by allowing an utterance to virtually skip over the unsuitable states, while lingering unreasonably long in more suitable states [12]. In an improved model for speech structure, duration is incorporated explicitly by specifying probability distribution functions for the occupancy durations of states. Although this intuitively will lead to better recognition results, computational and memory requirements may

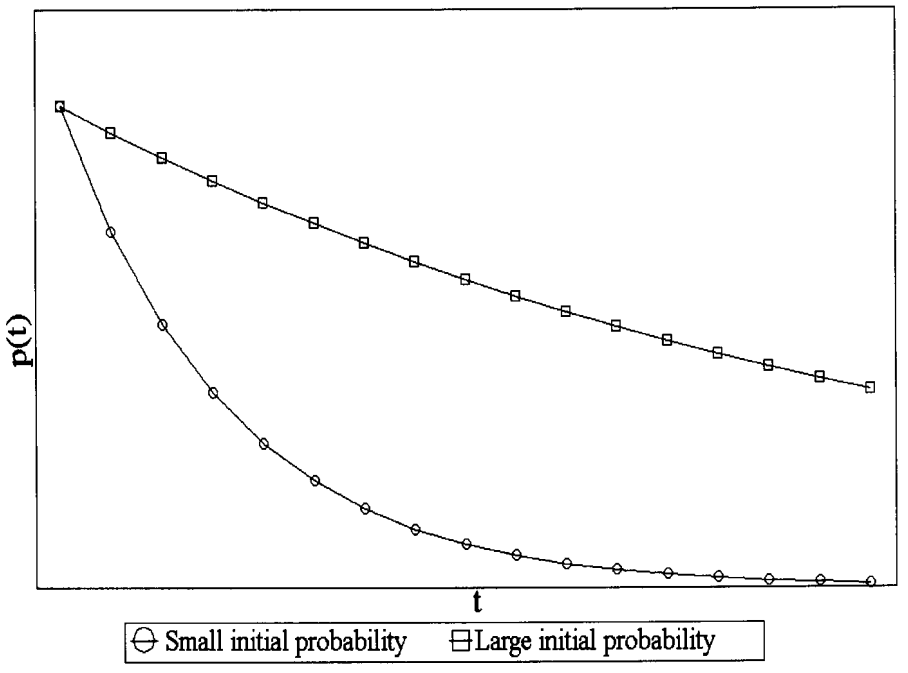

Figure 2.5: Implicit geometric duration modelling

be adversely affected. A method for implementing duration modelling with negligible overhead needs to be used to keep the system fast enough for real-time recognition.

Several methods of state duration modelling have been proposed in the literature to enhance the performance of HMM based speech recognition systems:

- The simplest state duration modelling technique is to have an upper and lower bound on the allowed state durations, and to disallow paths in the Viterbi search that exceed these specifications.
- In the Ferguson model [13] each state i has an associated discrete state duration probability density $d_i(\tau)$,$\tau = 1, 2, \ldots, \tau^i_{max}$ where τ^i_{max} is the largest duration allowed. Ferguson incorporated the estimation of $d_i(\tau)$ into the Baum-Welch reestimation algorithm. This method has two disadvantages, namely the excessive computational overhead that it imposes, and the excessive amounts of training data that might be required to estimate all the duration parameters: each state

i has τ_{max}^{i} duration parameters and sufficient statistics on each duration τ need to be collected at each state i so as to estimate $d_i(\tau)$ reliably.

- Russel & Moore [14] and Levinson [15] used parametric state duration distributions (Poisson and gamma distributions respectively) to solve the problem of requiring large amounts of training data. Their methods still imposed excessive computational requirements.

- Rabiner *et al.* [16] suggested a postprocessor approach in order to incorporate duration modelling in a computationally efficient way. Besides real-time difficulties, a major disadvantage of the backtracking approach is that the duration contribution to the standard Viterbi metric is only added after candidate paths have been collected. This could mean that the correct path might not be one of those candidates.

Burshtein [17] has suggested a practical approach to duration modelling that avoids the above mentioned problems. He proposes a modified Viterbi algorithm that incorporates duration modelling, and has essentially the same computational requirements of the conventional Viterbi algorithm.

Since his method also relies on a parametric distribution, he performed experiments dedicated to finding the most appropriate distribution for state durations. He compared histograms of state durations to a Gaussian distribution, gamma distribution and a geometric distribution (to show how inaccurate this implicit duration model is), and found the gamma distribution to most accurately describe the state duration (Figure 2.6).

He shows that the gamma distribution is capable of describing both the monotonic character of the geometric distribution, and the unimodal character of the Gaussian distribution. In addition, unlike the Gaussian distribution, the gamma distribution is one-sided; i.e. it assigns zero probability to negative τ's, which is appropriate for duration distributions. Finally, the gamma distribution possesses a slower decay rate,

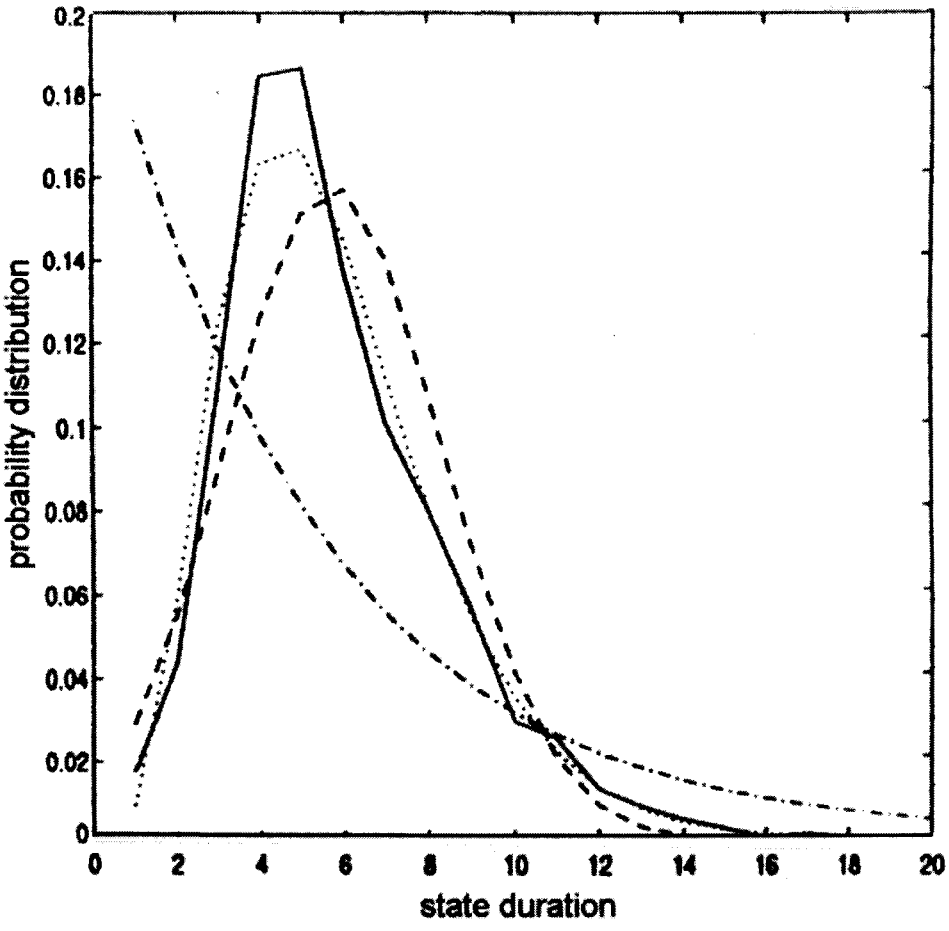

Figure 2.6: State duration distributions for the 7th state of the word 'seven': empirical distribution (solid line), Gauss fit (dashed line), Gamma fit (dotted line), geometrical fit (dash-dot line), from [17].

$e^{-\alpha x}$ which is more appropriate for duration modelling than the fast $e^{-\frac{x^2}{2\sigma^2}}$ decay of the Gaussian distribution.

The discrete gamma distribution [18] is given by

$$p(\tau) = Ke^{-\alpha\tau}\tau^{p-1}, \tau = 0, 1, 2, \ldots \tag{2.30}$$

where $\alpha > 0$ and $p > 0$. K is a normalizing term calculated so that the sum of $p(\tau)$ from 0 to ∞ equals 1. The parameters α and p can easily be estimated by noting that the mean and variance of the gamma distribution are given by $E\{\tau\} = p/\alpha$ and $VAR\{\tau\} = p/\alpha^2$. α and p can thus be estimated using the empirical expectation $\hat{E}(\tau)$ and empirical variance $\widehat{VAR}(\tau)$ of the duration by

$$\hat{\alpha} = \frac{\hat{E}(\tau)}{\widehat{VAR}(\tau)} \tag{2.31}$$

$$\hat{p} = \frac{\hat{E}^2(\tau)}{\widehat{VAR}(\tau)}. \tag{2.32}$$

The modified Viterbi algorithm keeps track of the duration $D_s(t)$ of each state s at time t. Let M_s denote the time at which the peak value of the gamma distribution occurs, at state s, and let $l(u) = log\ p(u)$, where $p()$ here is the gamma density. The duration penalty, P, of making a transition from state s at time t to state $\hat{s}$ at time $t+1$ is given by:

$$P = \begin{cases} 0 & if D_s(t) < M_s \text{ and } \hat{s} = s \\ l(D_s(t+1)) - l(D_s(t)) & if D_s(t) \geq M_s \text{ and } \hat{s} = s \\ l(D_s(t)) & if D_s(t) < M_s \text{ and } \hat{s} \neq s \\ l(M_s) & if D_s(t) \geq M_s \text{ and } \hat{s} \neq s. \end{cases} \tag{2.33}$$

The motivation is the following: before the duration is M_s, we do not penalize for remaining in the same state, unlike the conventional Viterbi algorithm ($P = 0$ in Eq.(2.33)). After the duration is M_s, we penalize gradually, unlike the backtracking

approach of Rabiner *et al.* mentioned earlier ($P = l(D_s(t+1)) - l(D_s(t))$ in Eq.(2.33)). If the next state is different to the current state and the duration is less than M_s, the penalty is $l(D_s(t))$. If moving to a new state and the duration is greater or equal to M_s, the penalty is fixed at $l(M_s)$.

The duration penalty is incorporated into the Viterbi algorithm instead of using the log state transition probability penalty $\tilde{a}_{ij}$ in Eq.(2.28). For greater computational efficiency, the $l(u)$ function can be stored in a lookup table for each state of each HMM, thus trading a relatively small amount of memory for a large computational saving.

It was found in practice that it is unnecessary to iteratively update the duration modeling parameters during training. Instead, the models are trained with the standard transition matrix reestimation formulas (Eq.(2.16)). After the models have been trained, each model is Viterbi aligned to all its training examples, and the empirical mean and variance of the durations of each state for each model are obtained as described below.

The sum of each state's durations (over all the training examples for each HMM) is stored as $state_{dur}$, and the sum of the squares of the durations is stored as $state_{dursq}$. These values can be used to calculate the empirical mean and variance values of duration for each state:

$$\hat{E}(\tau) = \frac{state_{dur}}{examples}$$
$$\widehat{VAR}(\tau) = \frac{state_{dursq}}{examples} - \hat{E}^2(\tau),$$

where *examples* is the number of training examples for each hmm. $\hat{E}(\tau)$ and $\widehat{VAR}(\tau)$ can now be substituted into Eqs.(2.31) and (2.32) to obtain the values of $\hat{\alpha}$ and $\hat{p}$ needed to calculate the discrete gamma distribution (Eq.(2.30)). This method is less computationally intensive and easier to implement than recursively updating the duration modeling parameters, and results in values that accurately model the state durations.

