Modeling Speech using a Partially Observable Markov Decision Process

A dissertation

submitted by

Michael Jonas

In partial fulfillment of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

TUFTS UNIVERSITY

May 2003

© 2003, Michael Jonas

Advisor: James S. Schmolze
Abstract

For over a decade, the Hidden Markov Model (HMM) has been the primary tool used for acoustic modeling in the field of speech recognition. In this dissertation we examine a more general approach using a Partially Observable Markov Decision Process (POMDP) to model the base phonetic unit. We introduce the concept of multiple phonetic context classes, one for each of the infinite possible contexts a phoneme can be in, and show how a POMDP can be used to represent such a model. Much the same way that tying mixtures at the state level across phonemes sharing linguistic properties is used to fill in gaps in the model space due to lack of data, the POMDP model can fill in additional gaps, in effect adding a second level of clustering driven by the data itself.

An important result of this work is the ability of the POMDP model to represent the base phonetic unit, in all its possible contexts, within a single model. This enables us to look at the problem in a more abstract manner, allowing us to model better a phoneme to its fullest capability by developing relationships among its contexts that have thus far not been explored in current speech recognition research.
Acknowledgment

I would first like to thank my advisor and committee chairperson James Schmolze who has been highly supportive during my stay at Tufts University. His encouragement, sincerity, intelligence and humor were always greatly appreciated and extremely valuable.

I would also like to thank the members of my committee, which include Anselm Blumer, Stephen Levine and Amro El-Jaroudi. I would especially like to thank Amro, who was the first person to enlighten me on the fundamentals of speech recognition. Without the knowledge that he bestowed onto me, I doubt I would have been able to pursue this work.

Finally I want to acknowledge the many great people at BBN Technologies, with whom I shared many interesting and insightful years with as a staff scientist, struggling to learn everything I could about speech recognition. I would especially like to thank my mentor, Jim Van Sciver, without whose guidance throughout the years, I doubt I would be where I am today. He has supported me both professionally as a colleague, as well as personally as a dear friend. I will always fondly remember our soccer matches played on those sweltering summer evenings in and around the Boston area.
Contents

1 Introduction .............................................................................................................1
   1.1 Automatic Speech Recognition ................................................................. 2
   1.2 Notation and Style ................................................................................. 4
   1.3 Thesis Overview ................................................................................. 5

2 Hidden Markov Models ..................................................................................7
   2.1 Models in Speech ................................................................................ 7
   2.2 Template Matching ......................................................................... 8
      2.2.1 Dynamic Time Warping ............................................................... 9
      2.2.2 Decoding via DTW ................................................................... 11
      2.2.3 Limitations ............................................................................... 11
   2.3 Hidden Markov Models ....................................................................... 12
      2.3.1 Definition of HMM ............................................................... 13
      2.3.1.1 Probability of the observation ........................................... 15
      2.3.2 Training models with EM ....................................................... 16
         2.3.2.1 Baum-Welch algorithm ............................................... 17
      2.3.3 Discrete EM ..................................................................... 19
      2.3.4 Continuous EM .................................................................. 21
      2.3.5 Mixtures ......................................................................... 23
      2.3.6 Context modeling ............................................................. 25
      2.3.7 Tying mixtures .................................................................. 25
         2.3.7.1 Tied Mixtures ............................................................. 26
3.6.3 Cross-context Mixed Model Viterbi .............................................................. 59
   3.6.3.1 CMM weighting strategies ................................................................. 63
3.7 Summary ............................................................................................................... 64

4 Results ................................................................................................................... 66
   4.1 PASS system .................................................................................................. 67
   4.2 ISIP system .................................................................................................. 68
   4.3 Experiments .................................................................................................. 69
      4.3.1 TIDigits corpus .................................................................................. 70
         4.3.1.1 TIDigits training set ................................................................. 70
         4.3.1.2 TIDigits development set ....................................................... 71
      4.3.2 TIMIT corpus ...................................................................................... 72
         4.3.2.1 TIMIT training set .................................................................. 73
         4.3.2.2 TIMIT development set ......................................................... 73
      4.3.2.3 Baseline .......................................................................................... 75
   4.4 Experimental results ..................................................................................... 77
      4.4.1 UMM Viterbi ....................................................................................... 78
      4.4.2 WMM Viterbi ....................................................................................... 79
      4.4.3 CMM Viterbi ....................................................................................... 80
   4.5 Summary ....................................................................................................... 82

5 Analysis ................................................................................................................. 84
   5.1 Behavior of decoding .................................................................................. 85
      5.1.1 UMM Viterbi ....................................................................................... 86
      5.1.2 WMM Viterbi ....................................................................................... 87
      5.1.3 CMM Viterbi ....................................................................................... 88
   5.2 Data independence ....................................................................................... 88
Chapter 1

Introduction

We introduce a new model to represent the basic phonetic unit of an acoustic speech signal. For the past two decades, most research in speech recognition has used the Hidden Markov Model (HMM) to represent the acoustic features of a phoneme. We use a more general model, a Partially Observable Markov Decision Process (POMDP), in place of an HMM.

Although HMMs have proven to be successful in modeling speech, the complexity of systems using them has grown profoundly [Lev83] [Rab89] [Jua92] [Ost96]. In order to optimally utilize HMMs, the number of models and the interrelationships between those models has grown exponentially. An HMM no longer represents just a single phoneme, representing instead the phoneme plus some context surrounding that phoneme. Additionally, each HMM exchanges information with other HMMs that have linguistic similarities. We propose to capture the HMM’s complexity and its associated
interrelationships in a richer, more general model: a POMDP [Whi80] [Che88] [Lit96] [Cas98]. This enables us to represent the basic unit in speech, a phoneme, with all its different contexts in a single, unified model.

In this work we demonstrate the versatility of the POMDP model by showing how such a model gives a better representation of a phoneme compared to current HMM based systems. We also show how such a model improves overall recognition performance by enabling a richer, more robust framework for the parametric feature set of the acoustic speech signal. By collecting related information, organized into a single model, we are able to simplify our understanding of the relationships that exist between the acoustic models and associated sets of observations.

We describe how such a POMDP model is constructed and introduce several new search methods, based on the Viterbi algorithm, that utilize the additional features provided by the model to improve overall performance. A rigorous set of experiments, designed to fine tune parameters of the POMDP model, demonstrate that the model performs well under these new search algorithms using two small data sets, the TIDigits and TIMIT corpora.

1.1 Automatic Speech Recognition

Speech recognition is the process of taking an audio signal of spoken language and converting it to a sequence of words representing what was said. Although simple to describe, this process requires many complex steps.

Figure 1.1, below, illustrates the main parts needed to process a speech signal into a sequence of words. The analysis step converts the raw audio into a set of numeric values describing the signal. The dictionary contains phonetic spellings of all possible words that can be recognized, and the language model is a statistical summary of word combinations that help us determine which words are more likely to occur in pairs or triplets [Bah89] [Jel90].
The acoustic model drives the entire process by combining all that information to achieve a hypothesis of what was said.

\begin{figure}[h]
\centering
\includegraphics[width=0.7\textwidth]{speech_recognition_process.png}
\caption{The speech recognition process}
\end{figure}

In this work we concentrate on the block labeled “acoustic model,” a set of mathematical representations of a related set of speech features [Rab78a] [Hua90] [Jua92]. We discuss how these acoustic models were represented in the past, in the form of prototype templates, as well as how they are currently represented, as HMMs [Rab93]. It is important to understand the relationship between the past model and the present model and how the latter improved on the shortcomings of the former. We create a new speech model based on a POMDP that will parallel this relationship, thus providing a similar improvement over shortcomings of an HMM.

By combining information from multiple, related HMMs, we are able to generate a more accurate representation of a phonetic model. This new model, a POMDP, is a representation of a generalized form of an HMM. A POMDP will enable greater flexibility in associating different acoustic features, represented in a single framework. Flexibility in this acoustic framework will enable us to devise a sophisticated set of algorithms that, along with a new set of parameters, will better map the acoustic feature stream to their representative phonetic units.
CHAPTER 1: INTRODUCTION

More important, however, is the potential for exploring entirely new avenues of research. A POMDP provides a general framework for representing a phonetic unit in its purest form, by collapsing all related phonemes into a single model and having that model represent a phoneme in all possible situations. This lets us explore the interrelationships between related phonetic models, and opens up as yet unexplored new feature sets to the speech recognition research community.

1.2 Notation and Style

The following set of style guidelines will be employed throughout the text of this document. When introducing important new terms, we will italicize the first occurrence of that term in our document. When listing any type of source code we will use the 10 point font courier.

All equations will be numbered by chapter and the order of sequence they appear in that chapter, and the number will be placed in the rightmost margin next to each equation. Figures and tables will be labeled accordingly and also numbered by chapter and the order of sequence they appear in that chapter. Important results in tables will be highlighted using shading of the particular cell or group of cells in question.

Each chapter begins with a brief introductory discussion giving a synopsis, outlining any important information that the reader should be made aware of. At the end of each chapter, a summary will provide highlights of the more important points that were discussed and presents an overview of the chapter.

Citations will generally be placed at the end of the relevant sentence or paragraph pertaining to the referenced material. They will take the form of bracketed three letter and two digit strings representing the primary author’s first three letters of their last name followed by the last two digits of the referenced works published date. Additionally, these
citation markers will be incorporated into the complete set of references listed at the end of this document.

1.3 Thesis Overview

This dissertation is divided into six chapters. Chapter one, this chapter, gives an overview, an introduction on the speech recognition problem, plus notation and style conventions.

Chapter two outlines the current body of research on speech recognition. A brief discussion on the precursor to today’s HMM based speech systems, namely template matching, precedes the main thrust of this chapter. The majority of chapter two describes the speech recognition process as it applies to building acoustic models using HMMs. We go into great detail discussing all aspects of modern acoustic modeling techniques. We end this chapter by discussing several variations on HMMs that have been proposed to augment and improve the model.

Chapter three introduces our new speech model, as represented by a POMDP. We first give some background on traditional uses of and solutions to the POMDP model, as well as a simpler form know as a Markov Decision Process (MDP). We then show how to apply POMDPs to the speech recognition process, and give a representation of a phoneme as a POMDP. The chapter ends by discussing several new search variations on the Viterbi algorithm, which is the traditional search engine for HMMs.

Chapter four discusses the experimental framework devised to test our new POMDP model on two small data sets, the TIDigits and TIMIT corpora. A brief description of the experimental environment, the PASS system, as well as the code base of the ISIP speech recognition system is given. Next, both the TIDigits and TIMIT corpora are presented in detail, describing both their respective training and development sets. A set of initial HMM
based baseline results are presented, followed by a set of experimental results for each of the new decoding methods for our new model.

Chapter five begins with a discussion of the experimental results obtained. All three search methods are described and evaluated. We also touch on the importance of data independence and give additional results that address this issue.

Chapter six concludes with a discussion of the significance of the new POMDP model, not only as a means to achieve better recognition performance, but as a rich framework to model better the different aspects of the acoustic speech signal. We end this chapter with plans of future work needed to improve the model even further.
Chapter 2

Hidden Markov Models

The goal of an automatic speech recognition system is to convert an audio signal of speech into a sequence of words representing what was said. This conversion is done by matching the audio signal to a set of models representing units of speech. The basic models may represent words or more commonly phonetic units [Bro87]. In this chapter we look at various models and techniques used to accomplish this task.

2.1 Models in Speech

The basic principle in modeling speech is to generate unique mathematical models for each possible phonetic unit [Bil99b]. Once generated, these models are then used to match up an audio signal to a sequence of models that best fit the incoming signal [Jua85b] [Bah86]. The result is a reconstruction of what was said.
Most current speech recognition systems are based on using a Hidden Markov Model (HMM) to represent each phonetic unit [Por88] [Rab86] [Rab89]. Due to the many sources of variability in speech, this model has been shown to perform better than other modern techniques, such as Artificial Neural Networks (ANN), because of its ability to model both the time dependent nature of the acoustic signal and the random nature of the values of its feature set.

A predecessor of the HMM was the use of prototype templates that classified each phonetic unit into a sequence of time aligned values corresponding to the states of the phoneme as we transition through it [Rab93]. This scheme was simpler than HMMs, and forced an averaging of the feature space that reduced flexibility in modeling the speech signal's variability.

In this chapter we begin by examining the template matching approach and explain how addressing its weaknesses led to use of HMMs. In chapter three, we examine a more general form of HMMs called a Partially Observable Markov Decision Process (POMDP) and explain how weaknesses in HMMs led to our use of POMDPs.

### 2.2 Template Matching

One of the traditional methods of modeling speech is based on matching a set of prototype templates to an incoming feature stream. The feature stream is comprised of a sequence of frames, typically 10 milliseconds in length, created during the analysis phase (see figure 1.1). Each frame is represented by a vector of numeric values where each value measures a particular acoustic feature. This representation of the acoustic signal is used with all models for phonemes: prototype templates, HMMs, and POMDPs.

For each phoneme, a template is generated that represents a prototype of that phoneme. The prototype is made up of multiple, sequential states, usually numbering 3 or 5. The
number of states is generally determined by the user during system configuration. There is a single, weightless transition between each consecutive state. Each state represents the averaged value of the collection of frames of the incoming feature signal that map directly to the state of a specific phoneme. The sequence of states in the prototype represents the progress in time through the pronunciation of the phoneme.

Figure 2.1 illustrates a 5 state prototype template. The dashed lines represent zero weight transitions.

The length of a prototype template (i.e. the number of states for each prototype) is generally uniform and predetermined, although various strategies could assign different lengths to different prototypes based on linguistic information or phonetic information derived directly from the training data.

### 2.2.1 Dynamic Time Warping

Training begins by generating a prototype template for each phoneme. We collect all sample cases of acoustic data for each phoneme, align all samples associated with a specific phoneme to the template for that phoneme, and average the feature values at each state. To align an acoustic sample to a phoneme means to identify which frame or collection of successive frames are associated with each state in the template. This association is called a “path.”
Figure 2.2: Sample alignment path of observed data to a prototype

Figure 2.2 illustrates an observed template aligned to a prototype template. In this particular example, since the observed data has one fewer frames than the number of states in the prototype template we skip a state in our model during alignment. More commonly, the number of frames will be greater than the number of states in the prototype template, in which case successive frames are aligned to the same state, thus compressing the observed set of features to fit the prototype template.

Alignment is done using a technique called dynamic time warping (DTW) [Rab78b] [Rab93], which finds the alignment path that minimizes the distance between the observed features and the prototype’s representation of those features. In other words, DTW finds the path that minimizes the distance:

$$\min_{\text{path}} \sum_{i=1}^{n} \text{dist}(O(i), P_{\text{path}}(i))$$  \hspace{1cm} (2.1)

where \(n\) is the length of the observed sequence, \(O(i)\) is the set of features from the observed sequence at time \(i\), and \(P_{\text{path}}(i)\) is the set of features from the prototype sequence along the alignment path at time \(i\).
Note that dist() in our equation is some distance measure. We use squared error in the formulation given below:

\[
\text{dist}(O(i), P(i)) = \sum_{j=1}^{NF} (f_{O(i,j)} - f_{P(i,j)})^2
\]  

(2.2)

where NF is the number of features, and f is the set of features.

### 2.2.2 Decoding via DTW

Once we have trained our models of prototype templates, we can use them on unknown data to determine what was said, e.g. we decode using our prototypes. We do this by aligning a given input feature sequence with the state sequence of the prototype template for each of the phonemes using the same technique as was used in training. We derive a score from each aligned template using a distance metric to the input sequence.

By concatenating the scored phoneme prototypes, we then create sequences of words, dictated by a dictionary of allowable phonetic spellings, and sentences, dictated by a language model of likely word pairs. Our final answer is the combination of prototype templates yielding the highest overall score.

### 2.2.3 Limitations

One of the biggest problems with using the template matching approach is that a prototype represents each state's feature set mappings by a single, averaged value. This averaging of the feature space is not good for capturing the random nature of the feature set that characterizes a given phoneme because of the variability that is inherent in the speech signal.
What is needed is a more robust model, one that could accommodate the variability within speech. In the next section we introduce such a model and discuss how it solves these shortcomings.

2.3 Hidden Markov Models

What we are looking for in speech is the most likely words given what we observe. Using a probabilistic model allows us to divide the problem into stages and combine various sources of knowledge under a single mathematical framework [Lev83] [Rab89] [Jua92] [Ost96]. We seek to find the word sequence $W$ that maximizes $P(W \mid O)$ where $O$ is the sequence of observed acoustic features. By applying Bayes’ Rule, we get:

$$P(W \mid O) = \frac{P(O \mid W) P(W)}{P(O)} \quad (2.3)$$

which we try to maximize by choosing $W$.

$P(O)$ is the probability of the acoustic features from the incoming acoustic signal and can be ignored as it does not affect the maximization process. $P(W)$ represents the probability of the word sequence occurring and is computed by a language model, which we do not discuss further in this work. Of interest to us is $P(O \mid W)$, the probability of the sequence of observed acoustic features given the word sequence [Fer90] [Rab89] [Rus95]. We examine this more closely in the sections that follow.

The speech signal exhibits inherent randomness in its feature set. However, it is not some noisy distribution around a single mean as assumed in the template matching approach. This randomness, or variability in speech, can result from many different sources, including the physical characteristics of the speaker, the dialect and accent, the technical nature of how the
speech was recorded and the environment it was recorded in. We need a way to capture this variability within our model.

