Efficient Algorithms for Speech Recognition

Mosur K. Ravishankar

May 15, 1996

CMU-CS-96-143

School of Computer Science
Computer Science Division
Carnegie Mellon University
Pittsburgh, PA 15213

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Thesis Committee:

Roberto Bisiani, co-chair (University of Milan)
Raj Reddy, co-chair
Alexander Rudnicky
Richard Stern
Wayne Ward

This research was supported by the Department of the Navy, Naval Research Laboratory under Grant No. N00014-93-1-2005. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. government.

Keywords: Speech recognition, search algorithms, real time recognition, lexical tree search, lattice search, fast match algorithms, memory size reduction.

Abstract

Advances in speech technology and computing power have created a surge of interest in the practical application of speech recognition. However, the most accurate speech recognition systems in the research world are still far too slow and expensive to be used in practical, large vocabulary continuous speech applications. Their main goal has been recognition accuracy, with emphasis on acoustic and language modelling. But practical speech recognition also requires the computation to be carried out in real time within the limited resources—CPU power and memory size—of commonly available computers. There has been relatively little work in this direction while preserving the accuracy of research systems.

In this thesis, we focus on *efficient and accurate* speech recognition. It is easy to improve recognition speed and reduce memory requirements by trading away accuracy, for example by greater pruning, and using simpler acoustic and language models. It is much harder to improve both the recognition speed and reduce main memory size while preserving the accuracy.

This thesis presents several techniques for improving the overall performance of the CMU Sphinx-II system. Sphinx-II employs semi-continuous hidden Markov models for acoustics and trigram language models, and is one of the premier research systems of its kind. The techniques in this thesis are validated on several widely used benchmark test sets using two vocabulary sizes of about 20K and 58K words.

The main contributions of this thesis are *an 8-fold speedup* and *4-fold memory size reduction* over the baseline Sphinx-II system. The improvement in speed is obtained from the following techniques: lexical tree search, phonetic fast match heuristic, and global best path search of the word lattice. The gain in speed from the tree search is about a factor of 5. The phonetic fast match heuristic speeds up the tree search by another factor of 2 by finding the most likely candidate phones active at any time. Though the tree search incurs some loss of accuracy, it also produces compact word lattices with low error rate which can be rescored for accuracy. Such a rescoring is combined with the best path algorithm to find a globally optimum path through a word lattice. This recovers the original accuracy of the baseline system. The total recognition time is about 3 times real time for the 20K task on a 175MHz DEC Alpha workstation.

The memory requirements of Sphinx-II are minimized by reducing the sizes of the acoustic and language models. The language model is maintained on disk and bigrams and trigrams are read in on demand. Explicit software caching mechanisms effectively overcome the disk access latencies. The acoustic model size is reduced by simply truncating precision of probability values to 8 bits. Several other engineering solutions, not explored in this thesis, can be applied to reduce memory requirements further. The memory size for the 20K task is reduced to about 30-40MB.

Acknowledgements

I cannot overstate the debt I owe to Roberto Bisiani and Raj Reddy. They have not only helped me and given me every opportunity to extend my professional career, but also helped me through personal difficulties as well. It is quite remarkable that I have landed not one but two advisors that combine integrity towards research with a human touch that transcends the proverbial hard-headedness of science. One cannot hope for better mentors than them. Alex Rudnicky, Rich Stern, and Wayne Ward, all have a clarity of thinking and self-expression that simply amazes me without end. They have given me the most insightful advice, comments, and questions that I could have asked for. Thank you, all.

The CMU speech group has been a pleasure to work with. First of all, I would like to thank some former and current members, Mei-Yuh Hwang, Fil Alleva, Lin Chase, Eric Thayer, Sunil Issar, Bob Weide, and Roni Rosenfeld. They have helped me through the early stages of my induction into the group, and later given invaluable support in my work. I'm fortunate to have inherited the work of Mei-Yuh and Fil. Lin Chase has been a great friend and sounding board for ideas through these years. Eric has been all of that and a great officemate. I have learnt a lot from discussions with Paul Placeway. The rest of the speech group and the robust gang has made it a most lively environment to work in. I hope the charge continues through Sphinx-III and beyond.

I have spent a good fraction of my life in the CMU-CS community so far. It has been, and still is, the greatest intellectual environment. The spirit of cooperation, and informality of interactions as simply unique. I would like to acknowledge the support of everyone I have ever come to know here, too many to name, from the Warp and Nectar days until now. The administrative folks have always succeeded in blunting the edge off a difficult day. You never know what nickname Catherine Copetas will christen you with next. And Sharon Burks has always put up with all my antics.

It goes without saying that I owe everything to my parents. I have had tremendous support from my brothers, and some very special uncles and aunts. In particular, I must mention the fun I've had with my brother Kuts. I would also like to acknowledge K. Gopinath's help during my stay in Bangalore. Finally, "BB", who has suffered through my tantrums on bad days, kept me in touch with the rest of the world, has a most creative outlook on the commonplace, can drive me nuts some days, but when all is said and done, is a most relaxed and comfortable person to have around.