2.10 Channel compensation techniques

Environmental robustness and speaker independence are important issues of current speech recognition research. Normalisation methods might make use of the model but primarily influence the signal such that important information is kept and unwanted distortions are cancelled out.

When using the telephone system, the overall channel transfer function can differ from connection to connection. These variations can be due to the physical wires in different parts of the connection, different microphones and speakers, codecs such as GSM and others[4] and analogue to digital as well as digital to analogue converters in the signal path. These variations lead to degradation in speech recognition performance. To try to improve the performance in the presence of such a varying channel, a method needs to be used to try to compensate for the transfer function of the channel, effectively normalising the speech to the channel transfer function. Most modern speech recognition systems use a channel normalisation approach called Cepstral Mean Subtraction to compensate for the accoustic channel, as well as the speaker.

2.10.1 Cepstral Mean Subtraction

Cepstral Mean Subtraction[19] (CMS) is a simple but effective channel normalisation technique. Many adaptation methods require an adaptation method that has to be trained. In contrast to this, CMS is purely signal based and tries to eliminate disturbing channel and speaker effects before the signal is used to train a recogniser. It is also a very convenient technique to use if cepstra are already being used as the features for a recogniser.

[4]Telephone network providers are starting to use the internet as part of their core network in some countries. Telkom is also considering Voice-Over-IP to be an essential element of their Next Generation Network.

When a speech signal passes a linear time invariant channel, this convolutional distortion becomes multiplicative in the spectral domain, and additive in the log-spectral domain. Since the cepstrum is just a linear transformation of the log-spectrum, both can be regarded equally in this context. CMS deals with channel effects, such as the use of different microphones on differing telephone brands, or the use of a cellphone.

Cepstral Mean Subtraction works as follows:

For speech recognition, a short time analysis is performed, resulting in the speech spectrum $S_t(\omega)$ and the measured spectrum $Y_t(\omega)$ after modification by the channel transfer function $C(\omega)$ (which is assumed not to be a function of time), where

$$Y_t(\omega) = C(\omega) \cdot S_t(\omega)$$

and where $y_t(\omega)$ is the log-spectrum or cepstrum (the cepstrum is just a linear transformation of the log-spectrum, so both can be treated equally in this context), given by

$$y_t(\omega) = c + s_t$$

The assumption of a constant channel $C(\omega)$ allows to compensate for it by subtracting the mean of the measured cepstrum, leading to a cepstral mean subtracted feature z_t:

$$z_t = y_t - \overline{y_t} = c + s_t - (c + \overline{s_t}) = s_t - \overline{(s_t)}$$

This shows that a speech mean $\overline{s_t}$ is also subtracted. We can now use z_t as the input feature vector for training and performing recognition, with the effect of the channel transfer function essentially eliminated.

Variations of CMS

Several variations of Cepstral Mean Subtraction have been developed to try and solve some of the problems that arise from using it:

1. Standard CMS: taking $z_t = y_t - \overline{y_t}$ as new input feature, Westphal [19] studied the effect of CMS on several cases (continuous speech, between-word pauses, and silence) assuming static noise. For segmented speech without many pauses, the compensation works well although some noise dependence is introduced. For the pause case, a shift is introduced that is proportional to the channel. In conversational speech there is a greater variance of the pause proportion that will reduce the desired channel compensation.

2. Speech-based Cepstral Mean Substraction (SCMS) [19]: To try to overcome the dependence on the pause proportion, SCMS estimates the mean only on speech frames ($z_t = y_t - m_{\text{spe}}$) using

$$m_{\text{spe}} = \frac{\sum_t \omega_t \cdot y_t}{\sum_t \omega_t} \tag{2.34}$$

where ω_t is the probability $p(speech|y_t)$ or the output of a speech detector (1 for speech, 0 for pause). Experiments suggest that it is only really worthwhile to employ this method in cases where there are relatively long between-word silences. For speech utterances it is therefore easier to strip off the (longer) initial and final silences, and then use standard CMS on the speech. Normal between word pauses are too short to make using SCMS worth the extra overhead.

2.10.2 Spectral Mean Subtraction

CMS deals with channel distortion, which introduces convolutional noise to the signal. A method is also needed to get rid of additive spectral noise, to eliminate any slow time varying signals and DC offsets, and background noise (car engine noise, constant babble, white noise).

There are several variants of spectral subtraction, similar to CMS, that try to compensate for the additive spectral noise [20]. One approach estimates the noise power spectrum, and subtracts this from the entire speech utterance. This approach gives rise to a number of problems. Firstly, because noise is estimated during pauses, the performance of the spectral subtraction system relies upon a robust noise/speech classification system. If a misclassification occurs, this may result in a mis-estimation of the noise model and thus a degradation of the speech estimate. Spectral subtraction may also result in negative power spectrum values, reset to non-negative values in a variety of ways. This results in residual noise often called musical noise.

Continuous spectral subtraction continuously updates an estimate of the average of the long term spectrum (since speech changes rapidly, the average over a long period should closely match the spectrum of the additive noise signal). In an off-line recogniser, the average of an entire utterance can be calculated before recognition if the utterance is not longer than a few seconds. In this case, the compensation method becomes known as Spectral Mean Subtraction (SMS).

SMS [20] simply involves calculating the average (over time) power spectrum by adding the power spectra of all the frames in the utterance, and then dividing by the number of frames. This average power spectrum is then subtracted from each frame. Within this framework occasional negative estimates of the power spectrum occur. To make the estimates consistent, some artificial flooring is required, yielding *musical noise* as previously mentioned. This musical noise is caused by the remaining isolated patches of energy in the time frequency representation.

Much effort has been put into reducing this musical noise. One effective way is smoothing over time of the short-time spectra. This has the contrary effect, however, of introducing echoes.

While reducing the average level of the background noise substantially, plain SMS has been found in the literature to be rather ineffective in improving intelligibility and

quality for broadband background noise, and thus not very effective for improving speech recognition. This has been reaffirmed by the experiments in Section 4.6.

This chapter on theory has provided the necessary background information for describing the implementation of the telephone speech recognition system.

Chapter 3

Telephone speech recognition system

The ultimate goal of our system is to act as an automatic telephone receptionist, prompting the caller into giving the name of the person (within the department) to whom they'd like to speak, and forwarding the call.

Two major parts comprise the system, namely *hardware* necessary for implementation, and the *software* which performs the recognition and controls the flow of events during a call. These two parts will be discussed in detail in this chapter.

3.1 Hardware architecture

The relevant hardware of the implemented system consists of the following elements:

- Intel Pentium 133Mhz CPU
- 96 Mbytes RAM

- Dialogic D/21H telephone interface

The relatively slow[1] CPU speed encouraged careful and intelligent optimisation of the speech recognition software.

The Dialogic model D/21H telephone card enables two telephone lines to be connected to the computer, although only one was ever tested in practice. It interfaces to the PC on the PCI (Peripheral Component Interconnect) bus, providing facilities such as analogue-to-digital and digital-to-analogue conversion of the telephone speech, ring detection, dial-tone detection and DTMF (Dual Tone Multi Frequency signalling, in common use for telephone systems) encoding and decoding.

The telephone interface card was connected to the PABX at the campus of the University of Pretoria. This particular PABX has the facility of being able to forward a call by sending what is known as a hookflash (putting the phone "on-hook" for approximately 100ms, then taking it "off-hook" again) followed by the four digit internal (to the University of Pretoria) telephone number sent via DTMF. Since the recommended maximum rate of DTMF digit transmission is 10 per second (i.e. 100ms per digit), the whole process of forwarding a call takes approximately 500ms.

3.2 Software architecture

The software which performs the sound data processing, feature extraction, model training and level-building search was developed in-house at the University of Pretoria by the Pattern Recognition Group. The software toolkit (named the "Hidden Markov Toolkit for Speech Recognition", HMTSR[2]) was developed in Gnu C++ in the Linux operating system using an extensive object-oriented approach, described in detail later

[1] compared to the leading-edge Intel processors in 1999, 550MHz Intel Pentium III, 5 to 8 times faster. At the beginning of 2001, 1.2Ghz processors were available.

[2] See website `http://www.ee.up.ac.za/ee/pattern_recognition_page/HMTSR/HMTSR-0.2.tgz`

in this chapter.

The telephone speech recognition system was necessarily developed on a Windows NT 4.0 platform, since at the time of development no Linux drivers existed for the Dialogic card[3]. A Windows NT Gnu C++ compatible compiler along with libraries emulating Unix system calls had been developed by Cygnus Solutions, and HMTSR was ported into the Windows environment using this tool, with some modifications. HMTSR thus formed the backend of the telephone speech recogniser. The Dialogic API (Application Programmer's Interface) libraries had to be ported (they were originally written for Microsoft Visual C++) to be compatible with Cygwin with some major modifications relating to the way threads are handled under NT. A console application was written to handle incoming calls (multiple, i.e. 2 for the D/21H card, calls can be handled simultaneously), a thread being spawned for each incoming call. The thread takes care of the conversational interface with the caller by prompting him, and detecting when he has finished speaking. Speech to be recognised is passed to the HMTSR level-building search, and action is taken depending on the transcription of the models found in the search.

3.2.1 Hidden Markov Toolkit for Speech Recognition

The HMTSR toolkit comprises a number of classes (Object Oriented Programming) to facilitate speech processing and hidden Markov modelling of raw waveform data, and these classes are all grouped together as a library. The library provides the following functionality:

- Manipulation of wave data files in several popular file formats.
- Perform feature extraction (pre-emphasis, hamming window, mel-spaced filter banks, mel-cepstra).

[3]Dialogic released Linux drivers for this card in July 2000.

- Implement a finite state network grammar.
- Dynamic multi-dimensional array memory management.
- Dealing with the transcriptions of labeled training data, as well as storing transcriptions of recognised data.
- The hidden Markov model implementation.
- Implementation of Viterbi alignment, expectation-maximization, level-building.
- Code to speed up training time, e.g. cacheing feature vectors.

More detailed descriptions of each of the relevant classes is given in Appendix A.

3.2.2 Feature extraction

The process by which a matrix of cepstral coefficients (vector of coefficients over time) is extracted from a raw sound wave file can be summarised as follows (Figure 3.3). The input data consists of a vector of floating point values (representing the sound wave file, values normalised between 0 and 1) with a corresponding length (number of samples) and a sample rate (8kHz for the telephone speech recognition system). This data is divided into a number of *windows*, determined by two parameters, namely *windowsize* and $framestep$. The *windowsize* determines the number of samples (the unit of the *windowsize* parameter is seconds, so the actual number of samples can only be calculated given the sample rate) used for every cepstral calculation, and the $framestep$ determines the offset from the current position where the next *windowsize* samples begin. The $framestep$ parameter is usually less than *windowsize* so that there is some overlap between adjacent windows. For example, for *windowsize* = 16ms and $framestep$ = 10ms, the first cepstral coefficient vector will be calculated from the first 16ms of speech, and the second vector will encompass 10ms to 26ms of the speech

segment. The total number of windows in a given length of speech data is therefore given by:

$$n_{windows} = \frac{n_{samples} - \frac{windowsamples - framesamples}{2}}{framesamples}$$

where $windowsamples$ and $framesamples$ are calculated from the sample rate as follows:

$$windowsamples = samplerate \cdot windowsize$$

$$framesamples = samplerate \cdot framestep$$

Since a fast Fourier transform can only be calculated for lengths of data that are powers of two, the next power of two, that is larger than $windowsamples$, is determined. This gives the size of the Fourier transform vector.