A Hidden Markov Model (HMM) is a double random process where the underlying random process is hidden. The process is said to go through “states” where the next state depends only on the current state and not on previous states. This type of process is known as a first order Markov chain, incorporating a minimum amount of memory, in this case a history of only the current state, without actually being memoryless [Rus95] [Jel97]. The hidden process can only be observed through a second random process that produces a sequence of observed symbols.

We use an HMM as a base model for each phonetic unit. Because an HMM describes a process whose behavior changes and whose states cannot be observed directly, we use observed data as clues to the sequence of states that the process took. We use a probability density function (pdf) to represent the probability of the features we observe for each state of a phoneme. This captures the variability of the speech signal and helps solve the primary shortcoming of the template matching approach.

Additionally, because the beginning of a phoneme differs from the end, the HMM model can incorporate this variability along time. By allowing for transition skips and self loops in our model (see below), we are better able to account for the variability in the length of each phoneme.

2.3.1 Definition of HMM

As discussed in the previous section, an HMM describes a process whose rules change. Since we cannot directly observe the states of the process, we must rely only on observed sequences of features which may yield a clue as to the sequence of states taken. The next state
depends only on the current state and not on past states. We give a formal definition of an HMM below [Kae98] [Sha99].

An HMM is defined as a model $\lambda = \langle S, \Theta, A, B, \pi \rangle$ where

- $S = \{s_0, ..., s_{N-1}\}$ is a finite set of $N$ states.
- $\Theta = \{o_1, ..., o_T\}$ is a set of $T$ observations.
- $A$ is the state transition probability distribution matrix where $a_{i,j}$ is the probability of going from state $i$ to state $j$. Formally, $a_{i,j} = P(q_{t+1} = s_j \mid q_t = s_i); q_t$ is the state at time $t$.
- $B$ is the observation symbol probability distribution matrix where $b_j(o)$ is the probability of observation $o$ in state $j$. Formally, $b_j(o) = P(v_t = o_t \mid q_t = s_j); v_t$ is the observation recorded at time $t$.
- $\pi$ is the initial state distribution where $\pi_i$ is the probability of starting out in state $i$.

Speech uses a left-to-right representation of an HMM that models the time dependent nature of a speech signal. Current speech systems use either a 3 or 5 state HMM to model a phoneme [Del93]. Figure 2.3 shows a sample HMM as used in standard recognition systems. It has self-loops at each state plus skips between every other state.

Figure 2.3: a 5 state HMM with skips and self-loops
Note that this representation requires a minimum of three states to pass through the model. If each state represents 10 milliseconds of signal, then the shortest representation of a phoneme is 30 milliseconds.

In the sections that follow, we explain how to train and use HMMs in speech recognition. Each phoneme will be represented by its own HMM (or a collection of HMMs when introducing context).

### 2.3.1.1 Probability of the observation

From here until the end of section 2.3.8 we assume that we have a particular sequence of observations for a single phoneme. Let T be the length of the sequence. Let $O = <O_1, O_2, \ldots, O_T>$ be the sequence of observations and let $I = <I_1, I_2, \ldots, I_T>$ be the sequence of states the process followed. At the end of section 2.3.8, we show how to combine our phoneme models to estimate $P(O \mid W)$.

Training an HMM will require $P(O \mid \lambda)$. One way to calculate this is:

$$P(O \mid \lambda) = \sum_I P(O \mid I, \lambda) P(I \mid \lambda) \quad (2.4)$$

The first term is calculated with:

$$P(O \mid I, \lambda) = b_{I_1}(O_1) \ast \cdots \ast b_{I_T}(O_T) \quad (2.5)$$

The second term via:

$$P(I \mid \lambda) = \pi_{I_1} \ast a_{I_1,I_2} \ast \cdots \ast a_{I_{T-1},I_T} \quad (2.6)$$
We now have a way to compute $P(O \mid \lambda)$, though this particular method is not feasible since there are exponentially many $I$’s. We will soon see a better method. The above maps out the overall approach to speech recognition and highlights the central role played by the acoustic model. In the sections that follow we will discuss how effectively to generate $\lambda$ by using an iterative algorithm known as EM [Dem77]. We will also describe an efficient dynamic programming solution to EM for HMMs known as Baum-Welch [Bau72].

### 2.3.2 Training models with EM

In speech, the states of an HMM represent the pronunciation task: beginning, middle, end of phoneme. We observe acoustic features associated with each state. The randomness in the state transitions accounts for time stretching in the phoneme: short, long, hurried pronunciations. The randomness in the observations accounts for the variability in pronunciations.

The basic principle in generating a model in speech is to align the observations with the states, then based on the alignment, estimate the transition probabilities and the probability distribution of the observations.

The EM algorithm is a broadly applicable algorithm for computing maximum likelihood estimates from incomplete data [Red84] [Mei89] [Moo96] [McL97] [Mit97]. Since each iteration of the algorithm consists of an expectation step followed by a maximization step, the algorithm is termed *Expectation-Maximization*, or EM for short. The EM algorithm has a wide range of applicability due to its simplicity and generality.

In general, the EM algorithm can be applied to estimate a set of parameters that describe an underlying probability distribution, given only the observed portion of the data produced by that distribution. Beginning with an arbitrary initial hypothesis of values for the set of
unknown parameters, EM repeatedly calculates the expected value of those parameters given that hypothesis (the expectation step) and then finds a new hypothesis that maximizes the likelihood of those expected values (the maximization step) until it converges to a local maximum likelihood hypothesis [Wu83] [Xu96] [Ort99].

We relate the EM algorithm to the speech process in the sections that follow, focusing specifically on creating acoustically trained HMM models in both the discrete and continuous form. First, however, we'll look at a dynamic programming [Bel57] [How60] approach to solving the EM algorithm.

### 2.3.2.1 Baum-Welch algorithm

Baum-Welch [Bau72] is a dynamic programming, implementation of the EM algorithm for HMMs. The state space for the EM algorithm grows exponentially as the number of possible models grows. Looking at equation 2.4, we note that since the state sequence is not known, in order to compute the probability of the observation we need to compute over all possible state sequences. This ends up being impossible to compute as there are $N^T$ possible sequences each requiring $2T$ multiplications.

To reduce the size of the computation we take a dynamic programming approach by defining a recursive calculation that avoids repeated computations of the same values. We denote $\alpha_t(j)$ as the probability of getting the sequence of observations up to time $t$ and of being in state $j$ at time $t$, i.e., $\alpha_t(j) = P(O_1, ..., O_t, I_t = j | \lambda)$ [Rab89] [Del93]. Notice that $\alpha_t(j)$ is independent of the actual path taken at times 1,...,t-1, which works thanks to the Markov property.

The recursive solution for $\alpha_t(j)$, known as the “forward variable,” uses the following two equations
\[ \alpha_t(j) = \pi_j b_j(O_t) \]
\[ \alpha_{t+1}(j) = \{ \sum_i \alpha_t(i) a_{i,j} \} b_j(O_{t+1}) \] \hspace{1cm} (2.7)

At time \( t=T \), \( P(O \mid \lambda) = \sum_i \alpha_T(i) \), which is the probability that the observation sequence \( O \) was produced by \( \lambda \) via any state sequence.

Next, we define the “backward variable” in a similar manner. \( \beta_t(j) \) is the probability of getting the observation sequence from time \( t+1 \) to \( T \) and starting in state \( i \) at time \( t \), i.e., \( \beta_t(j) = P(O_{t+1}, \ldots, O_T, I_t = j \mid \lambda) \). A recursive formula for \( \beta_t(j) \) follows:

\[ \beta_T(j) = 1 \]
\[ \beta_t(j) = \{ \sum_i \beta_{t+1}(i) a_{i,j} b_j(O_{t+1}) \} \] \hspace{1cm} (2.8)

Applying both the “forward” and “backward” variables provides a concise mathematical expression from which we can construct the following set of useful probabilities:

\[ P(O, I_t=i \mid \lambda) = \alpha_t(i) \beta_t(i) \] \hspace{1cm} (2.9)
\[ P(O \mid \lambda) = \sum_i \alpha_T(i) = \sum_i \alpha_t(i) \beta_t(i) \text{ for any } t, 1 \leq t \leq T \] \hspace{1cm} (2.10)
\[ P(I_t=i \mid O, \lambda) = \alpha_t(i) \beta_t(i) / P(O \mid \lambda) \] \hspace{1cm} (2.11)
\[ P(I_t=i, I_{t+1}=j \mid O, \lambda) = \alpha_t(i) a_{i,j} b_j(O_{t+1}) \beta_{t+1}(j) / P(O \mid \lambda) \] \hspace{1cm} (2.12)

The combination of both the “forward” and “backward” variables reduces the computational cost to \( N^2T \) possible sequences.
2.3.3 Discrete EM

Use of the EM algorithm in training HMM models takes the following approach. The basic principle is to align the observations with the states of our models based on their current transition probabilities and probability distribution over the observation space. This alignment represents the expectation step of the EM algorithm. We then update our models by using these alignments to estimate new transition probabilities and new probability distributions of the observations. This model update is the maximization step of EM. We continue this process until it converges to a local optimum [Wu83] [Del93].

To estimate the transition probabilities and the probability distribution of our observations, we take statistical summaries during the alignment process and normalize over all alignments. Specifically:

- we estimate the transition probabilities, $a_{ij}$, by accumulating the number of times we transition from state $i$ to $j$ during alignment and normalizing by the total number of times of being in state $i$.
- we estimate $b_j(o)$ by accumulating the number of times we observe $o$ in state $j$ during alignment, and normalizing by the total number of times of being in state $j$.

The pdf, $b_j(o)$, in the discrete case is simply a table of probabilities for each observation/state pair. Figure 2.4 below illustrates this.
Alignment is generally done using dynamic programming and specifically using the Baum-Welch algorithm. In estimating the state transition probabilities, we estimate the number of times the model transitioned from state $i$ to state $j$ when it produces $o$. This estimate is based on equation 2.12.

$$\sum_{t} P(I_{t}=i, I_{t+1}=j \mid O, \lambda) = \sum \frac{\alpha(i) \ a_{ij} \ b_{i}(O_{t+1}) \ \beta_{i+1}(j)}{P(O \mid \lambda)}$$

(2.13)

Thus, the estimated number of times the model transitioned for $i$ to $j$ takes the probability that the model transitioned from $i$ to $j$ at time $t$ and sums those probabilities for $t$ from 1 to $T-1$. Note that this estimate need not be a whole number.

For the normalization factor of the number of times the model was in state $i$ from times 1 to $T-1$ we apply equation 2.11

$$\sum_{t} P(I_{t}=i \mid O, \lambda, \lambda) = \sum \frac{\alpha(i) \ \beta(i)}{P(O \mid \lambda)}$$

(2.14)

Combining these two equations then yields our estimate of the transition probabilities using the forward-backward variables.
\begin{equation}
a_{i,j} = \frac{\sum \alpha(i) a_{i,j} b(O_{t+1}) \beta_{t+1}(j)}{\sum \alpha(i) \beta_{t}(i)} \tag{2.15}
\end{equation}

We can construct the forward-backward equation for estimating the \( b_j(o) \) in a similar manner. To determine the number of times we observe \( o \) while in state \( i \), we modify equation 2.14 and add the probability of observing \( o \). Note that now we can sum to \( T \) instead of \( T-1 \).

\begin{equation}
\sum_t P(I_t=i, O_t=o | \lambda) = \frac{\sum_t \alpha(i) \beta_{t}(i) b(O_{t}=o)}{P(O | \lambda)} \tag{2.16}
\end{equation}

When we normalize once again by the estimated number of times we were in state \( i \), equation 2.14, we get the estimate of \( b_j(o) \) using the forward-backward variables

\begin{equation}
b_j(o)' = \frac{\sum \alpha(i) \beta_{t}(i) b(O_{t}=o)}{\sum \alpha(i) \beta_{t}(i)} \tag{2.17}
\end{equation}

We do not need to estimate \( \pi \) as in speech we assume the model always starts in state 1. At this point, we have a new model \( \lambda' = \langle S, Q, A', B', \pi \rangle \). We repeat this Expectation-Maximization process until \( \lambda \) converges or we have exhausted our resources.

### 2.3.4 Continuous EM

Using continuous densities changes our computation of \( b_j(o) \) from a (discrete) table to a continuous function. We now have continuous observations and a continuous pdf. Instead of computing the probability distribution of the observations, we compute the means and variance of the continuous Gaussian curve representing that distribution. The pdf, \( N_j(o) \), now
has fewer parameters than the discrete \( b_j(o) \) to estimate and is smooth [Rab85b] [Xu96] [Del93] [Jel97].

\[
N_j(o) = \frac{1}{\sqrt{2\pi}\sigma_j} e^{-\frac{(o - m_j)^2}{2\sigma_j^2}}
\]

(2.18)

where \( m_j \) is the mean and \( \sigma_j \) the standard deviation. Note that \( \pi \) in equation 2.18 represents the value for \( \pi_i \), not the initial state distribution.

We use it as a measure of likelihood since it is impossible to speak of the probability of a real number occurring, as it is always 0, and our pdf can now have values larger or smaller than 1, making it unrepresentable as a probability. If we use a single Gaussian to represent our pdf, we have

\[
b_j(o) = N_j(o)
\]

(2.19)

The equation for the estimate of the transition probability (2.15) remains unchanged. We now need additional equations to compute the mean, which we will construct as a weighted time average of the observation vectors, weighted according to the likelihood of having been produced by the observation in state \( j \)

\[
m'_j = \frac{\sum \alpha_j \beta_j O_j}{\sum \alpha_0 \beta_0}
\]

(2.20)

and similarly we construct the variance
2.3.5 Mixtures

Speech is known for having data from different clusters due to the various sources of variability. Since a Gaussian density is unimodal, it is incapable of modeling data with multiple clusters. We need to develop a multimodal pdf for speech [Lip82] [Red84] [Jua85a] [Jua86] [Bil98].

A Gaussian mixture is defined in speech as follows

\[
\beta_j(o) = \sum_k c_{jk} N_{jk}(o) \tag{2.22}
\]

where \( N_{jk}(o) \) are K Gaussian pdfs with means \( m_{jk} \) and variances \( \sigma_{jk}^2 \).

The mixtures have K Gaussians and can represent data with K peaks. A weight, \( c_{jk}(o) \), is assigned to each peak representing the contribution of the \( k^{th} \) Gaussian to the likelihood of being in state \( j \) and observing \( o \), normalized by the likelihood of being in state \( j \) (see figure 2.6). We now have multiple means, variances and mixture weights to estimate per state.
Again the equation for the estimate of the transition probability (2.15) remains unchanged. We now need equations to compute the multiple means

$$m_{j,k}' = \frac{\sum_{t} \sum_{i} \alpha_{t-1}(i) a_{ij} c_{j,k} N_{j,k}(O_t) \beta_{t}(j) O_{t}}{\sum_{t} \sum_{i} \alpha_{t-1}(i) a_{ij} c_{j,k} N_{j,k}(O_t) \beta_{t}(j)}$$

(2.23)

multiple variances

$$\sigma^2_{j,k}' = \frac{\sum_{t} \sum_{i} \alpha_{t-1}(i) a_{ij} c_{j,k} N_{j,k}(O_t) \beta_{t}(j) (O_{t} - m_{j,k}')^2}{\sum_{t} \sum_{i} \alpha_{t-1}(i) a_{ij} c_{j,k} N_{j,k}(O_t) \beta_{t}(j)}$$

(2.24)

and a new equation that represents the individual mixture weights

$$c_{j,k}' = \frac{\sum_{t} \sum_{i} \alpha_{t-1}(i) a_{ij} c_{j,k} N_{j,k}(O_t) \beta_{t}(j)}{\sum_{t} \alpha_{t}(j) \beta_{t}(j)}$$

(2.25)

One final step is applied to reduce computation. In general, only the top N Gaussians with highest likelihood measures are updated by the observations with each alignment during EM. This reduces computation by only accumulating a fraction of the Gaussians.
2.3.6 Context modeling

To account for coarticulation effects of speech, building phoneme models in context improves performance [Sch85] [Lee90]. A model in context differentiates between phonemes with different preceding and following phonemes, i.e. the left and right context a phoneme finds itself in is considered when building individual models. When we look only at individual phonemes to the immediate left and right of the center phoneme, we call this context model a triphone. The number of triphone models that need to be built now grows by the cube of the number of total phonemes. With approximately 50 phonemes in English, the number of possible triphone context models becomes $50^3$, each with a Gaussian mixture and state transition matrix [Del93].

The number of free parameters now becomes very large, on the order of $10^7$. Estimating that many parameters becomes intractable as the amount of training data, computing power and storage would exceed current systems, not to mention the time requirements. We thus need to reduce the number of free parameters.

2.3.7 Tying mixtures

The idea of tying mixtures is to share parameters among models, thus reducing the number of parameters [Bel90]. This is needed not only because there is insufficient data to estimate all parameters, but also because the calculation at recognition time may be computationally too expensive [Jel97]. If the right number of parameters are shared, we can reduce the model space without reducing its accuracy. Of course if too many parameters are shared we lose discriminability and end up washing out models. Great care must be taken in choosing the method of combining parameters.

Mixture sharing can take place between two possible parameter classes. We can tie mixture densities, i.e. their means and variances, and we can tie their corresponding mixture
weights. In general, when discussing tied mixture systems, the mixture densities are always tied but the mixture weights may not be. The amount of information tied depends both on the strategy used as well as the amount of computing resources and data available.