Last but not least, I would like to thank Andreas Nowatzyk, Monica Lam, Duane Northcutt and Ray Clark. It has been my good fortune to witness and participate in some of Andreas's creative work. This thesis owes a lot to his unending support and encouragement.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Recent advances in speech technology and computing power have created a surge of interest in the practical application of speech recognition. Speech is the primary mode of communication among humans. Our ability to communicate with machines and computers, through keyboards, mice and other devices, is an order of magnitude slower and more cumbersome. In order to make this communication more user-friendly, speech input is an essential component.

There are broadly three classes of speech recognition applications, as described in [53]. In isolated word recognition systems each word is spoken with pauses before and after it, so that end-pointing techniques can be used to identify word boundaries reliably. Second, highly constrained command-and-control applications use small vocabularies, limited to specific phrases, but use connected word or continuous speech. Finally, large vocabulary continuous speech systems have vocabularies of several tens of thousands of words, and sentences can be arbitrarily long, spoken in a natural fashion. The last is the most user-friendly but also the most challenging to implement. However, the most accurate speech recognition systems in the research world are still far too slow and expensive to be used in practical, large vocabulary continuous speech applications on a wide scale.

Speech research has been concentrated heavily on *acoustic* and *language* modelling issues. Since the late 1980s, the complexity of tasks undertaken by speech researchers has grown from the 1000-word *Resource Management* (*RM*) task [51] to essentially unlimited vocabulary tasks such as transcription of radio news broadcast in 1995 [48]. While the word recognition accuracy has remained impressive, considering the increase in task complexity, the resource requirements have grown as well. The *RM* task ran about an order of magnitude slower than real time on processors of that day. The unlimited vocabulary tasks run about two orders of magnitude slower than real time on modern workstations whose power has grown by an order of magnitude again, in the meantime.

The task of large vocabulary continuous speech recognition is inherently hard for the following reasons. First, word boundaries are not known in advance. One must be constantly prepared to encounter such a boundary at every time instant. We can draw a rough analogy to reading a paragraph of text without any punctuation marks or spaces between words:

> myspiritwillsleepinpeaceorifthinksitwillsurelythinkthusfarewellhesprangfrom thecabinwindowashesaidthisupontheiceraftwhichlayclosetothevesselhewassoon borneawaybythewavesandlostindarknessanddistance...

Furthermore, many incorrect word hypotheses will be produced from incorrect segmentation of speech. Sophisticated *language models* that provide word context or semantic information are needed to disambiguate between the available hypotheses.

The second problem is that *co-articulatory* effects are very strong in natural or conversational speech, so that the sound produced at one instant is influenced by the preceding and following ones. Distinguishing between these requires the use of detailed acoustic models that take such contextual conditions into account. The increasing sophistication of language models and acoustic models, as well as the growth in the complexity of tasks, has far exceeded the computational and memory capacities of commonly available workstations.

Efficient speech recognition for practical applications also requires that the processing be carried out in real time within the limited resources—CPU power and memory size—of commonly available computers. There certainly are various such commercial and demonstration systems in existence, but their performance has never been formally evaluated with respect to the research systems or with respect to one another, in the way that the accuracy of research systems has been. This thesis is primarily concerned with these issues—in improving the computational and memory efficiency of current speech recognition technology without compromising the achievements in recognition accuracy.

The three aspects of performance, recognition speed, memory resource requirements, and recognition accuracy, are in mutual conflict. It is relatively easy to improve recognition speed and reduce memory requirements while trading away some accuracy, for example by pruning the search space more drastically, and by using simpler acoustic and language models. Alternatively, one can reduce memory requirements through efficient encoding schemes at the expense of computation time needed to decode such representations, and *vice versa*. But it is much harder to improve both the recognition speed and reduce main memory requirements while preserving or improving recognition accuracy. In this thesis, we demonstrate algorithmic and heuristic techniques to tackle the problem.

This work has been carried out in the context of the CMU Sphinx-II speech recognition system as a baseline. There are two main schools of speech recognition technology today, based on statistical hidden Markov modelling (HMM), and neural net technology, respectively. Sphinx-II uses HMM-based statistical modelling techniques and is one of the premier recognizers of its kind. Using several commonly used benchmark test sets and two different vocabulary sizes of about 20,000 and 58,000 words, we demonstrate that the recognition accuracy of the baseline Sphinx-II system can be attained while its execution time is reduced by about an order of magnitude and memory requirements reduced by a factor of about 4.

1.1 The Modelling Problem

As the complexity of tasks tackled by speech research has grown, so has that of the modelling techniques. In systems that use statistical modelling techniques, such as the Sphinx system, this translates into several tens to hundreds of megabytes of memory needed to store information regarding statistical distributions underlying the models.

Acoustic Models

One of the key issues in acoustic modelling has been the choice of a good unit of speech [32, 27]. In small vocabulary systems of a few tens of words, it is possible to build separate models for entire words, but this approach quickly becomes infeasible as the vocabulary size grows. For one thing, it is hard to obtain sufficient training data to build all individual word models. It is necessary to represent words in terms of *sub-word* units, and train acoustic models for the latter, in such a way that the pronunciation of new words can be defined in terms of the already trained sub-word units.