Preemphasis is performed on each window. Preemphasis is typically a first-order FIR filter used to spectrally flatten the signal (i.e. lessen the effect of spectral tilt) and make it less susceptible to finite precision effects later in the signal processing. The output of the preemphasis network is related to the input $s(n)$ by the difference equation

$$\tilde{s}(n) = s(n) - a \cdot s(n-1), 0.9 \leq a \leq 1.0,$$

a being another system parameter which defaults to the popular value of 0.98.

Each frame (consisting of $windowsamples$ samples) is now *windowed* so as to minimize signal discontinuities at the beginning and end of each frame. A window is used to taper the signal to zero at the beginning and end of each frame. There are a number of functions which achieve this, the most popular one being the Hamming window(Figure 3.1), which has the form

$$w(n) = 0.54 - 0.46 \cdot \cos\left(\frac{2\pi n}{windowsamples - 1}\right), 0 \leq n \leq windowsamples - 1.$$

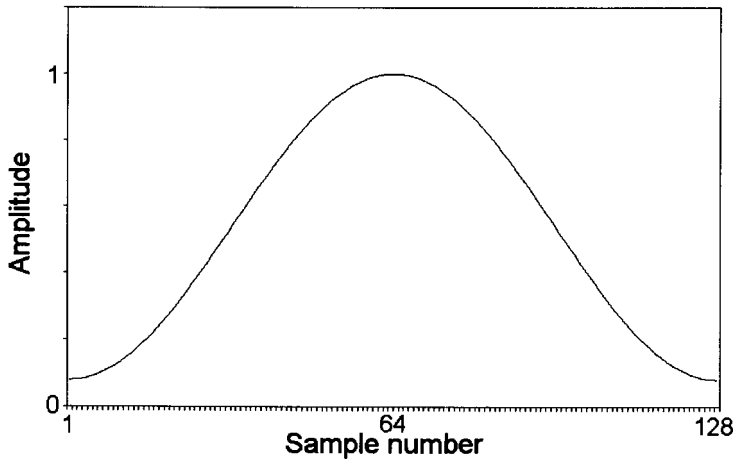

Figure 3.1: Hamming window for 8kHz sampling rate and 16ms window size

The power spectrum of this windowed frame is now calculated to $fftsize$ elements. The DC component of the power spectrum is set to zero, and the power spectrum vector is then passed through a mel frequency spaced triangular filter bank, the centre frequencies of which are determined by

$$filtercentre_i = 700 * \left((1 + \text{samplerate}/1400)^{\frac{i+1}{n_{filters}}}\right) - 700, 0 \leq i < f_{filters}.$$

where $n_{filters}$ is another system parameter. For the default case where samplerate is 8kHz and $n_{filters} = 12$, the centre frequencies are located at

$$120, 261, 427, 621, 848, 1114, 1426, 1791, 2220, 2722, 3310, 4000.$$

Hertz respectively, and the base corners of the triangles as indices into the power spectrum vector are determined by:

$$fc_i = (int)(\frac{filtercentre_i}{samplerate/fftsize} + 0.5)$$

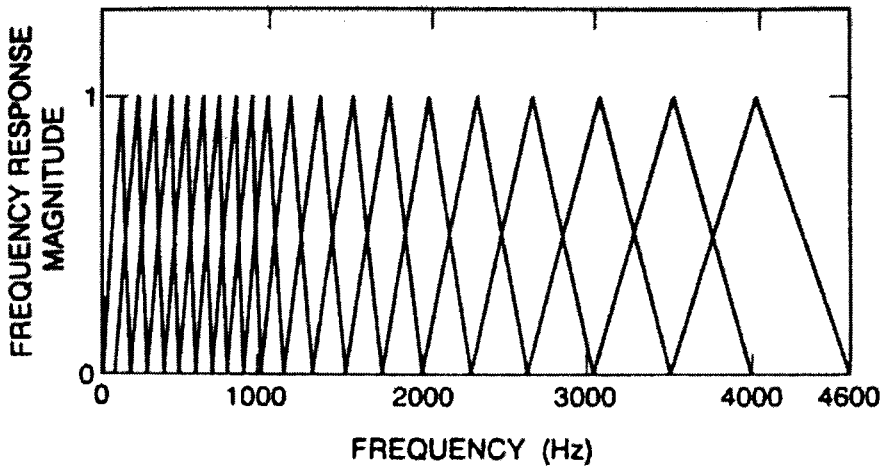

Figure 3.2: A filter-bank design in which each filter has a triangle bandpass frequency response with bandwidth and spacing determined by a constant mel frequency interval, from [7, p. 190].

The energy underneath each triangular filter in the filter bank is summed, and $n_{filters}$ energy values are returned. The *ln* of each of these values is taken, after which a discrete cosine transform is performed, resulting in $melorder$ (another system parameter) mel frequency scaled cepstral coefficients, $mfcc_i, 0 \leq i < melorder$. Liftering is applied to normalise the contribution from each cepstral term with the algorithm:

$$mfcc_i = mfcc_i \cdot i^{0.6}, 1 \leq i < melorder.$$

Temporal derivatives (so-called delta features) are optionally calculated next using polynomial approximation. Depending on the delta order (only first and second order delta parameters are considered), the feature vector doubles or triples in size per frame.

For the optional case where Spectral Mean Subtraction is performed, the power spectrum vectors for the whole utterance are calculated, a mean derived, and the mean

is subtracted with artificial flooring from each power spectrum vector. The artificial flooring employed assumes the following logic: if the mean spectral value is less than the spectral value, the result is $0.1 \cdot spectralvalue$, else $spectralvalue - meanvalue$. Several values (including 0) of this constant were tried, with 0.1 giving the best results.

For the cases of generating the feature cache and model training, Cepstral Mean Subtraction is optionally performed on this feature matrix since only whole words are being considered. In the case where a search on an utterance has to be performed, the initial and ending silences are first trimmed off (clarified further on) before CMS is performed.

After this entire procedure has been followed on a raw wave sound utterance, the feature matrix (cepstral coefficients vs. frame (time)) is ready to be used by the hidden Markov modelling routines. Figure 3.3 summarises the feature extraction process.

3.2.3 Finite State Network grammar

A graphical tool was developed in X-windows using the QT library to facilitate the creation of finite state networks. Network nodes can be dynamically created and moved around, and given properties such as model name, display name. Nodes can be connected to other nodes, or themselves, and flags can be set indicating whether the node is optional, initial (the start of a grammar path) or final (the end of a grammar path). The tool writes to and reads from text files with the same lexical format as used by the `grammar` class.

Many of the errors in the recognition process can be attributed to phrases that cannot possibly be correct, and would immediately be seen as such by a human. In the context of this system, this includes phrases with names and surnames from different people mixed, phrases with repetitions of a word, and phrases that have an incorrect word order. This can be remedied fairly easily by implementing a Finite State Network (FSN) grammar, limiting the search to "possible" combinations. The topology of the

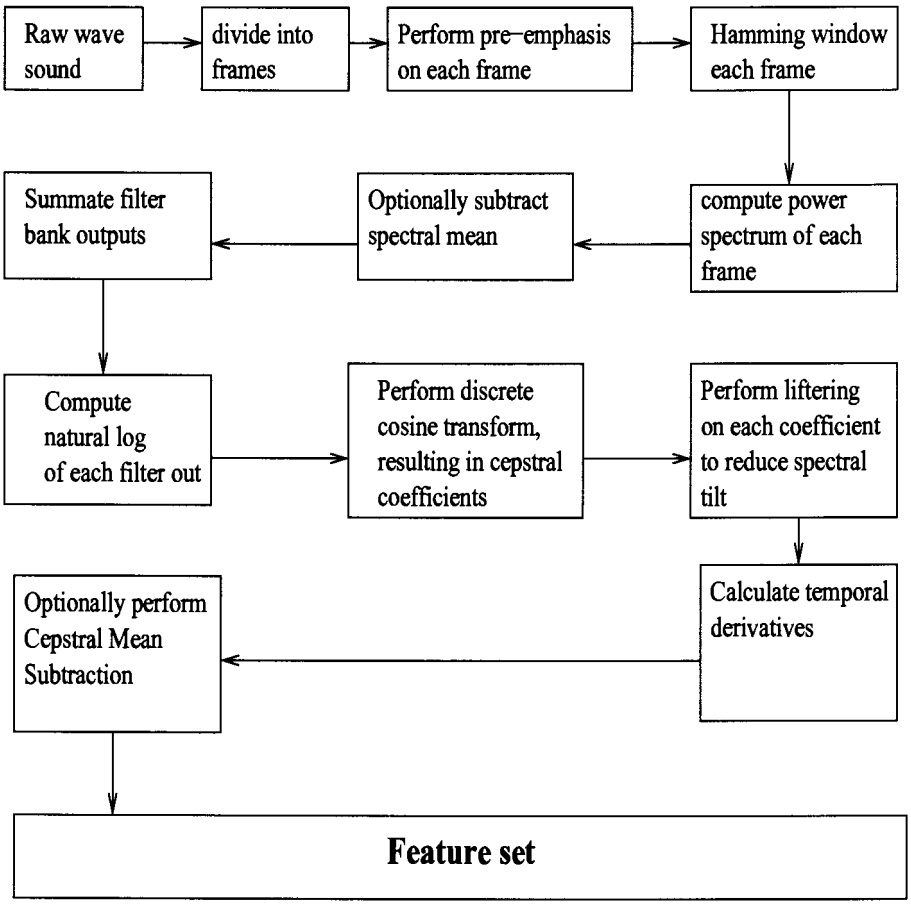

Figure 3.3: Feature extraction process in HMTSR

network implemented can be seen in Figure 3.4.

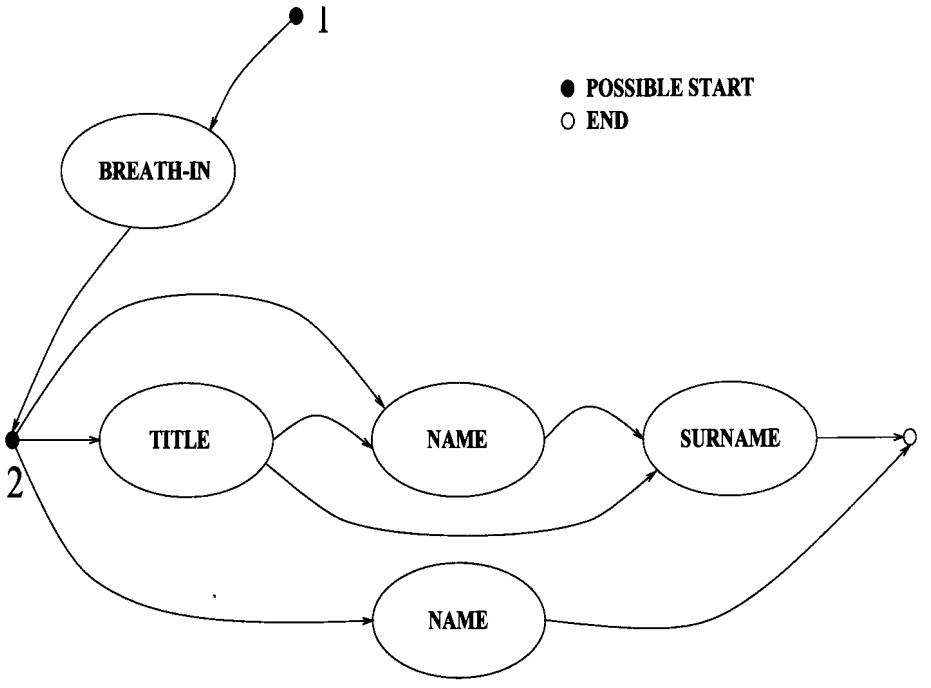

Figure 3.4: Finite state network grammar (`general` grammar)

This network allows for the following possible grammar paths:

- Name
- Name → Surname
- Title → Name → Surname
- Title → Surname

Other optional models are placed in the FSN to improve robustness. An initial silence model is included, followed by a "breath-in" module to compensate for the fact that many people inhale sharply just before uttering a phrase. An optional "breathout" model follows the above mentioned grammar network.