Various strategies exist in tying mixtures. The following sections will focus on three methods, each more complex than its predecessor. We will discuss tying mixtures across all models, tying mixtures across models sharing the same context, and finally, tying mixtures across clusters of states within models.

2.3.7.1 Tied Mixtures

The simplest way of sharing parameters is to tie mixtures across all models. We refer to this as a Tied Mixtures (TM) system [Jar98]. With TM, all context models share the same Gaussian means and variances, but not their mixture weights, creating a single mixture of many Gaussians.

In general, about 10,000 Gaussians is sufficient to achieve reasonable performance. Ideally, each individual model falls into its own, unique segment of this single pool of Gaussian mixture densities. In practice, however, performance suffers, compared to other methods, as model dependent specific features get washed out by other, non related models. It is therefore important to approximate mixture weights separately, as it greatly increases the discriminating power by tailoring each segment to its specific model.

2.3.7.2 Phonetic Tied Mixtures

An improvement over the TM method of tying a single set of mixtures across all phonetic models is to generate a set of tied mixtures across all context dependent phonetic models belonging to the same phoneme. We refer to this as a Phonetic Tied Mixture (PTM) system [Jar98].
CHAPTER 2: HIDDEN MARKOV MODELS

With a PTM system the number of mixtures are reduced from approximately 50\(^3\), one for each of the possible context phonemes, down to 50, one for each phoneme, in a typical standard English data set. The number of Gaussians per mixture will vary among systems and depends both on the user's preference as well as the limitations on the system's design. This method tends to perform better than a TM system as each shared mixture density model remains independent across phonemes.

Having mixtures tied at the phoneme level reduces the effects of context, thus losing some discriminability. Keeping mixture weights independent is important in retaining some of the knowledge gained from using context. Therefore, mixture weights are generally not shared.

2.3.7.3 State Clustered Tied Mixtures

Sharing the same Gaussian means and variances among clusters of states, where states do not necessarily belong to models sharing the same phoneme is a method that can yield the most accurate models while still reducing the number of parameters. These State Clustered Tied Mixture (SCTM) [Jor93] [You94a] [You94b] [Jar98] systems reduce the number of mixtures to a few thousand, each requiring a smaller number of Gaussians per mixture than typically needed in a PTM system.

Although the number of Gaussians per mixture is determined by the user, limited only by the system's design, the number of mixtures depends on the data itself. Clusters are tied at the state level based on linguistic knowledge using a decision tree based approach [Par01]. As states are tied, a small portion of the training data is used to determine which models are sufficiently trained and require no tying and which undertrained models are acoustically similar to those sufficiently trained models.

With this technique, tying mixture weights tends to improve performance of the models, at the cost of additional complexity in the overall system.
2.3.8 Decoding models with Viterbi

The final step in a speech recognition system is the process of taking an unknown speech signal and applying the trained acoustic models to determine what was said. As discussed in section 2.3, we want to find the W that yields the maximum P(W) P(O | W), where P(W) is the probability of the word sequence occurring and P(O | W) is the probability of the observation given the word sequence. The former is calculated from the language model and the latter comes from the acoustic model.

Since we do not have the word sequence, we have to look at all possibilities. Since each word is a sequence of phonemes, our search for the best W will require finding the most likely sequence of phonemes. To determine the likelihood of a given phoneme, we seek the sequence of states, I, through a given phoneme’s HMM that maximizes P(I | Ø, λ), where Ø is the subsequence of observations that corresponds to the observations associated with the given phoneme. As we will see in the next section, we do not really need this I, we just need to know P(I | Ø, λ) that is maximum over all sequences I.

We want to maximize P(I | O, λ). Using Bayes’ rule we rewrite this as:

\[
\max_I P(I | O, \lambda) = \max_I \frac{P(O | I, \lambda) P(I | \lambda)}{P(O | \lambda)}
\]  \hspace{1cm} (2.26)

Which is equivalent to:

\[
\max_I P(I | O, \lambda) = \max_I P(O | I, \lambda) P(I | \lambda)
\]  \hspace{1cm} (2.27)

Which is equivalent to:
\[
\max_{I} P(I \mid O, \lambda) = \max_{I} P(O, I \mid \lambda) 
\] (2.28)

The Viterbi algorithm \cite{Vit67} computes the maximum $P(O, I \mid \lambda)$ efficiently. Instead of examining the exponential number of sequences $I$, $\delta_t(j)$ is constructed similarly to the forward variable $\alpha_t(i)$. However, instead of summing over all probabilities, $\delta_t(j)$ computes the maximum probability of the observation sequence up to time $t$ and being in state $j$ at time $t$. Since it does not keep track of the state sequence that leads to this maximum, we also define $\psi_t(j)$ to recover the actual state sequence. $\psi_t(j)$ is the state just before $j$ that leads to this maximum probability sequence.

We define $\delta_t(j)$ as follows:

\[
\delta_t(j) = \max_{\{I_1, \ldots, I_{t-1}\}} P(O_1, \ldots, O_{t-1}, I_1, \ldots, I_t \mid \lambda) 
\] (2.29)

$\delta$ is a useful quantity because $P(O, I \mid \lambda) = \max_j \delta_T(j)$. We initialize both $\delta$ and $\psi$ as follows:

\[
\delta_1(j) = \pi_j b_j(O_1)
\]

\[
\psi_1(j) = 0 
\] (2.30)

and compute $\delta$ and $\psi$ up to time $T$ recursively:

\[
\delta_{t+1}(j) = \max_i \{\delta_t(i) a_{ij}\} b_j(O_{t+1})
\]

\[
\psi_{t+1}(j) = \argmax_i \{\delta_t(i) a_{ij}\} 
\] (2.31)
Computing equation 2.31 requires on the order of $N \times T$ operations. To recover the sequence $I^*$ that yields the maximum $P(I \mid O, \lambda)$ we use:

$$I'_t = \psi_{t+1}(I'_{t+1}) \quad (2.32)$$

Computing $I'_t$, the most likely path, requires $T$ additional steps. This algorithm is widely used in current state of the art speech recognition systems as the primary engine for decoding.

The language model estimate, $P(W)$, takes the form of an additional score computed at each state of our model. We have $P(W)$ expanded as follows:

$$P(W) = w_{I_1} \ast w_{I_1, I_2} \ast \ldots \ast w_{I_{T-1}, I_T} \quad (2.33)$$

In practice most weights, $w_i$, do not contribute to the likelihood score (i.e. they are set to 1) and only at word boundaries is a weight associated with an actual language model probability. We use this weight not only for language model scores but also for adding additional rewards or penalties to the overall likelihood score. For example, the ISIP system attempts to inhibit the insertion of common, poorly articulated words such as “the”, “a”, and “uh” by adding a word insertion penalty at word boundaries [Par01]. However, the flexibility of adding a weight at every state enables us to modify the likelihood score through $P(W)$ with various other reward strategies. We will utilize this ability in chapter three when discussing modifications to the Viterbi algorithm.

### 2.3.8.1 Putting HMM models together

Our goal is to maximize $P(W \mid O)$. To construct the most likely sequence of words, $W$, we expand the acoustic HMM models of each phoneme, as represented by $\lambda$, guided by the
Viterbi algorithm. As this creates a lattice of all possible phoneme model combinations, we apply both a dictionary of phonetic spellings to include only allowable words, as well as a language model of probabilistic word occurrences to include only statistically relevant sentences, reducing the number of possible hypothesis.

Figure 2.7 below illustrates this process. A dictionary of two phonetically spelled words is specified along with a language model containing only a single bigram word pair.

The lattice depicted in figure 2.7 represents the possible expansions of triphone models during a Viterbi decoding pass. In the diagram above, when referring to a triphone model, the phoneme to the left of “-” is the left context phoneme, the phoneme to the right of “+” is the right context phoneme and the phoneme in between is the phoneme being modeled.

The dictionary guides the internal word expansions and the language model guides the crossword expansions. The transition labeled (1) is not taken because the language model prunes out this expansion since there are no word pairs that allow for a “one one” occurrence. The transition labeled (2) is not taken because the dictionary has no phonetic word spellings that include the phoneme sequence “ow m aa” for the triphone “ow-m+aa” and thus prunes this expansion.
2.3.9 Existing HMM Systems

In this section we look at two different acoustic model configurations used to build models in existing HMM speech recognition systems. We describe how each system trains its acoustic models as well as how it uses those models during decoding. The basic principle is the same for both systems, although the complexity, and therefore the overall performance, differs greatly between them.

First we look at a state of the art system developed by BBN technologies called Byblos. This system has performed near the top in recognition performance over the last decade compared with research sites throughout the world. Next we look at the ISIP system [Par01] developed at Mississippi State University, which applies many of the same techniques used in Byblos. The ISIP system is important as it is the speech recognition system that all subsequent research discussed in this paper is based on.

Building an acoustic model is only part of a complete speech recognition system. Two other important components include feature analysis and language modeling (see figure 1.1). In feature analysis the raw audio signal is converted into a set of feature vectors used throughout the acoustic modeling phase. The language model is used to guide the search algorithm to look at only reasonable hypothesis. Building a robust language model, therefore, is crucial in reducing the overall search space during the decoding phase. Although both components are integral parts to a complete speech system, we will not discuss them further in this paper.

2.3.9.1 Byblos

The Byblos system uses a hybrid PTM and SCTM configuration to create a set of context dependent, triphone based, acoustic models. It builds an initial set of small course PTM models using 64 Gaussians per mixture (a total of approximately 3,000 Gaussians when
building standard English models). It then refines those models by looking at word boundary phonemes, called crossword phones, and builds a more detailed set of crossword PTM models using 256 Gaussians per mixture (a total of approximately 10,000 Gaussians when building standard English models). It also builds a set of non-crossword PTM models using 256 Gaussians per mixture that ignores word boundary phonemes. Finally, it uses the PTM crossword models as an initial estimate to build very detailed crossword SCTM models using 40 Gaussians per mixture (a total of approximately 40,000 Gaussians when building standard English models). Once training is done, Byblos uses the non-crossword PTM models and the crossword SCTM model during decoding.

Figure 2.8 shows a diagram of the general Byblos training procedure. Note that the labeled data represents phonetic alignment estimates that are derived from the PTM 64 step. The phonetically labeled data is used in addition to the previous set of trained acoustic models and the training data to help the system converge more quickly to the best set of acoustic models.

![Diagram of Byblos training procedure](image)

**Figure 2.8:** Byblos training sequence
Byblos’s strength lies in its decoding strategy. It uses a multipass decoder to achieve both high accuracy and fast decoding results. The multiple passes are used to refine decoding results, with initial passes working quickly with less accurate models to get rough estimates of possible hypotheses and the final pass using the more detailed model to search exhaustively those rough estimates for the final answer [Gup88] [Sch96].

Byblos uses the non-crossword PTM models in its first pass to achieve an initial rough estimate of possible hypotheses, thus pruning out unlikely ones. It also uses a less detailed bigram language model. The hypotheses that are generated carry with them information on candidate words and their statistics in the form of estimated likelihoods and end times. In the second pass, the decoder reverses directions and, using non-crossword PTM models and a more detailed backward trigram language model, decodes from end to start. The third pass reverses directions again and uses the non-crossword PTM models in the forward direction, and a forward trigram language model, constructing a lattice of possible hypotheses.

The final decoding pass uses the lattice of the previous pass and the more detailed crossword SCTM model, along with a forward trigram language model, to achieve the final decoding result. The lattice of possible hypotheses arranges the candidate words into a directed graph representing possible word sequences and allows an intensive search with the best possible language model and acoustic model.

Figure 2.9 below illustrates the various steps of the Byblos decoding sequence.
**2.3.9.2 ISIP**

Similar to Byblos, the ISIP system builds models using a multiple stage trainer. It differs from Byblos in that its model building process is much simpler, a tradeoff that results in loss of recognition performance. The ISIP system, being in the public domain, was designed not for state of the art recognition performance, but more for research and development in the speech community as a whole.

The ISIP system builds a set of very coarse, untied monophone models, using only a single Gaussian per mixture (a total of approximately 50 Gaussians when building standard English models), as its initial set of models. Using those monophone models it then builds a set of detailed crossword SCTM models using 16 Gaussians per mixture (a total of approximately 50,000 Gaussians when building standard English models). It builds these models incrementally, starting off with an initial set of 2 mixture models. From those models
it creates a set of 4 mixture models, followed by a set of 8 mixture models, and finishes with a set of 16 mixture models.

Figure 2.10 illustrates the general ISIP training procedure. The labeled training data in the ISIP system differs from that of the Byblos system. It simply represents a set of modified training transcripts where specific silence markers between words that show little or no pause in signal strength have been removed.

For decoding, ISIP uses a single pass decoder on its crossword SCTM models. Compared to Byblos, the speed of its recognition suffers. Still, the ISIP recognizer achieves reasonable recognition performance because it applies an exhaustive search on its most detailed models.

2.3.10 Variations on HMMs

Variations of HMMs have been explored for some time. New types of models can be derived with varying degrees of success by adding additional sets of new features, adding
dependencies between existing feature sets, looking at sets of different features, or modifying
the relationships of features between states.

What follows in this section is a survey of four variations on the traditional HMM model. We will discuss in detail the properties of a Partially Hidden Markov Model, a Buried Markov Model, a Hidden Articulatory Markov Model and a 2D Extended Markov Model. In chapter three we will look at extending the model further, by increasing the dimensionality of its
topology on transitions and states and thus forming a Partially Observable Markov Decision Process (POMDP).

2.3.10.1 Partly Hidden Markov Models

A traditional HMM models only piecewise stationary stochastic processes based on
dependencies of the observations. By introducing a modified second order Markov Model, in
which the first state is hidden and the second one is observable [Kob99] both the observations
and the state transitions are dependent on the previous observation.

Since the output probability of feature vectors in an HMM’s state is unique, the HMM
neglects the dynamic features and deals with only piecewise stationary processes. If the
dynamical features of the speech patterns are incorporated into the model, an improved and
more robust model results. By constructing a model with the first state hidden and the second
state observable, both the observations and the state transitions are dependent on the previous
observations, making it possible to deal with a transient rather than a piecewise stationary
process. This model is called a Partly Hidden Markov Model (PHMM).

In a Markov Model (MM), the output probability of feature vector $x_t$, $P_t(x_t)$, is given by
the conditional probability of the past $K$ observations $x_{t-K} \ x_{t-K+1} \ ... \ x_{t-1}$

$$P_t(x_t) = P(x_t \mid x_{t-K} \ x_{t-K+1} \ ... \ x_{t-1}) \quad (2.34)$$
Having state $S_i$ uniquely given to the sequence of $x_{t-K} x_{t-K+1} \ldots x_{t-1}$, the previous equation becomes

$$P_i(x_t) = P(x_t | S_i) \quad (2.35)$$

When the relation between the output sequence and state is not unique but probabilistic, then our model becomes hidden (HMM).

In the PHMM, the output probability is represented by a second order model that takes the following form

$$P(x_t | x_{t-K} x_{t-K+1} \ldots x_{t-1}) = P(x_t | S_f, S_j) \quad (2.36)$$

where the mapping from the sequence of $x_{t-K} x_{t-K+1} \ldots x_{t-2}$ to state $S_f$ is probabilistic and the mapping from the output $x_{t-1}$ to the state $S_j$ is unique. Since the output probability of $x_t$ is conditioned by state $S_j$ (that means it is conditioned by $x_{t-1}$), the model can deal with a more complicated process than piecewise stationary.

The characteristics of a PHMM has its state defined by the previous $f$-state and previous observation, and its observation dependent on current $f$-state and previous observation (current $s$-state).

### 2.3.10.2 Buried Markov Models

Two conditional independence assumptions characterize HMMs. First, observations are conditionally independent of other observations given the hidden state at the current time.
Second, the hidden state is conditionally independent of any preceding variables given the previous hidden state.

When HMM conditional independence assumptions are relaxed by adding dependencies between observation elements for each hidden state value, the underlying Markov chain is further hidden (buried) by the specific cross-observation dependencies [Bil99a] [Bil00]. These additional dependencies increase both accuracy and discriminability. We refer to this resulting model as a Buried Markov Model (BMM).

To increase discriminability between different states, dependencies between observations should be chosen that both decrease entropy in the context of the correct state and do not decrease the entropy in other contexts. These additional dependencies are chosen according to natural statistical dependencies observed in the training data that are not well modeled by an HMM. The collection of dependency variables can contain observations from past, present, future, or entirely different feature streams.

Since BMMs extend Gaussian mixture HMMs by including cross observation dependencies, the observation models should allow their entropy to be affected by the additional dependencies while still leading to efficient EM update equations.

2.3.10.3 Hidden Articulatory Markov Model

HMM's can be extended by combining articulatory information with acoustic information, in effect adding additional features. We call an HMM built using articulatory information a Hidden Articulatory Markov Model (HAMM) [Erl96]. Although a HAMM alone does not generally outperform a traditional HMM, by combining both using a weighted average, a better model is derived [Ric00].

A HAMM is simply an HMM in which each state represents an articulatory configuration. An articulatory configuration represents the physical state the vocal tract is in during speech.
CHAPTER 2: HIDDEN MARKOV MODELS

production. Typical articulatory features are derived from jaw position as well as lip and tongue characteristics and may more accurately model the production of speech. The HAMM introduces knowledge about speech production, incorporated via its state space, transition matrices, and phoneme to articulatory mappings.

Given that a sequence of articulatory features represents a word, in mapping words to articulatory targets, a simplifying assumption that words can be modeled by a sequence of phonemes, each of which is mapped to a sequence of one or more articulatory configurations, is made. Phonemes are mapped into a vector of feature ranges; each feature can be in any of the values specified by the range. Some phonemes require a specification of articulator motion rather than static positioning. In these cases, a phoneme is produced by the movement from one articulatory state to another. Thus, phonemes can be mapped to a sequence of articulatory configurations.

To limit the possible articulatory configurations, both static and dynamic constraints are added. Static constraints are either physical, imposed by the limitations of the articulatory system, or others that disallow states that may be physically possible but would not normally be used while speaking naturally. Dynamic constraints are imposed on the model to prevent physically impossible articulatory movement.

Training a HAMM requires an initial model, which is iteratively improved until it converges to a local optimum. The quality of the initial model is important as it can have a significant effect on the performance of the trained model, as well as how quickly it converges. Though a HAMM tends to perform worse than standard HMM models, since it is based on articulatory knowledge, it makes different mistakes. Thus, by combining both HAMM and HMM models using a weighted sum results in improved performance over each model taken individually.
2.3.10.4 2D Extended HMM

A two-dimensional extension of HMMs looks at the conditional joint distribution of state durations in the length of utterances to improve the speech model’s performance. It extends the dependency of observation densities to current and neighboring states. A local averaging procedure is applied to smooth the outcome that is associated to transitions from successive states [Luc95].

Because speech signal representations correspond naturally to two-dimensional objects; one of the dimensions representing time, and the other, frequency, a class of two-dimensional HMMs called a 2D Extended HMM is derived [Li00]. This class of processes models the state duration explicitly. In extending the dependency of observation densities to current, preceding, and subsequent states, local smoothing is applied to act on mean energies associated with successive jumps of states.

A 2D Extended HMM differs from a traditional HMM by using a double stochastic array to represent its parameter space

\[ \{Y_{tf}, X_t; t=1,...,T, f=1,...,F\} \] (2.37)

where \( \{Y_{tf}\} \) is the observation process and \( \{X_t\} \) is the state process, to model the incoming speech signal. The state process \( \{X_t\} \) takes values on a finite and ordered state space and is assumed to be a left-to-right process. It replaces the Markov chain process of the usual HMM framework and focuses on the joint distribution rather than on state transition probabilities of the model. This joint distribution depends only on the visited states and the length of the visit.

The observation process \( \{Y_{tf}\} \) restricts the dependencies of each observation to the set of observations, given the state process \( \{X_t\} \), of its neighboring states, excluding itself. Thus, we assume that the observation process depends on both the current state and neighboring
observations and states. In this way, the model incorporates the high temporal correlation observed in speech signals.

2.4 Summary

In this chapter we looked at the process of building acoustic models used in speech recognition systems. We initially described a technique of template matching which predates the use of HMMs. We discussed how a prototype template was built from training data and how it then was used to decode unknown data using a technique known as Dynamic Time Warping.

After pointing out some shortcomings of this approach, we moved on to the current state of the art use of HMMs in modern speech recognition systems. We gave a detailed description of how HMMs are used to model acoustically the speech signal, from the simple discrete form to the more complex yet important continuous form. We derived equations for both training these models in the form of the dynamic programming approach of Baum-Welch, as well as decoding them using the Viterbi algorithm.

We also discussed methods of improving upon the simple continuous model of a phoneme by adding a mixture of Gaussians to better discriminate among the input features as well as adding context to differentiate between closely related models. Because these techniques increase the number of parameters significantly, techniques of sharing information via the tying of Gaussian mixtures densities across acoustically similar models was also covered.

This was followed by a discussion of the architecture of acoustic model building for two current state of the art speech systems, Byblos and ISIP. Byblos, a research system developed by BBN Technologies, is a full fledged HMM based recognition system, performing among the top tier in systems to date. It offers both robust model building techniques as well as fast
decoding strategies to achieve high performance in both accuracy as well as speed. The ISIP system, developed by Mississippi State University, although not quite as good compared with Byblos, shares many of the same fundamental features. We will discuss the ISIP system further in subsequent chapters as it is the base recognition system that all research work in this dissertation is based on.

Finally, we finished by describing several variations on the use of HMMs in today's speech recognition systems. All these variations looked at extending the power of an HMM by adding flexibility to the model, either through the addition of new sets of features, development of dependencies among features, or creation of additional relationships between existing features. In the next chapter we introduce another such model, called a Partially Observable Markov Decision Process (POMDP). This model adds currently unexplored relationships among features, which allows us to look at speech recognition in new ways as well as providing a good theoretical framework with which to describe the speech process as a whole.
Chapter 3

POMDPs and Speech Recognition

In this chapter we will look at a Partially Observable Markov Decision Process (POMDP) [Mon82] [Lov91]. Traditionally POMDPs have been applied to planning problems, including in particular robot navigation within buildings [Cas94] [Cas96] [Dea95] [Sim95] [Sha97a] [Sha97b] [Sha98]. We look at a radically different use for a POMDP, as a framework in building acoustic phonetic models for a speech recognition system.

For over a decade, an HMM has been the primary tool used for acoustic modeling in the field of speech recognition. As we saw in the previous chapter, an HMM solved various problems that the previous technique, template matching, had in its approach. Just as an HMM can be seen as a generalization of a template, the POMDP is a generalization of an HMM. We will use a POMDP to model the base phonetic unit.

We introduce the concept of multiple phonetic context classes, one for each of the infinite possible contexts a phoneme can be in, and show how a POMDP can be used to represent
CHAPTER 3: POMDP AND SPEECH RECOGNITION

such a model. Much the same way that tying mixtures at the state level across phonemes sharing linguistic properties is used to fill in gaps in the model space due to lack of data, the POMDP model can fill in additional gaps, in effect adding a second level of clustering driven by the data itself.

More interestingly, we will explore the idea of cross context classes, whereby a model can switch to a different context class while transitioning through the state space of a particular phoneme during feature alignment. This is a departure from the many variations that have been applied to HMMs in the past. Before we delve into the specifics of these acoustic modeling techniques, we first describe in detail the basics of POMDPs.

3.1 Markov Decision Processes

POMDPs extend Markov Decision Processes (MDP), so we examine MDPs first. An MDP is a mathematical formalization of a problem in which a decision maker, an agent, must decide which actions to choose that will maximize its expected reward as it interacts with its environment. It is assumed that there is no uncertainty about what state an agent finds itself in, i.e., its environment is completely observable. There is, however, unpredictability in the effect of its chosen actions [Sut90] [Koe92] [Put94] [Lit95a].

More formally, an MDP is defined as a model $\lambda = \langle S, D, A, R \rangle$, where

- $S = \{s_0, \ldots, s_{N-1}\}$ is a finite set of $N$ states.
- $D = \{d_0, \ldots, d_{K-1}\}$ is a finite set of $K$ actions (decisions).
- $\{A^1, \ldots, A^K\}$ are state transition probability distribution matrices for $K$ actions where $a^k_{i,j}$ is the probability of going from state $i$ to state $j$ with action $k$.

Formally, $a^k_{i,j} = P(q_{t+1} = s_j \mid p_t = d_k, q_t = s_i)$; $q_t$ is the state at time $t$ and $p_t$ is the action at time $t$. 

• \( \{R^1, ..., R^K\} \) are reward vectors for \( K \) actions where \( r^k_i \) is the reward for taking action \( k \) in state \( i \).

With an MDP, the next state depends only on the action that was taken and the state we came from. Earlier states or actions have no impact. This is known as the Markov property. The reward at each state depends on the action taken.

The object of an agent is to choose a sequence of actions that maximizes its expected rewards. A policy maps each state to an action, the action to take from that state. An optimal policy yields the maximum expected reward from each state. In the next section we discuss the concept of a policy in more detail and derive a set of equations to compute an optimal policy.

### 3.1.1 Solving an MDP

A solution to an MDP is an optimal policy. Given a policy, we can evaluate it based on the long-run value that the agent expects to gain from executing that policy [Kae98].

We look at two frameworks that are used to find such policies. With a finite-horizon model the agent acts in order to maximize the expected sum of reward that it gets on the next \( k \) steps, thus maximizing

\[
E \left[ \sum_{t=0}^{k-1} r_t \right]
\]

where \( r_t \) is the reward received on time step \( t \).

However, in many situations there is no fixed finite horizon. An alternative approach is the use an infinite-horizon discounted model [Son78] [Pla81], in which the sum of rewards
over the infinite lifetime of the agent is discounted geometrically using a discount factor \( 0 < \gamma < 1 \), thus maximizing

\[
E\left[ \sum_{t=0}^{\infty} \gamma^t r_t \right] 
\]  

(3.2)

Because of the discount factor, rewards received earlier in its lifetime are more valuable to the agent. The larger the discount factor, the more effect future rewards have on the agent’s current decisions. For the remainder of this discussion, we will focus on the infinite-horizon discounted model. This material is taken largely from [Kae98].

A policy describes an agent’s behavior. We define a policy, \( p : S \rightarrow D \), as a state-action mapping specifying, for each state the agent is in, the action to be taken. The action chosen depends only on the current state. No previous state information is necessary.

Next, we define \( V_p(i) \) to be the expected discounted sum of reward gained from starting in state \( i \) and executing policy \( p \) recursively

\[
V_p(i) = r^{p(i)}_i + \gamma \sum_j a^{p(i),j} V_p(j) 
\]  

(3.3)

Equation 3.3 computes the value function \( V_p(i) \) given a policy \( p \). We now want to identify the policy \( p \) that maximizes the value function \( V_p(i) \), designated as \( p^* \). The value function for this policy, \( V_{p^*} \), written as \( V^* \), is the solution to

\[
V^*(i) = \max_d \left[ r^d_i + \gamma \sum_j a^{d,j}_i V^*(j) \right] 
\]  

(3.4)
CHAPTER 3: POMDP AND SPEECH RECOGNITION

This then defines the criteria for an optimal policy for an MDP. Two common techniques for identifying an optimal policy are \textit{value iteration} \cite{Bel57} and \textit{policy iteration} \cite{How60}. We do not discuss them further in this work since they play no role in the speech recognition setting.

3.2 Partially Observable Markov Decision Processes

MDPs are \textit{fully observable}—the agent always knows precisely what state it is in. However, if the agent cannot determine its state, its world is said to be \textit{partially observable}. In a partially observable world, an agent is unable to act on an MDP policy since it does not know the current state information. For such situations, we use a generalization of MDPs, called a Partially Observable Markov Decision Process (POMDP). In a POMDP, the agent does not know the current state precisely, When it acts, the state transition is as with MDPs. However, after acting it receives an observation that gives partial information about the current state \cite{Lit95b} \cite{Lit95c} \cite{Meu99}.

The material in this section is taken largely from \cite{Lit94} \cite{Kae98}. We formally define a POMDP as a model $\lambda = <S, \Theta, D, A, B, R, \pi>$, where:

- $S = \{s_0, ..., s_{N-1}\}$ is a finite set of $N$ states.
- $\Theta = \{o_1, ..., o_T\}$ is a set of $T$ observations.
- $D = \{d_0, ..., d_{K-1}\}$ is a finite set of $K$ actions (decisions).
- $\{A^1, ..., A^K\}$ are state transition probability distribution matrices for $K$ actions, where $a_{i,j}^k$ is the probability of going from state $i$ to state $j$ with action $k$. Formally, $a_{i,j}^k = P(q_{t+1} = s_j | p_t = d_k, q_t = s_i); q_t$ is the state at time $t$ and $p_t$ is the action at time $t$. 

• $\{B_1, ..., B^K\}$ are observation symbol probability distribution matrices for $K$ actions, where $b_{kj}(o)$ is the probability of observation $o$ in state $j$ with action $k$. Formally, $b_{kj}(o) = P(v_t = o_t \mid p_t = d_k, q_t = s_i); v_t$ is the observation recorded at time $t$.

• $\{R_1, ..., R^K\}$ are reward vectors for $K$ actions where $r_{ki}$ is the reward for taking action $k$ in state $i$.

• $\pi$ is the initial state distribution, where $\pi_i$ is the probability of starting in state $i$.

Note that with a POMDP there are two stochastic processes, one being the state transition and the other the generation of an observation. In these ways, both a POMDP and an HMM are similar.

As with an MDP, the goal remains the same: to find an optimal policy. In the section that follows we reformulate our equations for the optimal policy of an MDP to work with a POMDP. We introduce the concept of a belief state that gives a distribution over all states of the likelihood of being in each state.

### 3.2.1 Solving a POMDP

Starting with equation 3.4, we now have the additional constraint of not knowing which state we are in. We define $B$ as a belief state, where $B(j)$ is the probability of being in state $j$ given the history of observations made and actions taken. The initial belief state is just $\pi$. To update a belief state $B$ after taking action $d$ and observing $o$, we get a new belief state $B'$:

$$B'(j) = P(j \mid o, d, B) \quad (3.5)$$
Applying Bayes’ rule we rewrite equation 3.5 as follows

\[ B'(j) = \frac{P(o \mid j, d, B) P(j \mid d, B)}{P(o \mid d, B)} \]  

(3.6)

To expand the equation fully, we look at each part of equation 3.6 separately. In the numerator, the first term, \( P(o \mid j, d, B) \), represents the probability of getting observation \( o \) in state \( j \) after action \( d \). Here, we can ignore \( B \) since our model \( \lambda \) gives us:

\[ P(o \mid j, d, B) = b^d_j(o) \]  

(3.7)

i.e., the probability of getting observation \( o \) from state \( j \) after action \( d \).

Next, we take the second term of the numerator, \( P(j \mid d, B) \), which represents the probability of entering state \( j \) with action \( d \) from belief state \( B \). To calculate this, we calculate the probability of going to state \( j \) from every possible state, weighted by the likelihood of starting in that state, and summing the result.

\[ P(j \mid d, B) = \sum_i a^d_{ij} B(i) \]  

(3.8)

Finally, we take the term in the denominator, \( P(o \mid d, B) \), which represents the probability of making observation \( o \) after taking action \( d \) from belief state \( B \). Using the law of total probability we get:

\[ P(o \mid d, B) = \sum_j P(o \mid j, d, B) P(j \mid d, B) \]  

(3.9)
Using both equations 3.7 and 3.8, we substitute to get

\[
P(o \mid d, B) = \sum_j \left( b_{dj}(o) \sum_i a_{ij} B(i) \right)
\]

Substituting each of the above expanded terms into equation 3.6 gives us our final equation for belief state

\[
B'(j) = \frac{b_{dj}(o) \sum_i a_{ij} B(i)}{\sum_j (b_{dj}(o) \sum_i a_{ij} B(i))}
\]

Finally, we take the equation for the optimal expected value from our MDP (3.4) and replace state \( i \) by belief state \( B \), to derive the optimal value of a belief state.

\[
V^*(B) = \sum_i B(i) \, V^*(i)
\]

noting that we sum over all states \( i \), weighted by \( B(i) \), where in equation 3.4 we assumed a specific state.

From this we can derive an optimal policy \( p^* \) for a POMDP as a function of belief states. There are several techniques for calculating an optimal policy, such as the Witness algorithm [Lit94] and policy iteration [Han98]. However, we do not discuss them as they are not relevant to our dissertation.

### 3.3 Modeling speech with a POMDP

A Partially Observable Markov Decision Process (POMDP) can be viewed as a generalized form of a Hidden Markov Model [Sha99]. It differs from an HMM in several important ways.
Instead of having a single transition between any two states, a POMDP allows for multiple transitions between the same two states, where each transition represents an action—hence we refer to it as a “decision process.” In addition, a POMDP associates a reward value with each action/state pair. Conversely, we can describe an HMM simply as a POMDP with a single action, i.e. with no decision process, and no reward.

An untried approach is to use a POMDP in the speech recognition problem by having it model a phonetic unit of speech: the phoneme. As with an HMM, a POMDP's states represent the pronunciation task: beginning, middle, end of phoneme where the observed acoustics features are associated with each state. The randomness in the state transitions still accounts for time stretching in the phoneme: short, long, hurried pronunciations. The randomness in the observations still accounts for the variability in pronunciations.

Additionally, the multiple transitions between two distinct states now represents different contexts a phoneme finds itself in. Context represents the association of the phoneme to the set of phonemes to its immediate left, immediate right, or both left and right. A phoneme in context is modeled in all its different contexts by a single POMDP, where each action of the model represents a different context a phoneme can find itself in. The reward value of a POMDP model represents weights used to combine these different contextual phonetic models.

### 3.3.1 POMDP model of a phoneme

As with HMMs, a POMDP model consists of multiple states, moving in a left-to-right direction, with connections between consecutive states, skips over states, and self-loops. Figure 3.1 illustrates what a 5 state POMDP looks like.
The above POMDP models a phoneme in all possible contexts of its left and/or right neighboring phonemes. These phonemes in context are traditionally modeled in speech using individual HMMs, one for each phoneme context.

We use the actions of a POMDP to represent all possible contexts a phoneme finds itself in. In theory, this would encompass an infinite number of actions, as there are theoretically an infinite number of possible contexts a phoneme can be in. In practice we model only monophones having no context, left biphone having left context, right biphone having right context and triphone having both left and right context. Adding the constraint that a path through the model requires that the action used to leave a state must match the action taken when that state was entered, insures that a POMDP models the exact same context phonemes that an HMM model does. Relaxing this constraint will be an important feature that a POMDP provides that is not possible with an HMM and will be discussed in more detail when discussing the various decoding strategies used when searching our newly formed POMDP model.