Other possibilities exist, e.g. Title → Name, but since this is very informal and uncommon, it is considered erroneous for the purpose of this system.

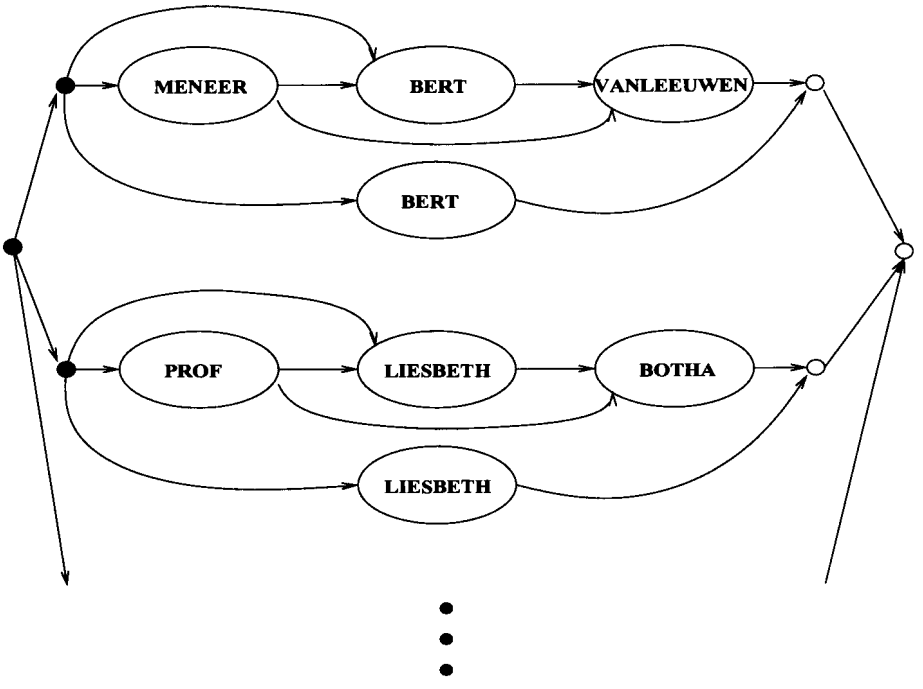

Figure 3.5: Finite state network grammar (`full` grammar)

The `full` grammar network (Figure 3.5) consists of the previously mentioned paths for each callee in parallel, so that names, surnames and titles of different callees cannot be confused.

The FSN grammar is implemented in the forward Viterbi search by adding either $ln(0)$ (approximated by the largest negative number the PC can take) to the log probability for an impossible transition, or $ln(1)$ to the log probability for a grammatically correct transition. This is analogous to multiplying the probability by 1 or 0 if not working in the ln domain. In this way very little overhead is incurred on the search engine, but drammatically improved recognition results can be obtained (see Section 4.4).

3.3 End point detection

End point detection entails finding the ends of an utterance, i.e. finding where in temporal space the utterance starts, and where it ends. This can be done by the recognition engine by incorporating an optional beginning and ending silence model, but this gives rise to some complexities. Accurately detecting the endpoint is important, since incorrect endpoint detection can greatly affect the performance of the recogniser (see Figure 3.6).

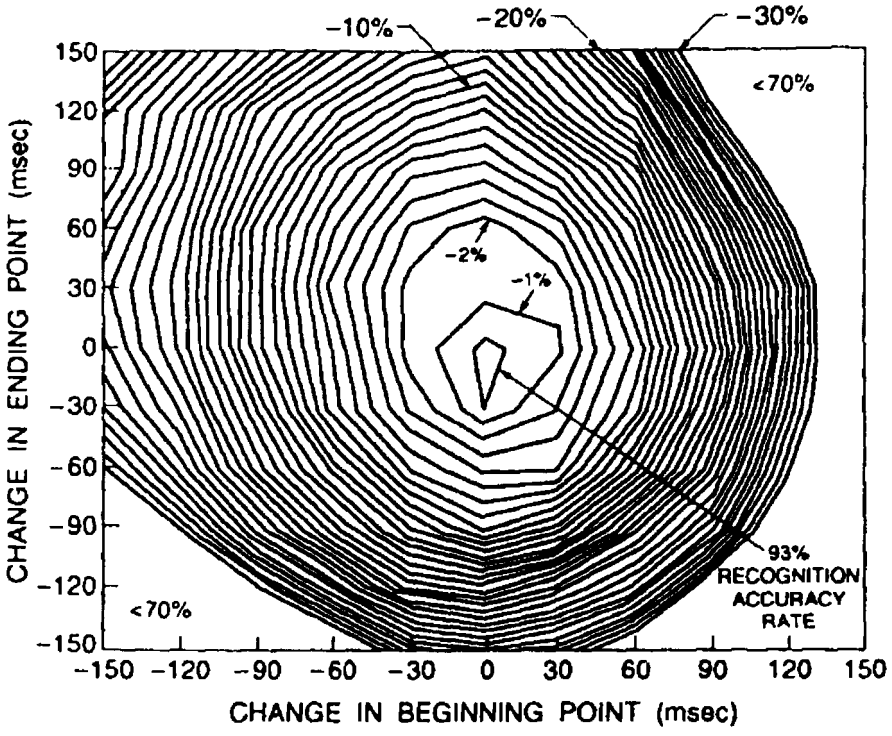

Figure 3.6: Contour of digit recognition accuracy (percent correct) as a function of endpoint perturbation (in ms), from [7, p.144].

The beginning and ending silences' lengths can vary greatly. Since the silence model is a single state model, the state duration modelling will apply to the entire silence. This is not desired, so compensation must be made in the system to allow for HMMs to which duration modelling does not apply. This unnecessary complexity can be avoided by first applying end point detection to the utterance. Furthermore, the Viterbi search

together with the level-building algorithm take linearly longer with longer utterances, and are potentially much slower than an algorithm which only has to find silence. Thus by first removing the silence, large time savings can be made in the recognition process. Optional noise models can still be incorporated in the recognition FSN, to compensate for noise artifacts that may have been missed by the endpoint detection. In this way, a "hybrid" two-stage speech endpoint detection system is implemented.

For speech produced in laboratory circumstances, i.e. carefully articulated and relatively noise free, accurate detection of speech is a simple problem. In practice however, one or more problems make accurate endpoint detection difficult. One particular class of problems is those attributed to the speaker and manner of producing the speech. For example, during articulation, the talker often produces sound artifacts, including lip smacks, heavy breathing, and mouth clicks and pops.

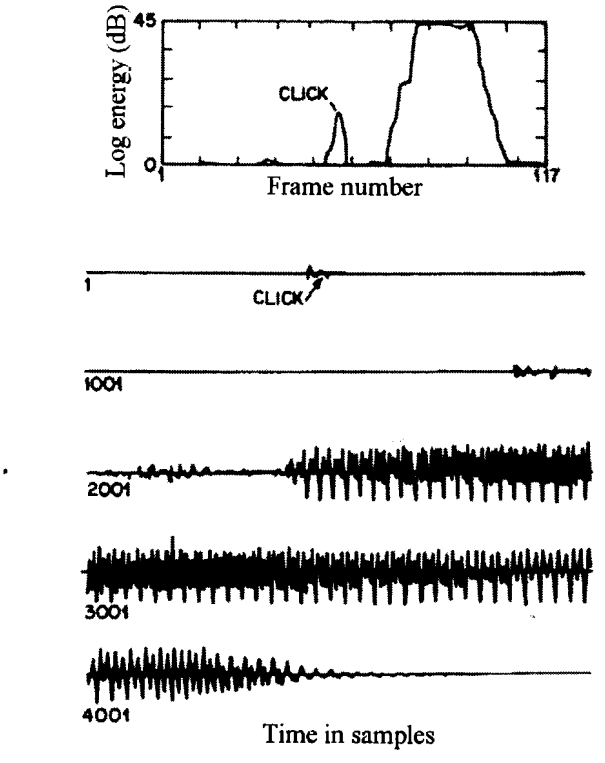

Figure 3.7: Example of a mouth click preceding a spoken word, from [7, p.145]

The energy levels of the artifacts are often comparable to speech energy levels, ruling out using an energy envelope threshold as an accurate means of speech detection.

To detect the silences, the HMM output probability for a silence model is considered with an empirically determined threshold. If the utterance silence output probability estimate falls below this threshold for an empirically determined number of frames $\tau_{initial}$ in a moving average series, the speech is considered to start $\tau_{initial}$ frames back. Similarly, if the silence output probability estimate is above the threshold for τ_{final} frames, the speech is considered to have ended τ_{final} frames ago. By choosing $\tau_{initial}$ and τ_{final} carefully so as to easily include the shortest vocabulary word, yet exclude between word pauses, we can eliminate most mouth clicks and pops and any other short unwanted sounds. The procedure described here was implemented by me in the system.

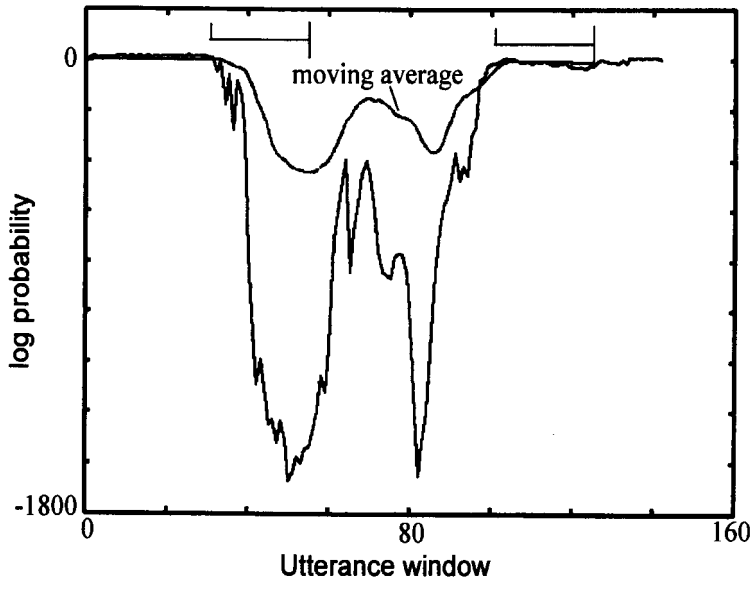

Figure 3.8: Output of the endpoint detector

3.4 Conversational interface

To facilitate more accurate system performance (in terms of the caller being put through to the correct person), a conversational interface is needed to clarify any cases where the recognition process did not generate a sufficiently high confidence measure (see Section 4.8).

The system conversational interface follows the flow diagram which is depicted in Figure 3.9.