### 3.4 Training POMDP models

Building acoustic POMDP models will take a very familiar approach. Our POMDP models are constructed from a collection of HMM models that share the same center
We identify each context by a different action in our model and add constraints to those actions when using the model.

We define a context class as a collection of context dependent models that share the same type of context. In this paper, for practical purposes, we limit ourselves to the following set of four context classes: monophone, left biphone, right biphone and triphone.

We train each context class (monophone, left and right biphone, triphone) independently. We use the same training data on each individual context class, using the Baum-Welch training algorithm. The individual HMMs of all context classes are then combined into a single, unified model, representing our POMDP. Combining the different models is done by labeling each transition and associated mixture density of every model by its specific context and context class. All model transitions and Gaussian mixtures belonging to a specific context class are kept independent of other context classes. Tying of mixtures and their weights, as discussed in chapter two, only occurs within each context class, not across context classes.

### 3.5 Searching POMDP models

As with HMMs, our goal is to find what was said given a set of input features. We aim to find the highest likelihood path through our model. We apply a modified Viterbi algorithm to search the space of our POMDP models, incorporating rewards as a weighting mechanism when different context classes interact. These rewards will be incorporated through the language model score, P(W), at each phoneme boundary, or at each state, depending on the decoding strategy used.

Properly constraining our POMDP models will be an important aspect of insuring that our search space does not grow out of control. We constrain our model by limiting the set of actions our model can choose from as we transition through it. These limiting constraints will
be based on context. In order to better understand the different types of constraints, we will discuss various context scenarios.

### 3.6 Modified Viterbi

With our POMDP model, we are able to model a particular phoneme in all possible contexts, including all possible classes of context. Our POMDP model uses actions to specify specific contexts. For now we constrain our model so that once an action is chosen when we enter the model, we must pick that same action (i.e. the same context) as we traverse through the model.

We define a *cross context* model set as a tuple made up of context models that all share the same *partial context*. Partial context is defined by the following rules:

- a triphone and left biphone share a partial context if they share the same left context and center phoneme,
- a triphone and right biphone share a partial context if they share the same right context and center phoneme,
- a monophone shares the same partial context with either a triphone or left and right biphone by sharing the same center phoneme.

Having a model that now contains not only all occurrences of a particular context class, but also multiple context classes, we modify our search of the model to utilize this additional information. We use the Viterbi algorithm as a starting point and modify it to fit within our model’s additional complexity.

Several strategies will be explored in creating a modified Viterbi algorithm. The basic approach extends the search space to include all context classes by expanding all cross
context model sets in place of each original model. Since under-training of models can be a problem, especially with small data sets, augmenting our search space by including all context class models may strengthen overall the ability to match acoustically correctly to the input signal.

One method in combining all context classes will use a general weighting factor per context class to balance the impact of each class on our search. This weighting factor represents a reward in our POMDP model that we apply upon entering the model. We call this modified Viterbi a Uniform Mixed Model Viterbi (UMM Viterbi).

A second method will follow a similar technique used in language modeling when combining mixed classes of ngram models [Kat87]. In this approach we use estimates of context class weights for each model to combine models from the different context classes. We use frequency counts of the occurrences of each particular context class’s model to weight it relative to the other models. We call this modified Viterbi a Weighted Mixed Model Viterbi (WMM Viterbi).

A third method will not only combine all context classes, but also relax the constraint prohibiting transitions between context classes. In HMM parlance, this will allow transitions across models, or “jumps,” something that has not been explored before. This relaxation will only allow transitioning between phonemes sharing the same partial context. As with WMM Viterbi, we use frequency counts of the occurrences of each model to weight the cross model transitions among the partial context models. We call this modified Viterbi a Cross-context Mixed Model Viterbi (CMM Viterbi).

3.6.1 **Uniform Mixed Model Viterbi**

In this approach, we simply add to the mix all context classes and allow Viterbi to choose the best path through the entire lattice of possibilities. As Viterbi expands all possible context
models, guided by both the dictionary and the language model, we relax the context rules by matching up all partial context phones. This, in effect, will wild-card all monophones to match up with all biphones and triphones that use the same center phone and all biphones to match up with those triphones whose other context they share. The Viterbi algorithm itself remains unchanged from equations 2.30-2.31, only the number of models it sees grows by the addition of the multiple context classes, and the relaxation of the context rules increases the number of hypotheses explored.

Figure 3.2 illustrates the expansion of partial context phonemes. Notationally, the phoneme to the left of a “-” is the left context phoneme and the phoneme to the right of a “+” is the right context phoneme. The center phoneme is either to the right of a “-“ or to the left of a “+” or both. Where in the traditional Viterbi algorithm, we focus on a particular triphone (a), we now also include the partial context classes of the associated monophone, right and left biphones, as well as their corresponding expansions (b).

Figure 3.2: (a) standard Viterbi and (b) UMM Viterbi expansion of “tomato” having two phonetic spellings “t-ow-m-ey-t-ow” and “t-ow-m-aa-t-ow”
CHAPTER 3: POMDP AND SPEECH RECOGNITION

We apply a weighting factor, \( W^c \), to each of the context classes, \( c \), to balance the impact each class has on the entire search space. The weighting will discount, as a percentage, under-trained models as well as less discriminating but well-trained models, and is applied to the model upon entering its start state. These weights are incorporated by adjusting the normal language model scores, \( P(W) \), at each phoneme boundary. This will enable fine-tuning of effects of the interaction between different classes.

The global weighting factor for each context class, \( W^c \), will be bounded within the following interval

\[
0 < W^c < 100\% \quad (3.16)
\]

Finding the optimal global weighting factor for each of the associated context classes requires iteratively narrowing in on the best weighting combination among those classes. In chapter four, we will discuss in detail the experimentation techniques used in identifying a set of weights that produce optimal decoding results.

### 3.6.2 Weighted Mixed Model Viterbi

A more sophisticated strategy in decoding our new models is to combine all models as in the UMM Viterbi approach but individually weight each model belonging to a particular context class relative to the frequency it appears in the training set of that context class. The idea is similar to techniques used in developing language models, whereby frequency models of word occurrences are derived from a representative data set to generate probabilistic models of word pairs and word triplets.

To generate the relative weighting, we take a statistical summary of the training set relative to each context class, \( c \), and generate a weight, \( w^c_m \), for each phonetic context model, \( m \), in that
context class based on each model’s frequency count, \( f^c \). This weight is again applied as a percentage to the model upon entering its start state. Since small data sets tend to have many unseen models, we apply a lower threshold weight, \( L^c \), for all unseen models.

Additionally, to compute a model’s weight, we take the frequency count of the specific model and normalize it by a constant \( K^c \). The value of our constant \( K^c \) is arbitrary and can differ for each context class.

Finally, we apply an artificial upper bound, \( W^c \), to each context class which represents the class weight we applied in our UMM Viterbi algorithm. We construct our individual phonetic context model weight as follows:

\[
 w^c_m = L^c + \min\left(\frac{f^c_m}{K^c}, 1\right) \times (W^c - L^c)
\]  

(3.17)

The \( \min \) function in our equation insures that our weight remains within the bounds of \( L^c \) and \( W^c \), since \( K^c \) can be smaller than \( f^c \). This weight is included in the overall likelihood score as we pass through the respective model.

As with the UMM Viterbi decoding method, the Viterbi algorithm itself remains unchanged from equations 2.30-2.31. Its model space, again, grows by the number of multiple context classes added to the search and the relaxation of the context rules increases the number of hypotheses explored. Also, as with UMM, the weights are incorporated by adjusting \( P(W) \).

### 3.6.3 Cross-context Mixed Model Viterbi

In both UMM and WMM Viterbi, we enforced the strict constraint that as we go through the model, the action representing the context is kept constant throughout the model. By relaxing this constraint and enabling the switching to actions of other context models, we
increase our search space. We thus utilize the full power of the POMDP model, in effect, further increasing the search space by creating additional branches at each state. To insure that we do not degrade our model by washing it out with non-related information from other models, we carefully choose how to relax these constraints, choosing only contexts belonging to the same cross context model set.

Now, at each time step, we can choose between all cross context models and transition to any of them. We will apply several strategies in choosing the possible transitions, all limited by the pruning threshold of our general Viterbi search. We apply the same weighting factor, $w^c_m$, used in our WMM Viterbi, implemented by adjusting $P(W)$. But this time we apply it at each state as we transition to another state, and perhaps to another model. The weight is added into the maximum likelihood score of each transition.

What we have added, in essence, is a set of “jump” transitions to our traditional HMM model that consumes no time and no observations. Along with skips and self-loops, our model can now jump to a constrained set of other models at each state. We add the ability to weight these jumps to allow for tuning of our model to the data.

Figure 3.3 illustrates how jump transitions would appear between cross context models for the triphone “t-ow+m”. The dashed lines represent the jump transitions. In the next section we discuss various strategies for adding weights to these transitions. Referring to Figure 3.1, it is easily seen how a POMDP incorporates jumps in its natural topology.
Our Viterbi equations from 2.30-2.31 now have to incorporate the addition of these jumps. We do this by incorporating the actions of our POMDP model into the equation. We now look to maximize $P(I \mid D, O, \lambda)$ over both the state sequence where the action sequence $D_1, ..., D_t$ are the actions taken at each point where $D_t$ is the action taken from state $I_{t-1}$. We derive our new Viterbi equation from this, starting with Bayes’ rule:

$$\max_{I, D} P(I \mid O, D, \lambda) = \max_{I, D} \frac{P(O \mid I, D, \lambda) P(I \mid D, \lambda)}{P(O \mid D, \lambda)}$$  \hspace{1cm} (3.18)$$

Which is equivalent to:

$$\max_{I, D} P(I \mid O, D, \lambda) = \max_{I, D} P(O \mid I, D, \lambda) P(I \mid D, \lambda)$$  \hspace{1cm} (3.19)$$

Which is equivalent to:

$$\max_{I, D} P(I \mid O, D, \lambda) = \max_{I, D} P(O, I \mid D, \lambda)$$  \hspace{1cm} (3.20)$$
Which is equivalent to:

\[
\max_{I,D} P(O, I \mid D, \lambda) = \max_{\{I_1, \ldots, I_n, D_1, \ldots, D_n\}} P(O_1, \ldots, O_n, I_1, \ldots, I_n \mid D_1, \ldots, D_n, \lambda) \tag{3.21}
\]

We use the chain rule again for \( O_t \), expanding equation 3.21 to:

\[
= \max_{\{I_1, \ldots, I_n, D_1, \ldots, D_n\}} P(O_1, \ldots, O_{t-1}, I_1, \ldots, I_{t-1} \mid D_1, \ldots, D_{t-1}, \lambda) P(O_t \mid O_1, \ldots, O_{t-1}, I_1, \ldots, I_{t-1}, D_1, \ldots, D_{t-1}, \lambda) \tag{3.22}
\]

We reduce the second term in equation 3.22, since the observation at time \( t \), \( O_t \), only depends on the state and action at time \( t \):

\[
= \max_{\{I_1, \ldots, I_n, D_1, \ldots, D_n\}} P(O_1, \ldots, O_{t-1}, I_1, \ldots, I_{t-1} \mid D_1, \ldots, D_{t-1}, \lambda) P(O_t \mid I_t, D_t) \tag{3.23}
\]

Using the chain rule again, we extract \( I_t \) from the first term, and reduce it since the state at time \( t \) only depends on the previous state and current action. We thus get:

\[
= \max_{\{I_1, \ldots, I_n, D_1, \ldots, D_n\}} P(O_1, \ldots, O_{t-1}, I_1, \ldots, I_{t-1} \mid D_1, \ldots, D_{t-1}, \lambda) P(O_t \mid I_t, D_t) P(I_t \mid I_{t-1}, D_t) \tag{3.24}
\]

Finally, dropping from the first term the action at time \( T \), \( D_n \), since we are only in state \( I_{t-1} \), we get:

\[
= \max_{\{I_1, \ldots, I_n, D_1, \ldots, D_{t-1}\}} P(O_1, \ldots, O_{t-1}, I_1, \ldots, I_{t-1} \mid D_1, \ldots, D_{t-1}, \lambda) P(O_t \mid I_t, D_t) P(I_t \mid I_{t-1}, D_t) \tag{3.25}
\]
The first term in equation 3.25 represents $d_{t-1}(I_{t-1})$. The second and third terms represent the observation and transition probabilities, respectively. From this we can formulate our recursive equations for CMM Viterbi, similar to equations 2.30-2.31 for the original Viterbi. However, we only need maximum $P(O, I \mid \lambda)$—we do not need the state and action sequence that led to it.

We initialize $\delta$ as follows:

$$\delta_1(j) = \max_d \{ \pi_j b^d_j(O_1) \}$$  \hspace{1cm} (3.26)

and compute $\delta$ up to time $T$ recursively:

$$\delta_{t+1}(j) = \max_{i,d} \{ \delta_t(i) a^{i,j} \} b^d_j(O_{t+1})$$ \hspace{1cm} (3.27)

The maximum $P(O, I \mid \lambda) = \max_j \delta_T(j)$.

The complexity of Viterbi now changes from $N \times T$ operations to $N \times T \times K$. However, since $K$ is relatively small, performance does not suffer tremendously.

### 3.6.3.1 CMM weighting strategies

Adding weights to the jump transitions gives us the ability to control further the behavior and improve the performance of our POMDP model. As discussed above, we use the same weighting factor, $w^c_m$, that was generated for the WMM Viterbi algorithm. How we apply these weights, gives us additional flexibly in how well we are able to tune our model to achieve optimal performance.
We introduce two different approaches for integrating weights into our search. In “maximum cross context,” for each cross context model, we add that model’s weight to the likelihood score, and transition to the context model that yields the highest likelihood score. This then will choose an optimal path through the constrained set of cross context models for each context dependent POMDP model.

A second approach will again add each cross context model’s weight to the likelihood score, but instead of choosing the transition with the highest likelihood score, all transitions are taken. In this weighting scheme, which we call “expanded cross context,” we allow Viterbi itself to choose the optimal path through all the weighted cross context models, therefore adding the fully expanded POMDP model to the search.

Additionally, we add the option to “restrict” each of the above two weighting techniques. The restrictions is that once a lower order context model is selected, only models of equal or lesser order can be chosen. Therefore, a triphone model can choose between all cross context models, a biphone model can choose from one of the two types of biphone models, or drop down to a monophone model, and a monophone model can only choose to stay within its context. This, in effect, causes the search to down shift as it transitions through the model, the idea being that once we decide on a lower order cross context model, we did so because the higher order context model was deemed insufficiently trained.

3.7 Summary

In this chapter we explored the constructs and uses of a Partially Observable Markov Decision Process. We first discussed a closely related Markovian process, a Markov Decision Process, and discussed its properties. We introduced the notion of finding an optimal policy, and developed an equation to solve for that optimal policy.
Next, we defined a POMDP, which is an MDP whose underlying state sequence is hidden, thus making its environment partially observable. We introduced the concept of a belief state, and constructed equations to calculate it. A belief state represents a probability distribution over the state space, giving us an idea of which states we may likely be in. We then converted the equation for finding an optimal policy for an MDP, to that of a POMDP by replacing actual state with belief state.

We introduced the main idea of this dissertation by developing a POMDP based phonetic context model to represent the acoustic speech signal. The topology of such a model, as well as how it relates to an HMM was discussed. We then described how such a model would be built, using tried and true techniques from the development of HMM models.

Finally we covered how such a model would be used in decoding utterances, the primary goal in speech recognition. We developed modified versions of the Viterbi algorithm and discussed each in detail. Three different decoding strategies were thus introduced: Uniform Mixed Model Viterbi, Weighted Mixed Model Viterbi and Cross-context Mixed Model Viterbi.
Chapter 4

Results

In this chapter we will discuss experiments applied to our newly formed POMDP model. In order to determine how well our new model performs, it is important to put it through its paces, rigorously exploring its strengths as well as its weaknesses. Developing both a sound experimentation system, as well as a good strategy of experiment design is important in accurately testing speech models.

We will describe the infrastructure developed for running experiments, as well as the various strategies used to test our new model on the three different decoding methods presented in chapter 3. As speech requires a tremendous amount of compute resources, it is important to develop sound strategies to avoid wasting those resources on frivolous experiments. This is especially important when working with a limited number of resources in a university environment.
CHAPTER 4: RESULTS

State of the art speech recognition research is presently done by sites that have a vast array of powerful computers running non-stop in parallel. Although this is a major advantage when devising an exhaustive list of experiments to test conceptual ideas, significant work can also be done on a limited basis with only limited resources. In the latter case, it is therefore important to devise carefully experimental strategies that have a high probability of yielding new information. For this work, this held paramount importance in guiding the experimental framework.