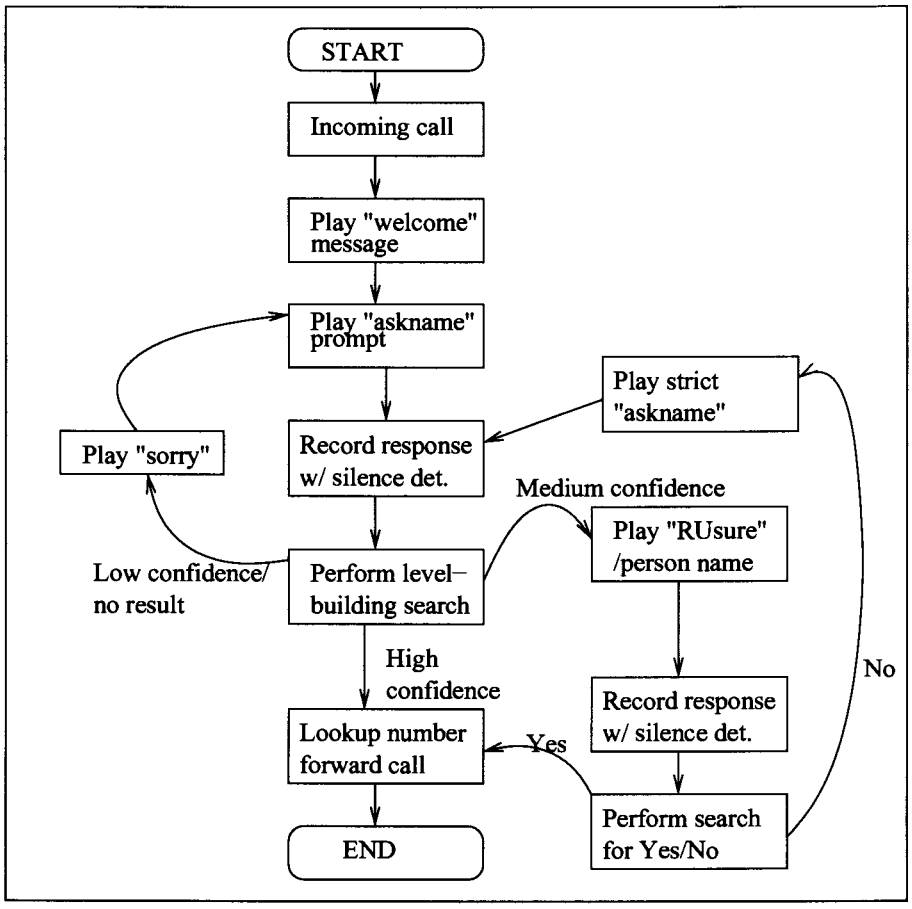

Figure 3.9: Flow diagram of conversational interface

The possible outcomes of this process are:

1. The confidence level in recognising the utterance is below a certain threshold (determined empirically). In this case, the prompt is repeated and the user asked to repeat his answer. This could be due to excessive noise, and invalid response or a speaker dependent problem (breathing noise, mouth pops/clicks, strange accent).

2. The system recognises the utterance incorrectly, and since the confidence level is not sufficiently high, the user is asked for confirmation. Assuming the yes/no recogniser has an extremely high recognition performance[4], the person will either have to repeat his answer (a stricter prompt is played, specifying that the user must say the name and surname of the callee), or his call will be forwarded.

3. The system recognises the phrase correctly, but since the confidence level is not sufficiently high, the user is asked for confirmation, as in the previous case.

4. The system recognises the phrase incorrectly, and the caller is forwarded to the wrong number as the confidence measure is sufficiently high.

5. The system recognises the phrase correctly, and the caller is forwarded to the correct person. This is obviously the case the system strives to achieve most often.

3.5 Dealing with extraneous speech

The conversational interface strives to prompt callers into giving only answers in the correct grammatical format (e.g. Name Surname, or Yes/No), however the cases of extraneous speech need to be taken into account. Some people respond to a prompt like "*Please say with whom you'd like to speak*" by replying: "*I'd like to speak to so-and-so please*," even though they realise its a machine they're speaking to. The system under normal conditions would expect a reply to be just the name[5].

[4]High possibility since only two vocabulary words need to be distinguished.

[5]In any of the generally accepted name lexical formats.

To attempt to deal with this, a grammar FSN can be constructed using a so-called *garbage* hidden Markov model to try and cater for any out-of-vocabulary (OOV) words. The idea of the garbage model is to give a higher score in the Viterbi search than any of the in-vocabulary models for an OOV word, whilst giving a lower score for an in-vocabulary word than the correct corresponding model [21].

Several HMM structures for a garbage model have been tried in the literature, the best result being achieved for a model with a single state and one or more mixtures.

To train this garbage model, all the vocabulary words are used together, based on the assumption that there is enough variance between these words to make a good garbage model.

Dealing with this extraneous speech has not been implemented in the online (real-time recognition) version of the system. It was left out for performance considerations, and it has been found in practice that the prompts are clear enough that the caller usually only utters phrases in one of the correct grammar patterns. Off-line experiments have been done to test the effectiveness of including a garbage model in the system (see Section 4.9). This concludes the description of the telephone speech recognition system that has been implemented. The next chapter details some experiments to test the efficiency of various components of the system.

Chapter 4

Experiments

To test the system performance, and determine the sensitivity to methods used and system parameter variations, experiments were performed. The experiments were done with the system in off-line mode, using data previously recorded from the telephone line. This was done so that all recognition results from the different methods and parameter settings reflect the same data, allowing fair comparisons to be made.

4.1 Experimental data

4.1.1 Data sets

Since the system uses the whole word recognition approach, only a limited number of words can be recognised by the system. The words which the system recognises can be seen in Table 4.1.

The name models (each first name is one word, and double surnames are also treated as a single word model) that the system recognises can be seen in Table 4.2.

Table 4.3 summarizes this target data.

Total distinct words	**42**
Total num. people	**18**
Titles	**5**

Table 4.3: Target data set

4.1.2 Data recording

The telephone system was initially set up to play a prompt, and then record an utterance without performing any kind of recognition or giving any feedback to the user. This was done so as to be able to record enough data to "bootstrap" the hmm models with some seed data to be able to perform recognition. Once sufficient data (approximately eight words by different speakers per model was determined to be good enough) had been recorded in this manner, the system was trained and put into "live" operation, where callers' utterances were recognised (and recorded), and the conversational interface guided them to the point where their call was forwarded to the callee.

The system was left to run in this manner for a period of time, until sufficient data (male and female) had been collected to perform meaningful experiments. The target for data collection was set so that at least 16 spoken words per model were recorded. Table 4.4 summarizes the data set.

Due to the nature of the data recording technique, it is difficult to establish exactly how many different speakers there are in the data set, but it is estimated that there are approximately 25 to 35 different speakers.

Total male utterances	**179**
Total female utterances	**183**
Num. labeled words	**950**

Table 4.4: Training data set

4.1.3 Experimental protocol

A "leave-one-out" experimental setup was used throughout the test. A total of five sets of training/testing data was used. Each training set consists of $\frac{4}{5}$ of all the data, with the remaining $\frac{1}{5}$ used for testing. In this way all the data can be used for testing and training, but the test data never appears in the training set. An average performance value is obtained over all five sets for each experiment to give the final result.

4.2 Adjustable system parameters

The system parameter set which can be optimised for maximum recognition performance was discussed in detail in the previous chapter, and can be summarised as follows:

- windowsize: Length of raw sound (in seconds) used to calculate each feature frame (in the order of 10ms to 30ms).
- framestep: Length of raw sound (in seconds) to step. The difference $windowsize - framestep$ determines the overlap between adjacent feature vectors (in the order of 5ms to 30ms).
- MelOrder: The number of mel-scaled cepstral coefficients to calculate per frame (in the order of 6 to 12).

- $n_{filters}$: Number of filters in the mel-scale filter bank. This value defaults to $1.5 \cdot \frac{samplerate}{1000}$.
- a: The preemphasis factor, which defaults to 0.98.
- EM_{iter}: Number of expectation-maximisation iterations to perform. This value defaults to 10, as the trained mean and variance values for the HMM mixtures seem to stabilise after approximately 8 iterations.
- $Viterbi_{iter}$: Number of iterations for Viterbi alignment (defaults to 10).
- Delta order [9]: The order of the temporal cepstral derivative. May take on the values 0 (no delta features), 1 or 2. For example, for a MelOrder of 8 and delta of 2, the feature vector will be of length 24 per frame.
- n_{states}: The number of states to use in the hidden Markov word models. This can actually be specified per model, so that different models have different number of states to try and account for the varying number of sounds in words.
- $n_{mixtures}$: The number of Gaussian mixtures to use per state in each model. The number of mixtures can also be independently set for each state in each model if desired.
- grammar: The grammar FSN to use, if any.
- performDuration: Boolean flag to choose between using traditional state transition matrices resulting in implicit geometric state durations, or using *lngamma* pdf based state duration modelling (see Section 2.9).
- performCMS: Boolean flag indicating whether or not to perform Cepstral Mean Subtraction on the feature matrix.
- performSMS: Boolean flag indicating whether or not to perform Spectral Mean Subtraction on the power spectrum.

4.2.1 Optimisation experiments for system parameters

Experimental results are shown where these parameters are varied to find the best system performance. Since the parameter variations can influence each other (e.g. if the *windowsize* is smaller, you might want to consider increasing n_{states} since more features will be calculated per word), the parameters are varied individually, leading to a large number of experiments, and extensive computation.

Some of the parameter variations were found not to have much influence on each other, and these were kept at a fixed value during the iteration procedure, to try and keep the computation realistic. The values in Table 4.5 were used.

performCMS	on
performSMS	off
Delta	2
pre-emph	0.98
performDuration	on
grammar	full
EM_{iter}	10
$n_{filters}$	12
$Viterbi_{iter}$	10

Table 4.5: Baseline fixed parameters for optimisation experiment

The parameters in Table 4.6 were varied in the specified ranges. These ranges were obtained through coarse initial experimentation to determine in what range the optimum point occurs.

The *framestep* and *windowsize* parameters were varied together, since it was previously found that the best results are obtained if *framestep* is equal to slightly more than half of *windowsize*. Since *framestep* is the increment for each successive start

$n_{mixtures}$	1,2,3
n_{states}	8,9,10,11,12
windowsize	8,12,16,20,30ms
framestep	5,7,10,12,19ms
MelOrder	6,8,10,12

Table 4.6: Variable parameters for optimisation experiment

of *windowsize* samples, this means that there is always an overlap with the previous frame.

The best overall person recognition rate attained was **95.68%**, with the parameter values as given in Table 4.7.

$n_{mixtures}$	1
n_{states}	10
windowsize	20ms
framestep	12ms
MelOrder	8

Table 4.7: Baseline "best" parameter set

The worst overall person recognition rate attained with these parameter variations was **73.02%**, with the parameter values as in Table 4.8.

The fact that the feature vectors will be large ($MelOrder = 12$) and that there are three mixtures per model to train, implies that there is probably insufficient training data available to successfully train the models, providing a possible explanation for the poor results with these parameters. The small *windowsize* also means that at an 8kHz sampling rate, there are only 64 samples per window, so the FFT results in frequency components with frequency spacings of 125Hz. This means that the resolution for the

$n_{mixtures}$	3
n_{states}	11
windowsize	8ms
framestep	5ms
MelOrder	12

Table 4.8: Baseline "worst" parameter set

power spectrum calculated is not very high and therefore poor results are obtained.

If we take the average recognition results over each parameter variation (e.g. average of all the results where $MelOrder = 8$), we get the following (figures 4.1, 4.2, 4.3 and 4.4):

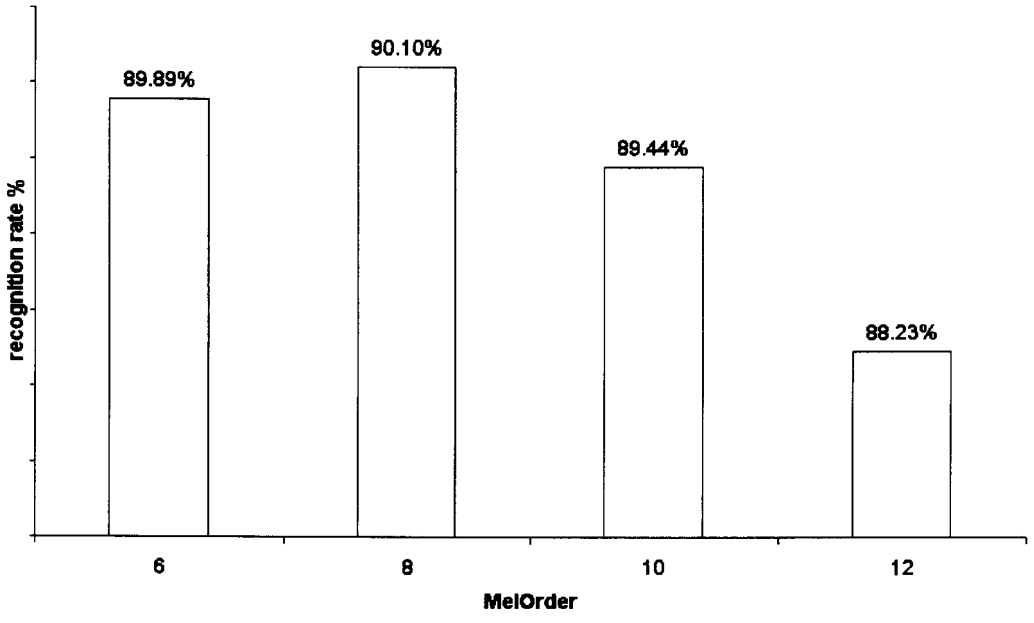

Figure 4.1: Average recognition rates for $MelOrder$ variations

Although the parameters are interdependent, there is still a very close correlation between the overall best and worst parameters sets, and the results of the above average parameter variation scores. These "best" and "worst" parameter sets are used as a

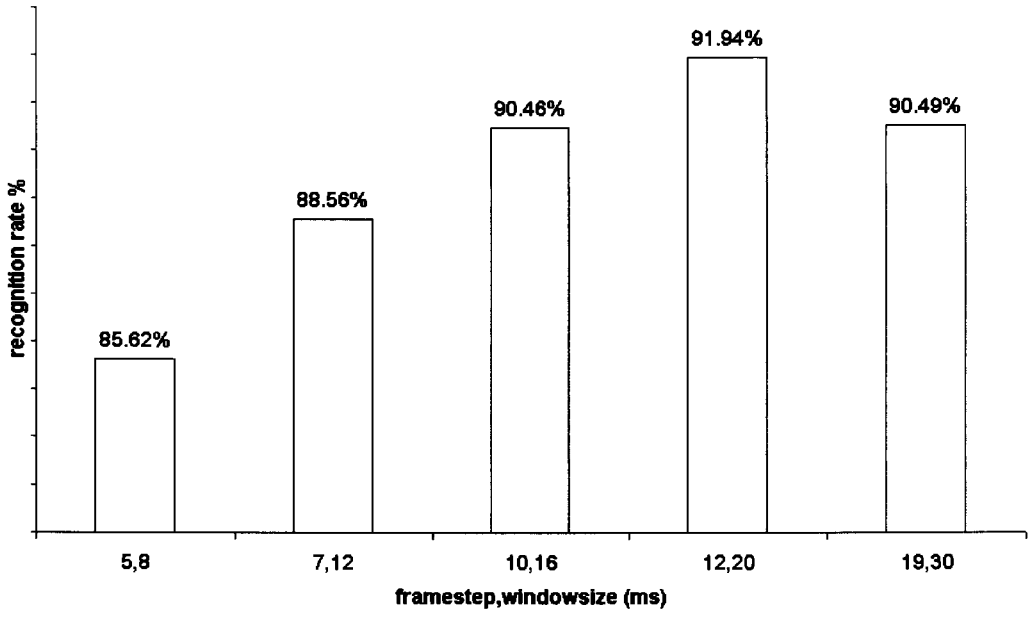

Figure 4.2: Average recognition rates for $framestep, windowsize$ variations

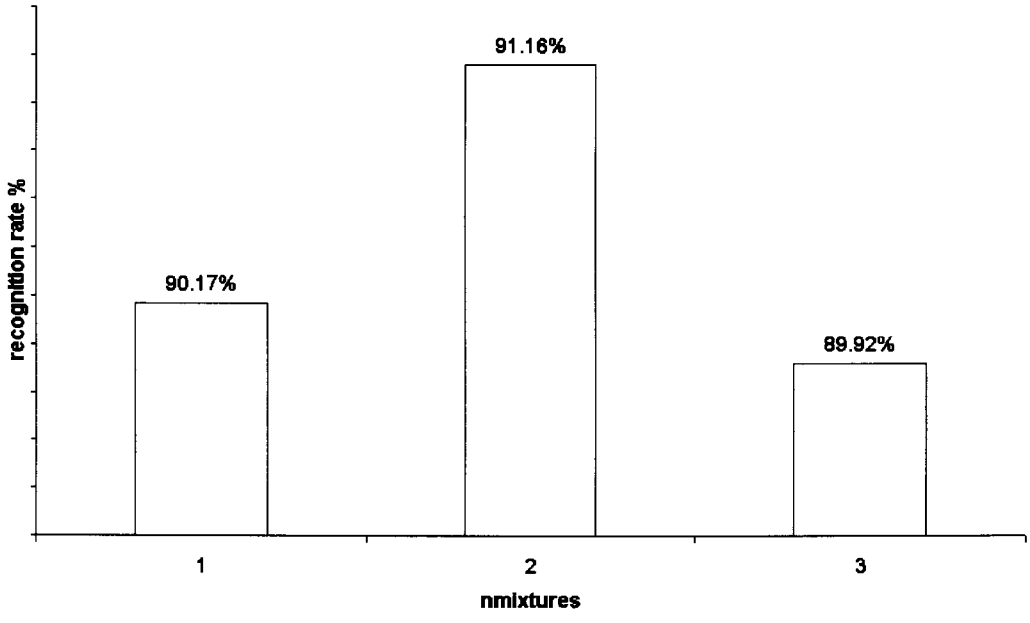

Figure 4.3: Average recognition rates for $n_{mixtures}$ variations

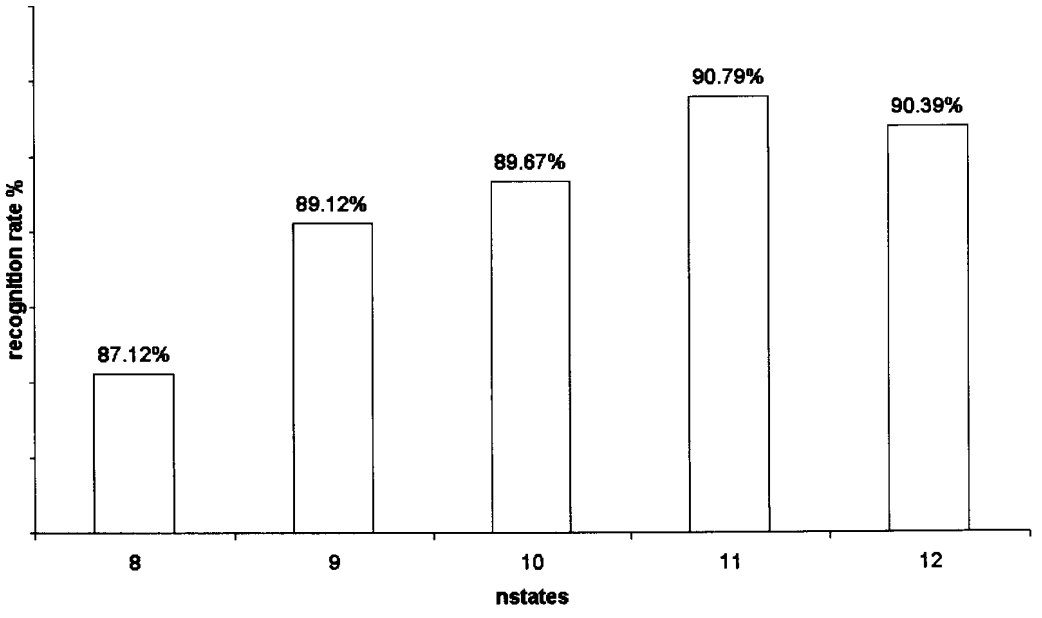

Figure 4.4: Average recognition rates for n_{states} variations

baseline for the rest of the parameter and method variation experiments, each effect (e.g. using CMS or not) being measured on both the best and worst sets to see the effect.

It can also be seen that the recognition results do not seem to vary much with the number of states, as long as there are enough states to model at least the minimum number of distinct sounds in each word. Another experiment was performed where the word models were categorised into *long* and *short* words, with the long words having 50% more states than the short words. This also made no significant difference to the recognition results, but when the experiment was repeated without duration modeling, it became clear that a difference could be observed. Without duration modeling, the number of states provides a crude lower duration limit due to the strict left-right nature of the Bakis models, leading to a noticeable performance increase. With less states, it is possible (in the absence of duration modeling) for the model to be in one particular state for only one frame, but if there are more states than distinct sounds in the word, there is a minimum duration time that will be spent in each sound model.

Experiments were also performed where the number of mixtures per state was varied, i.e. instead of having 2 mixtures per state for each state in the model, different variations were tried. The rationale behind this was that the initial and final states are more likely to have problems with breath-in, breath-out or other noises, as well as simply being prone to having the data affected by incorrect end-point detection (see Section 3.3). The following variations were tried:

- one mixture for initial and final state, 2 mixtures for the states in between, and
- two mixtures for the initial and final states, 1 mixture for the states in between.

No significant differences in recognition performance were observed with either the "best" or "worst" parameter sets previously determined, so this avenue of investigation was not pursued further.

4.3 Effect of temporal derivative

To determine the effect of the temporal derivative on the recognition performance, the system parameters in Table 4.5 were used, with $Delta = 0$ and $Delta = 1$. This parameter set was used with the sets in Tables 4.7 and 4.8 to give the results as in Table 4.9.

From this we can conclude that using a first or second order temporal derivative significantly reduces the recognition error rate (by 67%) for the "best" parameter case. The first order temporal derivative slightly improves the recognition rate for the "worst" parameter set, but the second order temporal derivative increases the error rate above what it was without using it at all. Therefore, we can use $Delta = 1$ for good performance and less computation.

best param ($Delta = 0$)	86.65%
best param ($Delta = 1$)	95.56%
best param ($Delta = 2$)	95.68%
worst param ($Delta = 0$)	77.08%
worst param ($Delta = 1$)	83.39%
worst param ($Delta = 2$)	73.02%

Table 4.9: Effect of temporal derivative

4.4 Effect of grammar

To determine the effect the grammar FSN has on recognition performance, experiments were set up with less restrictive FSN grammars. Two different grammars were used, one (`general`) which followed the pattern as shown in Figure 3.4, thereby allowing unknown names to be found (i.e. different titles, names and surnames may be mixed), and a second grammar (`allwords`) which allowed any number of words (titles, names and surnames) in any order. The recognition experiment was repeated using again the "best" and "worst" parameter sets, and comparing these to the results with the `full` grammar (see Table 4.10).

param set	full	general	allwords
best	95.68%	77.14%	19.18%
worst	73.02%	52.08%	9.92%

Table 4.10: Effect of grammar

From these results it is clear that the incorporation of a restrictive FSN grammar dramatically improves the recognition rate. As expected, on inspection of the error phrases for the `general` grammar, the additional (compared to the `full` grammar) errors were caused by the emergence of unknown names (e.g. "mevrou renier van-

leeuwen"). Closer inspection of the error cases for the `allwords` grammar revealed that the majority of the additional (compared to the `general` grammar) erroneous phrases did in fact contain the correct phrase, but extra words (usually short words) were found at the beginning or end of the phrase (e.g. "prof prof botha"). As the FSN grammar becomes less constrained, the level-building search takes longer to execute, since more possibilities arise.