4.1 PASS system

A standard experiment scripting language, Programmable Adjustable Speech System (PASS), was developed that allowed for easy specification of the various parts of a speech recognition system [Car01]. It allows one to easily define sequence of steps that need to be executed in an ordered manner, and it allows for the introduction of system wide constants that can be applied to both local instances of script files, as well as across multiple script files. We demonstrate the flavor of such a script file, or sequence file as they are called, by an example illustrated in figure 4.1 below.

```plaintext
; POMDP_decode.seq
;
; Sequence file that decodes and scores a POMDP via WMM Viterbi
;
; definitions for type of decoding we do
;
def +POMDP_TYPE+       WMM

def +DEC_TYPE+          ngram_decoding

def +DEC_CONTEXT+       cross_word
;
; create POMDP model
;
10 build_pomdp +type+ +POMDP_TYPE+ +context+ +DEC_CONTEXT+
;
; run decode and score results
;
20 run_isip_decode +type+ +DEC_TYPE+ +context+ +DEC_CONTEXT+
30 run_isip_eval +score_mode+ Isip_word
```

Figure 4.1: a sample sequence file for decoding
This scripting language not only enabled the development of robust experiment strategies, but it also provided a self documenting experimental logbook that could be easily searched.

4.2 ISIP system

Research in the field of speech recognition has changed significantly in the past several years. No longer are researchers required to develop speech systems from the ground up. There are now several well designed speech recognition system, along with their source code, freely available in the public domain. Although the HTK toolkit, developed by Cambridge University, is considered one of the better research systems, the ISIP system, developed by Mississippi State University, was freely available first.

An important feature when working with any complex system is how well organized its code base is. This is especially true when dealing with a sophisticated search algorithm such as Viterbi. The ISIP system uses a mix of C and C++ to implement the various parts of its speech recognition engine. It develops base classes for the important data models and then utilizes them in a procedural manner. We found this design invaluable when faced with the task of modifying the Viterbi decoder to add in our new POMDP model and the various decoding methods that utilized this model.

We used version 5.10 of the ISIP prototype speech recognition system. Our main focus was on modifying the decoder component of that system, trace_projector, as well as the associated class libraries that it used. We tested our final modified decoder by running a POMDP model that represented a traditional HMM (i.e. it used only one context class) and compared it to runs of the unmodified decoder using an equivalent HMM model. We ran this test, individually for all available context classes (triphone, left biphone, right biphone,
monophone) that we used in building our POMDP models. We checked for and indeed received identical answers between the single context POMDP model and the equivalent HMM model as an initial determination of the accuracy of our code changes.

4.3 Experiments

In setting up our experiment structure we first explored several corpora to use in testing our new model. We decided on two small data sets that were readily available and well suited for our task. Both provided the ability to create models in a reasonable amount of time with the resources at hand, and a rich set of acoustic features that we could apply in developing our models.

We chose TIMIT as our primary research corpus as it was of small enough size that fit well into our resource configuration, but significant enough in size to generate useful results. It is also a corpus that much research has already been based upon, enabling us to relate our performance to work done elsewhere. We also chose TIDigits, quite simply because it was a small, yet rich data set that allowed for fast model building and relatively fast decoding, enabling us to more quickly try various strategies without great loss of critical compute resources.

We used several tools in both developing our train and test corpora and in evaluating their performance on the many variations of decoding. For constructing language models, version 2.0 of the CMU-Cambridge Statistical Language Modeling Toolkit was applied. For evaluating recognition performance, version 1.2 of the standard NIST Scoring Toolkit was used.

We will discuss the makeup of each of these two corpora in more detail in the next few sections.
4.3.1 TIDigits corpus

TIDigits is a corpus made up of sequences of digits spoken in random order and at random length by a collection of speakers. Its dictionary is composed of 11 words, comprised of the digits '1' through '9' as well as two representations for '0' (zero and oh). Its phoneme set is made up of a subset of standard English, consisting of exactly 18 unique phones. Table 4.1 lists the phoneme set used by TIDigits.

<table>
<thead>
<tr>
<th>Phone</th>
<th>Example</th>
<th>Phone</th>
<th>Example</th>
<th>Phone</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ah]</td>
<td>but</td>
<td>[iy]</td>
<td>beat</td>
<td>[t]</td>
<td>ten</td>
</tr>
<tr>
<td>[ay]</td>
<td>buy</td>
<td>[k]</td>
<td>kick</td>
<td>[th]</td>
<td>thick</td>
</tr>
<tr>
<td>[eh]</td>
<td>bgt</td>
<td>[n]</td>
<td>net</td>
<td>[uw]</td>
<td>beauty</td>
</tr>
<tr>
<td>[ey]</td>
<td>ate</td>
<td>[ow]</td>
<td>boat</td>
<td>[v]</td>
<td>yet</td>
</tr>
<tr>
<td>[f]</td>
<td>fat</td>
<td>[r]</td>
<td>rat</td>
<td>[w]</td>
<td>wet</td>
</tr>
<tr>
<td>[ih]</td>
<td>bit</td>
<td>[s]</td>
<td>set</td>
<td>[z]</td>
<td>zoo</td>
</tr>
</tbody>
</table>

Table 4.1: TIDigits phoneme set

The data, comprised of a set of utterances made up of recorded audio files and their associated text transcripts, was split up into a training data set and a development data set. The training data set is used to build models and the development data set is used to test those models.

4.3.1.1 TIDigits training set

The TIDigits training corpus is considered small by speech recognition's standards. Table 4.2 below gives a breakdown of the different attributes of the data. On average, each of the 55 speakers recorded approximately 17 utterances.
Additional characteristics of the training corpus shows the average duration of an utterance being approximately 1.25 seconds in length, each made up of about 3 words. Composition of utterances range from as few as a single word up to 7 words.

### 4.3.1.2 TIDigits development set

The TIDigits development corpus is made up of utterances and speakers not present in the training corpus. Since the training corpus was a relatively small data set, the development data set is proportionately larger than is normally the case. The rule of thumb is that the test set should usually be between 5% to 10% in size to that of the training set. However, with only 20 minutes of training data, choosing a 10% portion would only yield 2 minutes for purposes of testing, far too small to obtain usable results. Table 4.3 lists the statistics of the development set.

<table>
<thead>
<tr>
<th>TIDigits development corpus</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of utterances</td>
<td>336</td>
</tr>
<tr>
<td>Number of speakers</td>
<td>20</td>
</tr>
<tr>
<td>Corpus length in minutes</td>
<td>9</td>
</tr>
</tbody>
</table>

Table 4.3: TIDigits development corpus statistics
As with training, each speaker recorded on average 17 utterances, each of length of approximately 1.5 seconds. The mean and the standard deviation on the number of words in the development set matches to that of training.

We chose to build a standard trigram language model with backoff using the full set of transcriptions from training. The language model was not augmented with any other data. However, we initially constructed a finite state grammar as the language model for TIDigits, allowing any combinations of digits of any length. After some initial tests, we discovered that performance suffered significantly over the trigram language model, increasing overall word error rates by at least 10% on all models tried.

### 4.3.2 TIMIT corpus

The TIMIT corpus is a well balanced data set made up of read text of English sentences whose composition was designed to insure good coverage among all phonetic context models. Table 4.4 lists the TIMIT phoneme set along with examples of how each is pronounced.

<table>
<thead>
<tr>
<th>Phone</th>
<th>Example</th>
<th>Phone</th>
<th>Example</th>
<th>Phone</th>
<th>Example</th>
<th>Phone</th>
<th>Example</th>
<th>Phone</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>[aa]</td>
<td>are</td>
<td>[ch]</td>
<td>chat</td>
<td>[g]</td>
<td>get</td>
<td>[n]</td>
<td>net</td>
<td>[l]</td>
<td>ten</td>
</tr>
</tbody>
</table>

*Table 4.4: TIMIT phoneme set*
The phoneme set consists of 45 symbols and is a slight variation on the standard English phoneme set specified by DARPA.

4.3.2.1 TIMIT training set

The TIMIT training corpus we used is the standard training set that comes with the TIMIT distribution. Table 4.5 gives a breakdown of the attributes of its data set.

<table>
<thead>
<tr>
<th>TIMIT training corpus</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of utterances</td>
<td>3869</td>
</tr>
<tr>
<td>Number of speakers</td>
<td>387</td>
</tr>
<tr>
<td>Corpus length in minutes</td>
<td>180</td>
</tr>
</tbody>
</table>

Table 4.2: TIMIT training corpus statistics

Each speaker on average spoke approximately 10 utterances, and each utterance lasted about 2.8 seconds in total. Since the TIMIT corpus is made up of read text of English sentences, the length of utterances ranged from 2 words to 18 words and on average consisted of 9 words.

The TIMIT training dictionary was constructed from all the words present in the training corpus, totaling in all 4,457 words.

4.3.2.2 TIMIT development set

The TIMIT development set was chosen from a standard TIMIT test set (DR1), assuring that no speaker was present in training. The size of the development set was chosen to be a large enough set to yield usable results but kept small enough to allow for reasonable
decoding times given the limited computational resources available. Even with careful consideration on the size of the test corpus, a single decoding run would require a minimum of 8 hours of processor time, though many lasted longer, up to twice that time. Table 4.6 lists the development corpus statistics.

<table>
<thead>
<tr>
<th>TIMIT DR1 development corpus</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of utterances</td>
</tr>
<tr>
<td>Number of speakers</td>
</tr>
<tr>
<td>Corpus length in minutes</td>
</tr>
</tbody>
</table>

Table 4.6: TIMIT development corpus statistics

Again, as in TIDigits, the TIMIT test corpus matched its train corpus in its compositions. Each speaker recorded on average 10 utterances, each lasting approximately 3 seconds. And the mean and the standard deviation on the number of words in the development set also matched that of training.

A trigram language model with backoff was constructed for TIMIT using all text transcriptions from the training set. Additional transcripts were collected from similar spoken English text found in the Oregon Graduate Institute’s National Cellular and Stories Corpora. Each of these corpora had a “story” section that required the speaker to record an arbitrary story of up to 10 seconds in length. The total number of text transcripts of the combined three corpora for building the TIMIT language model totaled 5,215 sentences consisting of 76,463 words. In all this increased the number of sentences by 35% and more than doubled the number of words from the standard TIMIT training corpus. Our decoding dictionary folded in words from all three corpora and contained a total of 7,835 words.
4.3.2.3 Baseline

Before being able to make determinations of how well a new model performs on any data set, a rigorous effort needs to be made to develop a set of robust baseline numbers. A baseline is important in that any gains made by newly developed work need to represent improvement over the best possible results attained before that work has been integrated. We therefore spent a good deal of our initial effort in finding a set of system parameters that yielded the best set of decoding results for both our corpora.

We did not attempt to optimize the analysis step and used the standard settings provided by the ISIP system. From the audio files of each corpus, we generated a standard set of Mel Frequency Warped Cepstral Coefficient (MFCC) files made up of 13 unique features along their first and second derivatives, using a frame duration of 10 milliseconds and Hamming window of 25 milliseconds, for a total of 39 features.

We generated baselines for 4 different HMM models (monophone, left biphone, right biphone and triphone) for each development corpus. We found optimal parameter settings for training and decoding on both corpora. For training, table 4.7 lists the important parameters for each corpus. We used a 3 state left to right HMM with self loops at each state but with no skip transitions.

<table>
<thead>
<tr>
<th>Training Corpus</th>
<th>no. states per model</th>
<th>beam width</th>
<th>state-tying threshold</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>split</td>
</tr>
<tr>
<td>TIDigits</td>
<td>3</td>
<td>500</td>
<td>100</td>
</tr>
<tr>
<td>TIMIT</td>
<td>3</td>
<td>2500</td>
<td>100</td>
</tr>
</tbody>
</table>

Table 4.7: ISIP model training parameters
For decoding, Table 4.8 lists the important parameters. The three different beam pruning thresholds help Viterbi determine how much information to keep at the state, model and word level.

<table>
<thead>
<tr>
<th>Decoding Corpus</th>
<th>LM scale factor</th>
<th>Word Penalty</th>
<th>beam pruning threshold state</th>
<th>model</th>
<th>word</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIDigits</td>
<td>12</td>
<td>0</td>
<td>400</td>
<td>400</td>
<td>400</td>
</tr>
<tr>
<td>TIMIT</td>
<td>9</td>
<td>-8</td>
<td>600</td>
<td>500</td>
<td>500</td>
</tr>
</tbody>
</table>

**Table 4.8: ISIP model decoding parameters**

Finally, our resulting baseline numbers for each of our development corpora on the different HMM models is given in Table 4.9 below.

<table>
<thead>
<tr>
<th>Corpus</th>
<th>WER</th>
<th>accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIDigits</td>
<td></td>
<td></td>
</tr>
<tr>
<td>triphone</td>
<td>60.7</td>
<td>42.8</td>
</tr>
<tr>
<td>left biphone</td>
<td>60.1</td>
<td>43.1</td>
</tr>
<tr>
<td>right biphone</td>
<td>56.9</td>
<td>46.2</td>
</tr>
<tr>
<td>monophone</td>
<td>98.1</td>
<td>9.1</td>
</tr>
<tr>
<td>TIMIT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>triphone</td>
<td>81.7</td>
<td>28.5</td>
</tr>
<tr>
<td>left biphone</td>
<td>78.5</td>
<td>27.8</td>
</tr>
<tr>
<td>right biphone</td>
<td>83.0</td>
<td>27.6</td>
</tr>
<tr>
<td>monophone</td>
<td>100.0</td>
<td>0.4</td>
</tr>
</tbody>
</table>

**Table 4.9: HMM baseline results**

The word error rate (WER) is a measure of three different attributes, combining number of word insertions, word deletions and word substitutions into a single measure. The accuracy is simply a measure of how many words were identified correctly and therefore is a less
CHAPTER 4: RESULTS

significant marker. However, using both markers, the goal was to find a set of parameters that both lowered the word error rate and gave the highest possible accuracy.

4.4 Experimental results

Our experimental strategy used the TIDigits corpus as the primary test bed in determining reasonable parameter ranges for our models. As it was fairly cost effective to run many experiments in a relatively short time, we used this as an opportunity to search exhaustively parts of the parameter space of our newly developed POMDP model. Although the values learned from these experiments were not necessarily applicable to the TIMIT corpus, the experience gained on the behavior of our model was invaluable.

With TIMIT we were much more careful as to what sets of experiments we ran. We used some of the insights we learned on the relationships among the individual HMM context class models to understand better their impact on the combined POMDP model. For instance, the baseline results demonstrated that each corpus had a context model (right biphone for TIDigits and left biphone for TIMIT) that performed best on the test data. We discovered, through our experimental process, that applying the higher weights to those respective context models and lower weights to the remaining context models, yielded our best overall results.

However, the first set of experiments that seemed natural for both corpora was a simple POMDP model that combined all possible context classes with no class weighting added and the constraint of having the same action taken (i.e. same context) through the model. Additionally, we tried every possible permutation of two, three and four different context groups. Note that all experiments that follow in this chapter were run on the respective development sets of the TIDigits and TIMIT corpora.

Table 4.10 below gives the complete list of results we obtained from these initial runs.
Table 4.10: POMDP combinational model permutation results

<table>
<thead>
<tr>
<th>tri phone</th>
<th>left biphone</th>
<th>right biphone</th>
<th>mono phone</th>
<th>TIDigits WER</th>
<th>TIDigits accuracy</th>
<th>TIMIT WER</th>
<th>TIMIT accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>57.2</td>
<td>46.1</td>
<td>90.2</td>
<td>27.6</td>
</tr>
<tr>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>57.8</td>
<td>46.0</td>
<td>88.9</td>
<td>28.0</td>
</tr>
<tr>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>59.2</td>
<td>44.0</td>
<td>84.9</td>
<td>27.5</td>
</tr>
<tr>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>59.7</td>
<td>43.7</td>
<td>84.0</td>
<td>28.1</td>
</tr>
<tr>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>58.1</td>
<td>45.5</td>
<td>86.5</td>
<td>28.1</td>
</tr>
<tr>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>57.7</td>
<td>46.1</td>
<td>86.0</td>
<td>27.9</td>
</tr>
<tr>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>61.3</td>
<td>42.4</td>
<td>84.0</td>
<td>28.0</td>
</tr>
<tr>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>57.5</td>
<td>46.4</td>
<td>85.5</td>
<td>27.6</td>
</tr>
<tr>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>57.0</td>
<td>47.0</td>
<td>84.4</td>
<td>27.7</td>
</tr>
<tr>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>60.1</td>
<td>43.2</td>
<td>79.2</td>
<td>28.7</td>
</tr>
<tr>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>57.4</td>
<td>45.8</td>
<td>85.4</td>
<td>27.9</td>
</tr>
</tbody>
</table>

Although no permutation of the POMDP model combination gave a clear improvement over our baseline HMM models, we were able to get some promising results for each data set, highlighted in table 4.10. With these initial results we then proceeded to fine tune our models by applying the Viterbi decoding variations presented in chapter 3.

### 4.4.1 UMM Viterbi

With the UMM Viterbi algorithm, we have four additional parameters, one for each of the context classes. Weights are treated as discounted rewards in our model and range from 0%, meaning the model is not used, to 100%, meaning the likelihood score remains unchanged. Weights in between that range discount the likelihood score of a particular context as we enter that model.
Many different weighting combination experiments were run to find the set of global uniform weighing factors, $W_c$, one for each context class $c$, that would give optimal accuracy with the smallest possible word error rate. As a starting point, we concentrated on the best models found in our initial experiments from Table 4.10 and weighted those context classes higher than the others. Experiments were run for both corpora, though with TIDigits we ran a more exhaustive set of experiments.

Although we tried many different weighting combinations, especially for the TIDigits corpus, we discovered that keeping unchanged the context class whose HMM score was optimal, consistently gave us the best recognition results. We did this by weighting the optimal class with a uniform weight of 100%. In addition, all models that were shown to give the best results in our initial POMDP combinational permutation model of Table 4.10, were also included with high uniform weight.