4.5 Effect of Cepstral Mean Subtraction

To determine the effect of CMS on the recognition performance, the system parameters in Table 4.5 were used, with $PerformCMS = 0$. This parameter set was used with the sets in Tables 4.7 and 4.8 to give the results in Table 4.11.

best param ($PerformCMS = 0$)	90.54%
best param ($PerformCMS = 1$)	95.68%
worst param ($PerformCMS = 0$)	69.1%
worst param ($PerformCMS = 1$)	73.02%

Table 4.11: Effect of CMS

CMS leads to a 54% reduction in errors for the "best" parameter set, a significant improvement. The telephone channel transfer function is highly variable, since phone calls go through at least two A/D and D/A conversions, as well as variable lengths of copper wire, or they could even be made from a cellular phone, where codecs such as GSM influence the overall channel transfer function. Using CMS to compensate for this is a very successful technique. The error rate is also slightly improved for the "worst" parameter set, which can be expected, since CMS does not increase the feature vector size, but merely attempts to improve the features by compensating for channel effects.

4.6 Effect of Spectral Mean Subtraction

To determine the effect of SMS on the recognition performance, the system parameters in Table 4.5 were used, with $PerformSMS = 1$. Since it is possible that CMS can have an effect on the result, the tests were also run with $PerformCMS = 0$. This parameter set was used with the sets in Tables 4.7 and 4.8 to give the results in Table 4.12.

best param ($PerformSMS = 1$)	90.65%
best param ($PerformSMS = 0$)	95.68%
worst param ($PerformSMS = 1$)	72.58%
worst param ($PerformSMS = 0$)	73.02%
best param ($PerformSMS = 1, CMS = 0$)	87.18%
best param ($PerformSMS = 0, CMS = 0$)	90.54%
worst param ($PerformSMS = 1, CMS = 0$)	72.06%
worst param ($PerformSMS = 0, CMS = 0$)	69.1%

Table 4.12: Effect of SMS

SMS always appears to diminish performance, except in the "worst" parameter case without CMS, where performance is only slightly increased. Subtracting the mean of the spectrum appears to have more of a detrimental effect on recognition performance than a positive effect. This could be due to the "musical noise" mentioned in Section 2.10.2, and the fact that most of the utterances do not exhibit audible constant additive noise, which SMS was developed to counter. SMS does not appear to be a very effective method for improving speech recognition in this system. Therefore, we use $CMS = 1$ and $SMS = 0$ for best performance (95.7%).

4.7 Effect of $ln(gamma)$ based state duration modeling

To determine the effect of duration modeling on the recognition performance, the system parameters in Table 4.5 were used, with $PerformDuration = off$. This parameter set was used with the sets in Tables 4.7 and 4.8 to give the results in Table 4.13.

best param ($PerformDuration = off$)	93.22%
best param ($PerformDuration = on$)	95.68%
worst param ($PerformDuration = off$)	80.8%
worst param ($PerformDuration = on$)	73.02%

Table 4.13: Effect of duration modeling

As can be seen by these results, duration modeling improves the recognition rate slightly for the "best" parameter set (error rate reduced by 36%), but significantly degrades performance for the "worst" parameter set. This could be due to the fact that since there are more states to train per model, there is less data available to train each state, leading to inaccurate estimates of the duration parameters. Therefore, we use $PerformDuration = on$ for best performance.

4.8 Normalisation of Confidence Measure

The output of the level-building search gives a $ln(probability)$ score. Since this score is strongly dependent on utterance length as well as a number of other factors, these scores cannot be used as an absolute or even relative point of reference for deciding whether the best utterance given by the level-building search was correctly recognised or not.

In an attempt to alleviate this problem, an FSN grammar consisting only of the garbage model was created, with the garbage model able to appear a number of times to make up an utterance. Once a search has been run on an utterance using the normal grammar FSN, another search can be done using this garbage FSN. The results relative to each other do show a significant correspondence with regard to recognition accuracy. This method can be used to generate a meaningful confidence measure by using the rules given in Table 4.14. The confidence measure can in turn be used by the conversational interface to decide the path of events to follow.

confidence	**decision**
1. $utterance > 10 \cdot garbage$	accept
2. $garbage < utterance < 10 \cdot garbage$	confirm
3. $utterance < garbage$	reject

Table 4.14: Confidence measure rules

Using these rules as a confidence measure, the classification of recognition results obtained for the "best" and "worst" parameter sets can be seen in Table 4.15. These results consider the case where a confirmation is necessary (i.e. case 2 in Table 4.14), to be the same as an incorrect decision. Of course, the rules can be altered to change the sensitivity of the system as desired, but ideally there should be more *false rejections* than *false acceptances*, since a *false acceptance* is far worse than a *false rejection.*

The sum of *false rejections* and *correct acceptance* gives identical results to the previous "best" and "worst" recognition rates (i.e. 95.68% and 73.02%), as can be expected. These results show that this simple method does give a reliable confidence measure.

best: false rejection	5.2%
best: false acceptance	2.8%
best: correct rejection	1.7%
best: correct acceptance	90.3%
worst: false rejection	1.7%
worst: false acceptance	21.5%
worst: correct rejection	5.2%
worst: correct acceptance	71.6%

Table 4.15: Classification of recognition results

4.9 Experiments with extraneous speech

Off-line experiments were performed with speech data that contained extraneous speech. A garbage model was trained to compensate for words not in the normal vocabulary by using all **950** labeled words re-labeled as *garbage*, since the overall variance between words is very high. The HMM for the garbage model consisted of one state and one mixture (this was found to give the best results), and a new grammar FSN (`extraneous`) was created which added multiple optional garbage models to the start and end of the `full` grammar for the level-building search.

Some examples of the extraneous speech used were:

- Ek wil met *meneer bert vanleeuwen* praat asseblief ("I would like to speak to *mister bert vanleeuwen* please").
- Sit my deur na *janus brink* nou dadelik ("Put me through to *janus brink* immediately").
- *meneer schoeman* asseblief ("*mister schoeman* please").

number of utterances	36
person recognition	88.89%

Table 4.16: Extraneous speech recognition results

Experiments were run using the "best" parameter set, giving the results in Table 4.16.

It can be seen that the recognition rate is still relatively high, but since the grammar is much less constrained and the overall utterance length longer, the level-building search takes much longer (in the order of 1 second on a Pentium III 550Mhz processor). On closer inspection of the four error cases, one was found to have somebody laughing loudly in the background during the crucial part of the utterance, and two more had only the first name (i.e. no title or surname) in the utterance, in between extraneous speech, leading to a greater chance of failure. This indicates (as can be expected) that there is a higher chance of successfully recognizing a name within extraneous speech when more of the name (e.g. title or surname) is specified.

4.9.1 Case study

One of the utterances ("Ek wil met *meneer bert vanleeuwen* praat asseblief") was used to provide an illustration of the working of the level-building algorithm and Viterbi alignment.

The utterance length was 2.18 seconds, giving 182 frames of features in total (with a framestep of 12ms). The resultant final match consisted of six garbage models, followed by *meneer bert vanleeuwen* followed by more garbage models, so that *meneer*, *bert* and *vanleeuwen* were found on levels six, seven and eight of the level-building search (the first level is marked zero). The transcription of the result showed that *meneer* went from frame 33 to frame 59, *bert* went up to frame 80 and *vanleeuwen* carried on till frame 115, which matches very well with the positions in the wave file if converted to

time.

Figure 4.5 shows the outputs of the $\delta_t(i)$ function (the data was modified slightly prior to plotting by replacing the $log(zero)$ values (represented by $-9.9999998 \cdot 10^{10}$) with the lowest probability number in the rest of the search) during Viterbi alignment of the aforementioned HMMs on levels six, seven and eight (see also Section 2.7 and Figure 2.4). Notice that *vanleeuwen* has 10 states, and the other two models have only 6 states (this experiment was still run with the long/short word models mentioned in Section 4.2.1), and that states 2 to N have probability $log(zero)$ in the first $N-1$ frames due to the left-right nature of the Bakis type HMMs.

4.10 Effect of gender on recognition performance

To see how much effect the gender of the speaker has on classification performance, the female data in the training sets were relabelled so that the HMM model names were all uppercase letters (this was done for backward-compatibility; to do a genderless search one only has to perform case-insensitive comparisons between the recognised phrase and the transcription). This of course also has the effect of reducing the training data for male and female models compared to the genderless model, so one can expect reduced recognition performance if the same parameter sets are used. A new optimum parameter set could be calculated, but ideally the amount of training data should be increased. The latter is not an easy option however, since many more utterances need to be recorded and painstakingly labeled.

The FSN grammar was expanded to include all female model paths (identical except for the model name case difference), so the level-building search on this experiment set takes twice as long as normal, but the training time remains about the same, since there are now twice as many HMMs, but each with (on average) half as much training data.

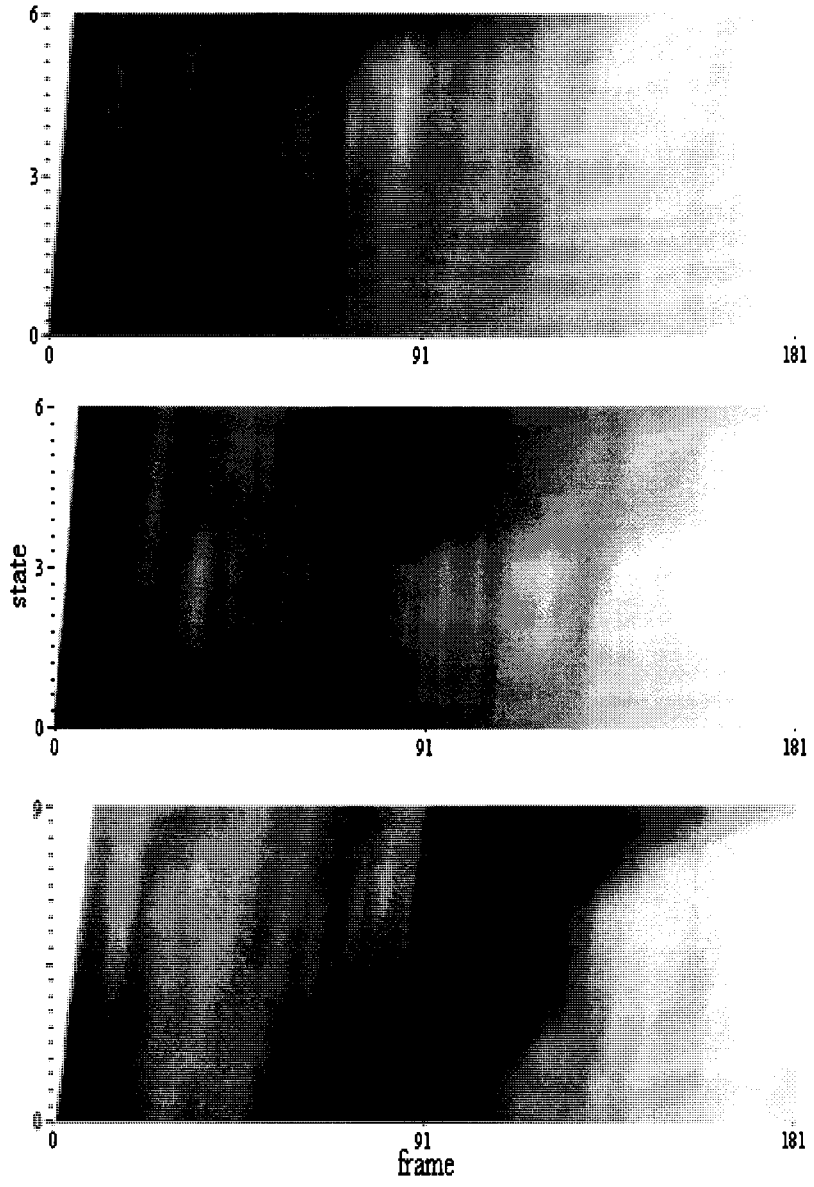

Figure 4.5: $\delta_t(i)$ for HMMs *meneer* (top), *bert* (middle) and *vanleeuwen* (bottom) on levels six, seven and eight of a level-building search with extraneous speech

parameter set	recognition rate	male class. accuracy	female class. accuracy
best param	85.29%	72.62%	93.64%
worst param	41.28%	93.85%	91.78%

Table 4.17: Effect of gender

The results for overall recognition accuracy as well as gender classification accuracy were determined, and can be seen in Table 4.17.