Table 4.11 gives the best set of uniform weights found for both the TIDigits and TIMIT corpora.

<table>
<thead>
<tr>
<th>Corpus</th>
<th>$W_c$</th>
<th>WER</th>
<th>accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>tri phone</td>
<td>left biphone</td>
<td>right biphone</td>
</tr>
<tr>
<td>TIDigits</td>
<td>5 100 80 60</td>
<td>56.5</td>
<td>46.7</td>
</tr>
<tr>
<td></td>
<td>0 100 100 0</td>
<td>57.0</td>
<td>47.0</td>
</tr>
<tr>
<td>TIMIT</td>
<td>1 100 0 25</td>
<td>81.6</td>
<td>28.7</td>
</tr>
<tr>
<td></td>
<td>0 100 0 10</td>
<td>78.8</td>
<td>29.0</td>
</tr>
</tbody>
</table>

Table 4.11: UMM Viterbi decoding results

4.4.2 WMM Viterbi

The WMM Viterbi algorithm requires finding three weights per context model. For both corpora we took a statistical summary of the data to get each model’s distribution in its
associated training set, accumulating frequency counts $f_{cm}$, per model $m$, and per context class $c$. Using those frequency counts, we set up experiments to find optimal values for the upper and lower bounds of each context class’s weight, $W^c$ and $L^c$, respectively, and an upper bound, $K^c$, for each context class model’s frequency count.

Initial experiments concentrated on determining a good upper bound, $K^c$, for each context class $c$. This constant is similar to the $K$ constant used in generating backoff language models [Kat87]. If $f_{cm}$ is sufficiently large, the model $m$ of context class $c$ does not need to be discounted from the upper bound $W^c$.

Once we determined a good constant, $K^c$, the focus shifted to finding an optimal sliding window specified by the lower bound $L^c$ and the upper bound $W^c$ for each context class $c$. Table 4.12 gives the best set of weight combinations for both the TIDigits and TIMIT corpora. Notice that, for each corpus, we found a single value for $K$ that was optimal across all context classes.

<table>
<thead>
<tr>
<th>Corpus</th>
<th>tri phone</th>
<th>left biphone</th>
<th>right biphone</th>
<th>mono phone</th>
<th>WER</th>
<th>accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIDigits</td>
<td>0 / 5 / 45</td>
<td>45 / 100 / 45</td>
<td>45 / 100 / 45</td>
<td>0 / 60 / 45</td>
<td>56.4</td>
<td>46.7</td>
</tr>
<tr>
<td></td>
<td>0 / 0 / 0</td>
<td>45 / 95 / 45</td>
<td>35 / 100 / 45</td>
<td>0 / 0 / 0</td>
<td>56.7</td>
<td>47.0</td>
</tr>
<tr>
<td>TIMIT</td>
<td>0 / 1 / 90</td>
<td>25 / 100 / 90</td>
<td>0 / 0 / 0</td>
<td>0 / 20 / 90</td>
<td>81.4</td>
<td>28.7</td>
</tr>
<tr>
<td></td>
<td>0 / 0 / 0</td>
<td>25 / 100 / 90</td>
<td>0 / 0 / 0</td>
<td>0 / 10 / 90</td>
<td>78.4</td>
<td>29.2</td>
</tr>
</tbody>
</table>

Table 4.12: WMM Viterbi decoding results

### 4.4.3 CMM Viterbi

The CMM Viterbi algorithm relaxes the constraint of having to stay with the same action we entered the model with, throughout the model. We now allow context switching within a
context model. As discussed in chapter 3, this context switching can be seen as a “jump”
transitions between models.

As a first attempt, with this relaxing of constraints, we ran two experiments for each
corpus that simply expanded the search space at each state to allow all partial context models
to be included. We ran both the restricted and non-restricted form of this expansion, as
described in chapter 3. No weighting was applied at any transition. Table 4.13 shows the
results for both runs on the TIDigits and TIMIT corpora.

<table>
<thead>
<tr>
<th>Corpus</th>
<th>non-restricted WER accuracy</th>
<th>restricted WER accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIDigits</td>
<td>59.6 43.9</td>
<td>70.0 44.2</td>
</tr>
<tr>
<td>TIMIT</td>
<td>84.2 26.7</td>
<td>90.0 28.9</td>
</tr>
</tbody>
</table>

Table 4.13: WMM Viterbi decoding results

Next we applied weights at each cross model transition of the search. We used the same
weighting strategy derived for the WMM Viterbi search algorithm and followed the two main
strategies, “maximum cross context” and “expanded cross context”, described in chapter 3.
Adding the restricted versus non-restricted property to each cross context mode of the search,
gave us four different configuration of the CMM Viterbi search algorithm. We ran this
configuration on both test corpora.

We used the optimal set of weights from our WMM Viterbi experiments of table 4.11 as a
starting point, and ran a number of experiments to determine a new, optimal set of weights for
our CMM Viterbi algorithm. Again, for TIDigits, we ran an exhaustive set of experiments and
for TIMIT we carefully chose deviations of weighting combinations from the optimal weights
found with WMM Viterbi.
CHAPTER 4: RESULTS

Table 4.14 lists the optimal weight values of all four cross context cases for both the TIDigits and TIMIT corpora.

<table>
<thead>
<tr>
<th>Corpus</th>
<th>cross context type</th>
<th>L^c / W^c / K^c</th>
<th>tri phone</th>
<th>left biphone</th>
<th>right biphone</th>
<th>mono phone</th>
<th>WER</th>
<th>accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIDigits</td>
<td>restricted maximum</td>
<td>45 / 95 / 45</td>
<td>0 / 10 / 45</td>
<td>45 / 100 / 45</td>
<td>0 / 0 / 0</td>
<td>56.8</td>
<td>47.3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>restricted expanded</td>
<td>45 / 95 / 45</td>
<td>5 / 10 / 45</td>
<td>35 / 100 / 45</td>
<td>0 / 0 / 0</td>
<td>56.5</td>
<td>47.7</td>
<td></td>
</tr>
<tr>
<td></td>
<td>maximum</td>
<td>45 / 90 / 45</td>
<td>5 / 10 / 45</td>
<td>45 / 100 / 45</td>
<td>0 / 0 / 0</td>
<td>57.3</td>
<td>45.3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>expanded</td>
<td>45 / 90 / 45</td>
<td>0 / 10 / 45</td>
<td>45 / 100 / 45</td>
<td>0 / 0 / 0</td>
<td>57.3</td>
<td>45.3</td>
<td></td>
</tr>
<tr>
<td>TIMIT</td>
<td>restricted maximum</td>
<td>0 / 0 / 0</td>
<td>25 / 100 / 90</td>
<td>0 / 0 / 0</td>
<td>0 / 10 / 90</td>
<td>81.3</td>
<td>28.8</td>
<td></td>
</tr>
<tr>
<td></td>
<td>restricted expanded</td>
<td>0 / 0 / 0</td>
<td>25 / 80 / 90</td>
<td>0 / 0 / 0</td>
<td>0 / 20 / 90</td>
<td>81.2</td>
<td>28.3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>maximum</td>
<td>0 / 0 / 0</td>
<td>45 / 100 / 90</td>
<td>0 / 0 / 0</td>
<td>0 / 10 / 90</td>
<td>77.0</td>
<td>29.2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>expanded</td>
<td>0 / 0 / 0</td>
<td>25 / 80 / 90</td>
<td>0 / 0 / 0</td>
<td>0 / 15 / 90</td>
<td>77.2</td>
<td>29.0</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.14: CMM Viterbi decoding results

4.5 Summary

In this chapter we discussed results obtained from testing our newly developed POMDP model. We discussed the importance of a good experimental strategy in utilizing limited resources and managing the vast amount of information that can result from running many experiments. Developing a model with the potential of improving speech recognition is only part of the process. A sound experiment methodology can help not only in demonstrating the power and robustness of the new model, but also in highlighting weaknesses that require further investigation.

An important component of any experimental system is a mechanism to generate and record experiments easily. A brief introduction to the PASS system was given, and to its importance in generating the vast number of experiments needed in testing the new model. Additionally, as speech recognition in general can be a computationally resource intensive
process, we also discussed the importance in the overall strategy of planning what experiments
to actually run.

The ISIP system, the source code foundation on which all algorithms were developed on,
was discussed next. Although overall recognition performance is an important guiding
attribute in choosing a system, a well designed code base is just as important. As the ISIP
system combines both Object Oriented aspects in its basic speech model description, as well as
procedural aspects in its overall algorithmic structure, we found it a perfect environment to
create our modified Viterbi decoder.

We next discussed the two data sets, TIDigits and TIMIT, that were used in our
experiments. Information about the general makeup of each corpus was given, as well as
specific statistics on various attributes of each. How the language model of each corpora was
generated, was also covered.

Finally, the individual experiments were covered. From the initial set of baselines, used as
a reference point for all future work, to each of the three different Viterbi decoding methods
that were developed specifically for the newly formed POMDP model. Results were given for
each of the modified Viterbi decoding algorithm on both of the two data sets, TIDigits and
TIMIT.
Analysis

In this chapter we will discuss the results obtained by the many decoding experiments that were run to test our POMDP model on the different Viterbi decoding algorithms. Although many positive results were obtained, a consistent configuration of our new decoding strategy has not yet been found. The goal of any speech recognition system is to perform optimally, independent of the data set used. As is true more often than not, however, data dependencies persist and need to be dealt with in a systematic way.

Although a great deal of work has already been done in both developing our new POMDP acoustic model, and rigorously testing the various decoding algorithms developed for it, more work is needed to find the best way to integrate this model into mainstream speech recognition research. We will discuss various ideas for future work at the end of chapter six.
CHAPTER 5: ANALYSIS

5.1 Behavior of decoding

Baseline results obtained for TIDigits were slightly better than the tutorial experiment, based on the TIDigits data set, that is delivered with the standard ISIP speech recognition system. The difference in performance is due to an improved language model that we used for our task. We could not find a reference of the ISIP system having run the TIMIT data set, though we did find similar results based on HTK, a state of the art speech recognition systems developed by Cambridge University. The word error rate and accuracy for a triphone trained HMM system using HTK on TIMIT was very close to our triphone HMM baseline system.\(^1\)

Table 5.1 summarizes the best results obtained through experimentation of each of the three newly developed Viterbi decoding algorithms for our POMDP model, and shows that these models generally improve upon the best baseline HMM systems. The high word error rate obtained on both corpora is due to the small size of the data used for training [Rab85] [Man99].

<table>
<thead>
<tr>
<th>Corpus</th>
<th>Model</th>
<th>Viterbi</th>
<th>(L^c / W^c / K^c)</th>
<th>WER</th>
<th>accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>tri phone</td>
<td>left biphone</td>
<td>right biphone</td>
</tr>
<tr>
<td>TIDigits</td>
<td>HMM</td>
<td>Standard</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td></td>
<td>POMDP</td>
<td>UMM</td>
<td>- / 5 / -</td>
<td>- / 100 / -</td>
<td>- / 80 / -</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WMM</td>
<td>0 / 5 / 45</td>
<td>45 / 100 / 45</td>
<td>45 / 100 / 45</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CMM (restrict exp)</td>
<td>45 / 95 / 45</td>
<td>5 / 10 / 45</td>
<td>35 / 100 / 45</td>
</tr>
<tr>
<td>TIMIT</td>
<td>HMM</td>
<td>Standard</td>
<td>no</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td></td>
<td>POMDP</td>
<td>UMM</td>
<td>- / 0 / -</td>
<td>- / 100 / -</td>
<td>- / 0 / -</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WMM</td>
<td>0 / 0 / 0</td>
<td>25 / 100 / 90</td>
<td>0 / 0 / 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CMM (maximum)</td>
<td>0 / 0 / 0</td>
<td>45 / 100 / 90</td>
<td>0 / 0 / 0</td>
</tr>
</tbody>
</table>

\(1\) We found reference of the TIMIT data set used with the HTK system on the user feedback forum specifically set up for the HTK user community.
Note that in a majority of cases of our best decoding results, the weights for at least half the models are set to zero. Though our POMDP models improved over the baseline HMM models, only half the models in our weighted combination were used. Our eventual goal is to find an optimal set of weights that includes all models and yields the best performance improvements, but this will require much larger data sets than TIDigits or TIMIT. These data sets are not large enough for the tirphone models to train fully.

Although we did not directly address the runtime speed of our new Viterbi decoding algorithms, as it was not a crucial part of our research, a brief analysis yielded a speed degradation over the HMM Viterbi algorithm for all three modified Viterbi methods. We gave a description of the runtime cost of the CMM Viterbi decoder at the end of section 3.6.3 and noted that both UMM Viterbi and WMM Viterbi did not alter the basic recursion of Viterbi as they only added to the number of models that Viterbi had to search. The increased number of models in all three modified Viterbi algorithms is the main catalyst for the reduced decoding speed, although the effects of pruning greatly diminishes the net effect to the overall runtime speed.

5.1.1 UMM Viterbi

With UMM Viterbi decoding, results were mixed, although as an initial attempt at decoding our POMDP model we were encouraged. Though accuracy improved on both data sets, the more important measure of word error rate only improved for TIDigits. Since we were able to run an extensive set of weight combination experiments for TIDigits, this result is not surprising.

It is unclear that with more compute resources similar results could have been obtained for the TIMIT corpus. Since the general approach of flatly weighting each class is not
optimal, as it does not take model coverage of the training data set into account, we believe that this method is not the best approach when combining multiple context class models to generate a POMDP model.

5.1.2 WMM Viterbi

The WMM Viterbi decoding method gave us promising results. As table 5.1 illustrates, for both the TIDigits and TIMIT data sets, we saw improvements in both word error rate and accuracy. Drawing on the ideas of language modeling techniques, our weighting approach took a simple minmax strategy with a K cutoff constant to avoid discounting well trained models. Though this technique resulted in improved performance, and seemed an intuitive first step in the cross context model weighting process, more sophisticated techniques that take into account the strengths of the different context models could be applied to the POMDP model as a whole.

Looking at the optimal results, one can deduce that the HMM model with the best performance in our baseline set, is also the context class that received the highest overall weighting factor across all data sets. However, this does not hold true for the remaining context classes. For example, the monophone model received a higher weight than the triphone model in all of our optimal weighting configurations, even though the HMM monophone model was by far the worst in performance.

A lower order context model, such as the monophone model, although less discriminating, requires less data to be well trained. Since all of our models have been built with very little data, perhaps a lower order model adds less uncertainty to the overall POMDP model, than does a more discriminating, yet undertrained higher order context model, such as a triphone model. More work is needed to understand better the relationships among the different context classes, so that less information is discarded.
5.1.3 CMM Viterbi

Our best performance for both the TIDigits and TIMIT data sets comes from our CMM Viterbi decoding method, as highlighted in table 5.1. This is of great interest to us as it utilizes the full power of a POMDP model. By allowing cross context model jumps within our model, we are able to explore a search space that has not been looked at in any previous acoustic modeling work with traditional HMM’s to date.

Allowing these cross model transitions, with proper weighting and a well constrained cross context model set, is an important discovery in acoustic modeling techniques. These new interconnections could open up an already existing yet unexplored parameter set, rich in acoustic features, that can further improve the search for the best possible path through the acoustic model.

As with WMM Viterbi, the same issues regarding the model weighting apply, so a better understanding of the relationships among context class models may result in further improvements in performance. However, another important aspect of the CMM Viterbi decoding algorithm is the constraint on the cross context model set. Currently that constraint is driven by the linguistic properties of the model (i.e. only models with similar partial context are included in the constraint set). Perhaps a data driven approach that excludes robust, well trained context models, and includes other, similar context models in that constraint set could further improve performance.

5.2 Data independence

An important consideration of any robust speech model is whether or not any performance improvements made are independent of the data. There are generally two ways to demonstrate this independence. One method uses a third, evaluation corpus, independent
of both the training and development corpora. A simple run of the best parameter settings of
the system, using this evaluation set can show whether or not the improvements achieved by
the new model are indeed improvements or simply a finely tuned model to the specific
development data set.

A second method, equally as sound in demonstrating data independence, involves analysis
of the corpus on the results obtained. By splitting the decoded results of the development set
in two, using a fair, random process, and then rescoring both halves of the data independently,
one can determine if improvements made as a whole are still applicable to both parts
independently. We apply both approaches to demonstrate the degree to which our model
tuning is independent of the data.

5.2.1 Corpus analysis

Both TIDigits and TIMIT development corpora were randomly split into two evenly
divided data sets, with no speaker overlap between the two sets. We labeled each new set by
marking them with “-1” and “-2” for either of the two halves of the split corpus. We ran the
equivalent set of baseline and optimal POMDP decoding experiments that were performed on
each corpora as a whole.

To demonstrate that the data set is independent of the parameters that were tuned using it,
we need to show that the two randomly constructed subsets of the data set have the same
relative performance characteristics within each of them, as the data set taken as a whole. Even
if the absolute performance differs greatly between the two subsets, as long as the relative
ranking within each subset to the various decoding strategies and the baseline numbers
remains unchanged, then we can feel confident that we did not tune the systems parameters to
the data.
5.2.1.1 TIDigits analysis results

TIDigits was split along speakers into two subsets, each containing 10 speakers. A speaker identifier string is made up of first and last name initials. Table 5.2 below lists the speakers in each set.