Since the number of male and female speakers were almost the same (179 vs. 183), the classification results were relatively high (much greater than 50%, which would be as good as chance), showing that the chosen features for recognition are not entirely gender independent, and therefore probably not entirely speaker independent either. The reason the cepstral coefficients were chosen as features was that in the literature they were found to have a large degree of speaker independence. The fact that the system works as well as it does for speakers that are not in the training set at all shows that there is truth in this, but obviously there is still a degree of speaker dependence. The leading edge dictation software available today still requires a large amount of training for a specific speaker, and thereafter the recognition rates are very speaker dependent. More research needs to be done on finding effective speaker independent features.

4.11 Real-time performance

The telephone speech recognition system was implemented on a PC equipped with an Intel Pentium 133MHz processor. It was found that the entire process of feature extraction, performing the level-building search, deciding whether this was a good enough match and subsequently dialing the call to forward it to the correct person took less than 1 second for an average utterance length of over 2 seconds, quick enough

for a "real-time" response.

Chapter 5

Summary and conclusion

5.1 Summary of work and results

The main goal of this dissertation was to implement a telephone speech recognition system (with the specific task of a telephone auto-attendant) using state-of-the-art algorithms, and to perform experiments with the system to determine performance and compare some different techniques to enhance performance.

The hardware solution consisted of a Pentium 133MHz class computer using the Microsoft Windows NT operating system, with a Dialogic D21/H telephone interface card.

A literature study was undertaken to determine which speech recognition algorithm to use, and find out what techniques have been developed to enhance performance. Based on this, a decision was made to use hidden Markov models to represent whole words in continuous speech.

A speech toolkit (HMTSR) was developed by our research group during the course of this dissertation, and a number of enhancements and modifications specific to this

project were made. This toolkit was ported to the Windows operating system, since initially there was no Dialogic support for the Linux operating system. The toolkit was integrated with the main state machine controlling the conversational interface, and the Dialogic API was utilised to control the D21/H card so that waveform speech data could be recorded and played back as desired.

Almost one thousand words in total were recorded and manually labeled to form the training set. This training set could then be used as a basis for performing experiments to compare different techniques, and assess system performance.

The system in operational on-line mode could then recognise spoken names with a confidence measure, and forward the telephone call to the desired person's telephone number.

Based on the experimental results in Chapter 4, the overall best person recognition performance attained off-line was **95.68%**, using the parameters as specified in Tables 4.5 and 4.7. The on-line system performed rapidly enough even on slow hardware to attain "real-time" recognition.

5.2 Goals achieved

The contributions which this dissertation set out to make were achieved:

- The process of implementing a speech recognition system from first principles was detailed.
- It was shown how low quality telephone speech data can be successfully used in a speech recognition system.
- The system performance sensitivity with regard to different implementation techniques and system parameters was shown.

- The system is the first of its kind known to be developed in the Afrikaans language.

Furthermore, the results of this work were also published and presented at various conferences (PRASA[1] 1998 [22], PRASA 1999 [23] and AfriCon 1999 [24]).

5.3 Future work

A feature of the current system is the fact that it uses word-based recognition models rather than phonemic models. This implies that it is labour intensive to add more callees to the system: Each word in the new callee's name needs to be recorded by at least ten different speakers, all the words carefully labeled in the `.wav` data files, new HMMs trained and added to the system, as well as adding the new name to the full grammar FSN.

To improve upon this restriction would mean using a sub-word unit based recognition system. The commercially available systems use the latter approach, and are more flexible in that they are not word-based, but rather based on the recognition of sub-word units, so that any vocabulary of words can be constructed using the sub-word units. However, the word-based approach is much less computationally intensive in terms of searches, and requires far less training data to implement each individual word model. A sub-word unit system requires massive amounts of carefully labeled data to achieve sufficient accuracy, and these data sets come at a large price. The search performed by sub-word unit systems also requires that words need to be found in strings of sub-word units, as well as only allowing the correct words in the grammar. The effective grammar models used thus need to be much more complex.

Using the KLT transform matrix to optimally encode temporal information (see section 2.3.1) can also be considered for future work, since it improves recognition accuracy

[1] Pattern Recognition Association of South Africa

over using standard first and second order temporal derivatives, with little or no additional computational cost. This research by Milner [9] looks promising, but was discovered only after the experiments had been completed, so was not investigated as part of this dissertation.

5.4 Conclusion

Using whole word hidden Markov modeling for telephone speech recognition has given very good recognition results. The use of a finite state grammar network has proven particularly helpful. The inclusion of the temporal derivative in the feature set, noise compensation by cepstral mean substraction and duration modeling have all helped to reduce the error rate. The ability to discern the accuracy of a recognition search was essential for the inclusion of a conversational interface, and the initial experiments with extraneous speech look very promising. The result from the gender separation experiment indicates that there is still a degree of speaker dependence in the feature set, but despite this the overall recognition performance of the system is still very high with speakers not in the training set.

A good telephone auto attendant can be on duty 24/7/52, handling one or a dozen incoming lines with ease - always patient, always doing its best to get the caller to the right extension - for a fraction of the cost of human staffing. An auto attendant is such a simple idea, yet it's taken so long to reach fruition. There are currently only a handful of companies providing commercial auto attendants, but the market is now ready to accept them, and technology is now ready to provide reliable auto attendants.

Appendix A

HMTSR detailed description

The Hidden Markov Toolkit for Speech Recognition[1] developed by the Pattern Recognition group at the University of Pretoria is written in C++ and makes extensive use of object oriented programming techniques. The following classes form the core of HMTSR:

`cache`: Facilitates cacheing the extracted feature vectors from the raw sound files to speed up experiments and model training.

`cluster`: Implementation of the K-segmental means algorithm to cluster the feature vectors for the specified number of mixtures in the HMM.

`endian`: Makes sure endianness of variables is correct on all platforms, so that HMTSR can run on a SUN for instance.

`grammar`: Implements a finite state network class, parsing the description from file and creating the network structure in memory. This is used to restrict grammar in the level-building search.

[1] See website `http://www.ee.up.ac.za/ee/pattern_recognition_page/HMTSR/HMTSR-0.2.tgz`

`math`: Routines to calculate Fast Fourier Transform, Discrete Cosine Transform, Inverse Discrete Cosine Transform, Power Spectrum.

`mem`: Routines for allocating and freeing memory regions for 2-dimensional,3-dimensional and 4-dimensional arrays of arbitrary type.

`processing`: Algorithms implemented to calculate pre-emphasis, mel-spaced filter banks, mel-cepstra, hamming window, based on parameters of the system such as windowsize (in milliseconds), stepsize (milliseconds), how many mel-cepstral coefficients to calculate. All the feature extraction is done by these routines. A raw soundfile is passed in, and a matrix of cepstral coefficients vs time (frame) is returned. Can also optionally perform Spectral Mean Subtraction, since this must be done on the power spectrum of the signal before the mel-cepstra are extracted.

`sound`: Class that deals with audio file formats, to extract the raw audio data as an array of floating point numbers from NIST, AIFC and raw data files. Supports various sample rates, bits per sample and number of channels.

`speech`: Class to encapsulate `sound` and `processing`. A speech object is instantiated with a filename as parameter, automatically retrieves the speech data from the file, and performs feature extraction and manipulation, e.g. Cepstral Mean Subtraction and temporal derivative calculation.

`transcription`: A class to deal with transcriptions of speech data, i.e. labeling which HMM represents which section of speech. Converts this information to and from the `.lola`[2] file format.

`util`: Various utility routines, e.g. parsing a configuration file for parameters.

`hmm`: This actually consists of a number of classes and utility functions.

[2] **.lola** comes from **lo**cation **la**bel, a file format for labeled transcriptions found in the OGI CSLU toolkit. See website `http://cslu.cse.ogi.edu/toolkit/old/man/n/seglist.html` for details of the file format.

- `logmath`: Utility functions to perform log mathematics. Log mathematics is used throughout the HMM calculations since it involves addition and subtraction instead of multiplication and division, resulting in faster, more efficient code.

- `state`: This class represents a state of an HMM, containing all the mixtures of that state with mixture weights, with all the means and variances of each mixture, the transition probabilities (or state duration gamma pdf properties), function to calculate the observation probability for the state (weighted sum of mixture probabilities), as well as functions to update the state parameters iteratively.

- `hmm`: This class represents the hidden Markov model, consisting of state objects, transition probability matrix (or state duration parameters) and a model name as well as a display name (e.g. the silence model has a blank display name). The class has methods to perform vector quantization, expectation-maximization, Viterbi alignment and duration parameter training.

- `search`: This class implements the level-building search with optional duration modeling.

- `model`: The model class consists of a number of HMM objects. It is used to group HMMs with similar parameters (in terms of number of states, mixtures and feature dimension), and can parse a text file describing the model to create the `hmm` objects.

- `duration`: This class encapsulates a single gamma pdf precalculated array, and the parameters (mean and variance), as well as functions to calculate *lngamma* and the *lngammapdf*.

Bibliography

[1] J.T. Chien and H.C. Wang, "Phone-dependent channel compensated hidden Markov model for telephone speech recognition," in *IEEE Signal Processing Letters*, June 1998, vol. 5, pp. 143–145.

[2] Z. Guojun, Z. Lieguang, and F. Chongxi, "Automatic telephone operator using speech recognition," in *Proceedings of the International Conference on Communication Technology (ICCT)*, 1996, vol. 1, pp. 420–423.

[3] N. Fraser, B. Salmon, and T. Thomas, "Call routing by name recognition: Field trial results for the Operetta system," in *IEEE Workshop on Interactive Voice Technology for Telecommunications Applications (IVTTA)*, 1996, vol. 1, pp. 101–104.

[4] K. Koumpis, V. Digalakis, and H. Murveit, "Design and implementation of auto attendant system for the T.U.C. campus using speech recognition," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Processing*, May 1998, vol. 2, pp. 845–848.

[5] A. Kellner, B. Rueber, F. Seide, and B.-H. Tran, "Padis - an automated telephone switchboard and directory information system," in *Speech Communication*, 1997, vol. 23, pp. 95–111.

[6] H. Schramm, B. Rueber, and A. Kellner, "Strategies for name recognition in automatic directory assistance systems," in *Speech Communication*, 2000, vol. 31, pp. 329–338.