<table>
<thead>
<tr>
<th>Corpus</th>
<th>Speakers</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIDigits-1</td>
<td>ah, ar, at, bc, be, bm, bn, cc, ce, cp</td>
</tr>
<tr>
<td>TIDigits-2</td>
<td>df, dj, ed, ef, et, fa, fg, fh, fm, fp</td>
</tr>
</tbody>
</table>

Table 5.2: List of speakers

Table 5.3 below gives the baseline results for each of the subsets of the TIDigits development corpus.

<table>
<thead>
<tr>
<th>Corpus</th>
<th>WER</th>
<th>accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIDigits-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>triphone</td>
<td>56.8</td>
<td>45.7</td>
</tr>
<tr>
<td>left biphone</td>
<td>56.6</td>
<td>45.8</td>
</tr>
<tr>
<td>right biphone</td>
<td>54.2</td>
<td>48.1</td>
</tr>
<tr>
<td>monophone</td>
<td>99.1</td>
<td>9.6</td>
</tr>
<tr>
<td>TIDigits-2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>triphone</td>
<td>64.4</td>
<td>40.1</td>
</tr>
<tr>
<td>left biphone</td>
<td>63.4</td>
<td>40.4</td>
</tr>
<tr>
<td>right biphone</td>
<td>59.6</td>
<td>44.4</td>
</tr>
<tr>
<td>monophone</td>
<td>97.1</td>
<td>8.7</td>
</tr>
</tbody>
</table>

Table 5.3: HMM baseline results

Table 5.4 below gives results of the best POMDP model parameters obtained from the entire TIDigits set run on each of the two individual subsets.
The analysis of the TIDigits development set demonstrates that the performance within each subset remains relatively the same to the set as a whole. We discovered only slight variations within each subset, both occurring with the CMM decoding method (shaded in the table above). Specifically, the CMM word error rate is now slightly better than the UMM word error rate in TIDigits-1 and slightly worse than it in TIDigits-2. In the TIDigits development set as a whole, these results for both methods yielded the same word error rate (refer to table 5.1).

However, all three decoding methods for both TIDigits-1 and TIDigits-2 still performed better than their respective baseline word error rates. This is an important result, as it demonstrates the robustness of the POMDP model.

### 5.2.1.2 TIMIT analysis result

The TIMIT development corpus was split along speakers into two subsets. Since there were a total of 11 speakers, one subset contained 6 speakers and the other subset contained 5 speakers. Additionally, as there were a total of 4 female and 7 male speakers in the entire set, we made sure that 2 female speakers were present in each subset.
The first letter of each speaker designates the gender, followed by three letter initials and a number to insure uniqueness among potentially identical speaker identifier strings. Table 5.5 below lists the speakers in each set.

<table>
<thead>
<tr>
<th>Corpus</th>
<th>Gender</th>
<th>Speakers</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMIT-1</td>
<td>female</td>
<td>faks0, felc0</td>
</tr>
<tr>
<td></td>
<td>male</td>
<td>mdab0, mjsw0, mreb0, mrjo0</td>
</tr>
<tr>
<td>TIMIT-2</td>
<td>female</td>
<td>fdac1, fjem0</td>
</tr>
<tr>
<td></td>
<td>male</td>
<td>msjs1, mstk0, mwbt0</td>
</tr>
</tbody>
</table>

**Table 5.5:** List of speakers TIMIT subsets

Table 5.6 below gives the baseline results for each of the subsets of the TIMIT development corpus.

<table>
<thead>
<tr>
<th>Corpus</th>
<th>WER</th>
<th>accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMIT-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>triphone</td>
<td>79.8</td>
<td>27.6</td>
</tr>
<tr>
<td>left biphone</td>
<td>78.4</td>
<td>27.1</td>
</tr>
<tr>
<td>right biphone</td>
<td>81.6</td>
<td>28.1</td>
</tr>
<tr>
<td>monophone</td>
<td>100.3</td>
<td>0.1</td>
</tr>
<tr>
<td>TIMIT-2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>triphone</td>
<td>84.0</td>
<td>29.8</td>
</tr>
<tr>
<td>left biphone</td>
<td>78.5</td>
<td>28.7</td>
</tr>
<tr>
<td>right biphone</td>
<td>84.6</td>
<td>26.9</td>
</tr>
<tr>
<td>monophone</td>
<td>99.5</td>
<td>0.7</td>
</tr>
</tbody>
</table>

**Table 5.6:** HMM baseline results
Table 5.7 below gives results of the best POMDP model parameters obtained from the entire TIMIT set run on each of the two individual subsets.

<table>
<thead>
<tr>
<th>Corpus</th>
<th>Viterbi</th>
<th>$L^c$ / $W^c$ / $K^c$</th>
<th>WER</th>
<th>accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIMIT-1</td>
<td>UMM</td>
<td>- / 0 / -</td>
<td>- / 100 / -</td>
<td>- / 0 / -</td>
</tr>
<tr>
<td></td>
<td>WMM</td>
<td>0 / 0 / 0</td>
<td>25 / 100 / 90</td>
<td>0 / 0 / 0</td>
</tr>
<tr>
<td></td>
<td>CMM (maximum)</td>
<td>0 / 0 / 0</td>
<td>45 / 100 / 90</td>
<td>0 / 0 / 0</td>
</tr>
<tr>
<td>TIMIT-2</td>
<td>UMM</td>
<td>- / 0 / -</td>
<td>- / 100 / -</td>
<td>- / 0 / -</td>
</tr>
<tr>
<td></td>
<td>WMM</td>
<td>0 / 0 / 0</td>
<td>25 / 100 / 90</td>
<td>0 / 0 / 0</td>
</tr>
<tr>
<td></td>
<td>CMM (maximum)</td>
<td>0 / 0 / 0</td>
<td>45 / 100 / 90</td>
<td>0 / 0 / 0</td>
</tr>
</tbody>
</table>

Table 5.7: Rescoring results of split TIMIT development set

As with the TIDigits analysis, we see a similar result with the analysis of the TIMIT development set. We see only slight variations within each subset (shaded in the table above). Specifically, in the TIMIT-1 data set we see that the UMM Viterbi decoding word error rate is now slightly better than its baseline left-biphone word error rate. Conversely, in the TIMIT-2 data set we see that the WMM word error rate is now slightly worse than its baseline left-biphone word error rate, though the accuracy is higher.

These variations are again small, and each subset still demonstrates the performance improvements of the POMDP model over the HMM model. This is especially so with our best model, CMM Viterbi, where the results remain unchanged within each of the two subsets.

5.2.2 TIMIT evaluation set

The TIMIT evaluation set (DR2s1), as with the development set, was chosen from a standard TIMIT test set (DR2), assuring that no speaker was present in training. Additionally, no speakers present in the evaluation set overlapped with those from the development set. The
size of the evaluation set was chosen to match that of the development set. Table 5.8 lists the evaluation corpus statistics.

<table>
<thead>
<tr>
<th>TIMIT DR2s1 evaluation corpus</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of utterances</td>
</tr>
<tr>
<td>Number of speakers</td>
</tr>
<tr>
<td>Corpus length in minutes</td>
</tr>
</tbody>
</table>

**Table 5.8:** TIMIT evaluation corpus statistics

Since the candidate evaluation set was more than twice the size, we randomly selected 11 speakers to create the evaluation set, and kept the ratio of the number of male to female speakers the same as in the development set. Table 5.9 lists the speakers, grouped by gender.

<table>
<thead>
<tr>
<th>Corpus</th>
<th>Gender</th>
<th>Speakers</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMIT-1</td>
<td>female</td>
<td>fdrd1, fjre0, fjwb0, fslb1</td>
</tr>
<tr>
<td></td>
<td>male</td>
<td>mabw0, mccs0, mdbb0, mdl0, mpdf0, mpgl0, mwew0</td>
</tr>
</tbody>
</table>

**Table 5.9:** List of speakers for TIMIT evaluation set

The TIMIT evaluation test set parallels both the training and development corpus in its compositions. Each speaker recorded on average 10 utterances, each lasting approximately 3 seconds. And the mean and the standard deviation on the number of words in the development set also matched that of training.

The same trigram language model and dictionary created for decoding the development set was used in decoding the evaluation set.
5.2.2.1 TIMIT evaluation results

Table 5.10 below details the baseline HMM results using the TIMIT evaluation test set.

<table>
<thead>
<tr>
<th>Corpus</th>
<th>WER</th>
<th>accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMIT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>triphone</td>
<td>80.7</td>
<td>27.2</td>
</tr>
<tr>
<td>left biphone</td>
<td>78.4</td>
<td>28.2</td>
</tr>
<tr>
<td>right biphone</td>
<td>80.9</td>
<td>27.5</td>
</tr>
<tr>
<td>monophone</td>
<td>99.5</td>
<td>0.9</td>
</tr>
</tbody>
</table>

Table 5.10: HMM baseline results for TIMIT evaluation set

Table 5.11 below details results of the best configuration for the three modified Viterbi decoding algorithms using the TIMIT evaluation test set.

<table>
<thead>
<tr>
<th>Corpus</th>
<th>Viterbi</th>
<th>Lc / Wc / Kc</th>
<th>WER</th>
<th>accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>tri phone</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>left biphone</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>right biphone</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>monophone</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIMIT</td>
<td>UMM</td>
<td>- / 0 / -</td>
<td>78.4</td>
<td>29.0</td>
</tr>
<tr>
<td></td>
<td>WMM</td>
<td>0 / 0 / 0</td>
<td>78.3</td>
<td>28.8</td>
</tr>
<tr>
<td></td>
<td>CMM (maximum)</td>
<td>0 / 0 / 0</td>
<td>76.3</td>
<td>28.9</td>
</tr>
</tbody>
</table>

Table 5.11: Best POMDP decoding results on TIMIT evaluation set

Both the baseline results and the modified Viterbi results for all three decoding methods closely matched those of the development set. The CMM Viterbi method again yielded the best overall word error rate. This demonstrates that the tuning done on the development set is indeed independent of the data and supports the results of the analysis done on both development sets.
5.3 Summary

In this chapter we discussed the results obtained from the many experiments that were run on the three different modified Viterbi decoding algorithms. We rated the different methods of decoding and pointed out strengths and weaknesses in each. We found that both WMM Viterbi as well as CMM Viterbi gave improved results for both TIDigits and TIMIT data sets over the initial baseline results.

We discussed the importance of demonstrating that performance improvements attained by the POMDP model were independent of the data. By merely fine tuning a set of parameters on a specific data set one could show improved performance that perhaps may not be representative of the model’s contribution. We did an analysis of our results to demonstrate this data independence.

Finally, we used an unseen evaluation corpus for the TIMIT data set, similar size and makeup to the development set, with no overlapping speakers to either the development or training set, to verify that the tuning is indeed independent of the data. We ran a set of baseline experiments and the best configuration of each of the modified Viterbi decoding algorithms to demonstrate that the results on the unseen data set were similar to the results of the development set.
Chapter 6

Conclusions

We started this dissertation with a general discussion of the speech recognition process. We mapped out the modern approach of an HMM based recognizer, and how it generates and applies its acoustic models to determine what was said. We then applied our new model, a POMDP, in place of the traditional HMM and developed various decoding methods to utilize better the richer feature set that it offers.

After developing a rigorous set of experiments not only to understand better our POMDP model but to also show how it improves performance over the traditional HMM model, we discussed the various results achieved. Although all three decoding methods showed improvements over the HMM based systems, we discovered that the model that utilized the POMDP to its fullest, the CMM Viterbi algorithm, outperformed the rest. This new model expands an entirely new search space by allowing cross model transitions while in the midst
of a model. This has not been explored, to our knowledge, with any acoustic model techniques employed in speech recognition research to date.

In this chapter, we conclude our discussion by focusing on the importance of the POMDP model in a theoretical sense. Our research was not only guided by the drive to improve performance, but also by the desire to develop a better model framework that could capture the benefits of today’s HMM based speech acoustic models, yet extend them in a new dimension so as to extract a even more information from training data. We also look at future work in several areas that should yield additional insights as well as performance improvements.

### 6.1 POMDP model for speech

So far, we have discussed the performance of our new POMDP model, and although many of the results have shown improvements, we believe this should not be the sole reason for continuing research into fully developing our model as the center piece of acoustic model development. As was discussed in chapter 3, our model is able to represent a single phoneme in all its many contexts. In essence, our new model is able to represent acoustic models in their purest form possible, the phoneme.

By collecting context and making it an attribute of our model, we can now look at the interrelationships among both the different contexts of a phoneme as well as the various context classes. Having our POMDP model in theory represent a phoneme in all possible contexts can have a profound impact on acoustic modeling building. Now the focus on model building will no longer depend solely upon developing good context models, but instead, developing ways to more completely specify and fill out this general representation of a phoneme using a POMDP.
CHAPTER 6: CONCLUSIONS

Just as the transition from building prototype templates to building HMM based acoustic models brought about many new discoveries in how to expand the model to represent better the speech signal, so too can the transition from HMM to POMDP open research avenues, rich in potential of new discoveries. We did not approach this research with an idea of combining different context models into a more complex HMM, and then later discovered a way to map this to a POMDP. On the contrary, we started by redefining the acoustic model in the form of a POMDP, and incrementally refined this until we integrated multiple contexts of a phoneme as a single representative model. From there, the idea of combining these models during the decoding process, similar to what is done in language model, was a natural progression.

We feel that the POMDP framework offers a rich environment for better modeling the complexities of speech, and for better understanding how these complexities affect overall performance. Additionally, much research has been done within the POMDP framework itself, specifically in developing efficient algorithms to find solutions, or policies as they are called. Integrating this knowledge bed of research into the domain of speech could add important discoveries that help to improve the overall performance of acoustic modeling techniques.

6.2 Future work

We have demonstrated improved performance with a well constructed POMDP model. More research needs to be done to determine how better to construct such models and how better to utilize the information contained within that model. We will expand on several ideas with the goal of improving the parameter set of our model and developing better techniques that utilize that parameter set.

We will also look at ways to integrate more closely the POMDP model building process at the basic training level. As we continue to investigate the POMDP model in speech
recognition, we hope to understand better its capabilities. This will create a more robust framework that utilizes the POMDP model to its fullest and increases its recognition performance.

### 6.2.1 Language Model weighting techniques

Further investigation is needed to determine how best to weight each of our different context models so we can combine them into an even more robust model. We plan to look more closely at language modeling techniques for combining related information, using both *linear interpolation* as well as *backoff*.

### 6.2.2 Clustering context classes

Expanding the constraint context classes used in CMM Viterbi is another area with potential for improvement. Using linguistic information to tie mixtures at the state level has been shown to provide the best discriminating acoustic model in modern speech recognition systems. We plan to investigate the use of linguistic information to group context classes into clusters to create better constrained cross context classes.

Additionally, by gathering a model’s statistics from the data set it was built from, we may be able to fine tune clusters by excluding models that are well represented and merging models that are underrepresented.

### 6.2.3 Modified EM

Most of our work thus far has centered around modifying the Viterbi algorithm. We have not looked at modifying the training process to incorporate the more general POMDP model. Currently, we train each of our different context models separately using EM and then combine them to build the POMDP model.
Perhaps a modified form of EM that has knowledge of the structure of the POMDP model can help in developing context model weights during the training process. Since we use the same data both during the training process as we build the models, as well as during the decoding process to generate the context model weights, it seems a natural area to combine both processes into a single training algorithm. This could save time and generate a better set of weights that helps improve the overall recognition performance.

6.3 Summary

In this chapter we reflected on the work done in this dissertation. We outlined the modern, HMM based speech recognition process, introduced a new POMDP based model in its place, experimented with the new model and showed performance gain, and identified strength and weakness of the model.

We discussed the importance of applying the POMDP model to the speech recognition process. Although the newly developed model has achieved significant performance improvements on both the TIDigits and TIMIT data sets, this is not the only guiding force behind this research. By replacing the traditional HMM model with the more general POMDP model, we allow for new discoveries to be made that could have important ramifications on speech recognition research. We have introduced one such new discovery in this work by developing the CMM Viterbi algorithm and showing how it has outperformed all other models applied in this work.

Finally, we concluded by discussing future work on this research. Better development of context model weights, as well as linguistically driven constraint context classes could have an important impact on performance of the model. Additionally, incorporating the rich structure of the POMDP model directly into the training process by developing variants of the
traditional EM algorithm, could, in fact, be an improved way to generate optimal context model weights.
Bibliography

Markov model parameters so as to maximize speech recognition accuracy,”
IEEE Transactions on Speech and Audio Processing, vol. 1, no. 1, pp. 77-83,
January 1993.

language model for natural language speech recognition,” IEEE
Transactions on Acoustics, Speech and Signal Processing, vol. 37, pp. 1001-08,
July 1989.

[Bau72] L. Baum, “An Inequality and Associated Maximization Technique in
Statistical Estimation of Probabilistic Functions of Markov Processes,”


modeling for speech recognition,” IEEE Transactions on Acoustics, Speech,

Parameter Estimation for Gaussian Mixture and Hidden Markov Models,”

International Conference on Acoustics, Speech, and Signal Processing, March
1999.


Combination: Application to Phone Recognition,” International Conference


Resnik, “Language modeling using decision trees,” IBM Research Report,

“Class-based n-gram models of natural language,” Computational


BIBLIOGRAPHY


[Man99] S. Manhung, M. Jonas, and H. Gish, “Using a large vocabulary continuous speech recognizer for a constrained domain with limited training,” IEEE


BIBLIOGRAPHY