The *phoneme* (or phone) has been the most commonly accepted sub-word unit. There are approximately 50 phones in spoken English language; words are defined as sequences of such phones[1] (see Appendix A for the Sphinx-II phone set and examples). Each phone is, in turn, modelled by an HMM (described in greater detail in Section 2.1.2).

As mentioned earlier, natural continuous speech has strong co-articulatory effects. Informally, a phone models the position of various articulators in the mouth and nasal passage (such as the tongue and the lips) in the making of a particular sound. Since these articulators have to move smoothly between different sounds in producing speech, each phone is influenced by the neighbouring ones, especially during the transition from one phone to the next. This is not a major concern in small vocabulary systems in which words are not easily confusable, but becomes an issue as the vocabulary size and the degree of confusability increase.

[1] Some systems define word pronunciations as *networks* of phones instead of simple linear sequences [36].

Most systems employ *triphones* as one form of *context-dependent* HMM models [4, 33] to deal with this problem. Triphones are basically phones observed in the context of given preceding and succeeding phones. There are approximately 50 phones in spoken English language. Thus, there can be a total of about 50^3 triphones, although only a fraction of them are actually observed in the language. Limiting the vocabulary can further reduce this number. For example, in Sphinx-II, a 20,000 word vocabulary has about 75,000 distinct triphones, each of which is modelled by a 5-state HMM, for a total of about 375,000 states. Since there isn't sufficient training data to build models for each state, they are clustered into equivalence classes called *senones* [27].

The introduction of context-dependent acoustic models, even after clustering into equivalence classes, creates an explosion in the memory requirements to store such models. For example, the Sphinx-II system with 10,000 senones occupies tens of megabytes of memory.

Language Models

Large vocabulary continuous speech recognition requires the use of a language model or *grammar* to select the most likely word sequence from the relatively large number of alternative word hypotheses produced during the search process. As mentioned earlier, the absence of explicit word boundary markers in continuous speech causes several additional word hypotheses to be produced, in addition to the intended or correct ones. For example, the phrase *It's a nice day* can be equally well recognized as *It sun iced A.* or *It son ice day.* They are all acoustically indistinguishable, but the word boundaries have been drawn at a different set of locations in each case. Clearly, many more alternatives can be produced with varying degrees of likelihood, given the input speech. The language model is necessary to pick the most likely sequence of words from the available alternatives.

Simple tasks, in which one is only required to recognize a constrained set of phrases, can use rule-based regular or context-free grammars which can be represented compactly. However, that is impossible with large vocabulary tasks. Instead, *bigram* and *trigram* grammars, consisting of word pairs and triples with given probabilities of occurrence, are most commonly used. One can also build such language models based on word *classes*, such as city names, months of the year, etc. However, creating such grammars is tedious as they require a fair amount of hand compilation of the classes. Ordinary word n-gram language models, on the other hand, can be created almost entirely automatically from a corpus of training text.

Clearly, it is infeasible to create a complete set of word bigrams for even medium vocabulary tasks. Thus, the set of bigram and trigram probabilities actually present in a given grammar is usually a small subset of the possible number. Even then, they usually number in the millions for large vocabulary tasks. The memory requirements for such language models range from several tens to hundreds of megabytes.

1.2 The Search Problem

There are two components to the computational cost of speech recognition: acoustic probability computation, and search. In the case of HMM-based systems, the former refers to the computation of the probability of a given HMM state emitting the observed speech at a given time. The latter refers to the search for the best word sequence given the complete speech input. The search cost is largely unaffected by the complexity of the acoustic models. It is much more heavily influenced by the size of the task. As we shall see later, the search cost is significant for medium and large vocabulary recognition; it is the main focus of this thesis.

Speech recognition—searching for the most likely sequence of words given the input speech—gives rise to an exponential search space if all possible sequences of words are considered. The problem has generally been tackled in two ways: *Viterbi* decoding [62, 52] using *beam search* [37], or *stack decoding* [9, 50] which is a variant of the A* algorithm [42]. Some hybrid versions that combine Viterbi decoding with the A* algorithm also exist [21].

Viterbi Decoding

Viterbi decoding is a dynamic programming algorithm that searches the state space for the most likely *state sequence* that accounts for the input speech. The state space is constructed by creating word HMM models from its constituent phone or triphone HMM models, and all word HMM models are searched in parallel. Since the state space is huge for even medium vocabulary applications, the beam search heuristic is usually applied to limit the search by pruning out the less likely states. The combination is often simply referred to as *Viterbi beam search*. Viterbi decoding is a *time-synchronous* search that processes the input speech one *frame* at a time, updating all the states for that frame before moving on to the next frame. Most systems employ a frame input rate of 100 frames/sec. Viterbi decoding is described in greater detail in Section 2.3.1.

Stack Decoding

Stack decoding maintains a stack of partial hypotheses[2] sorted in descending order of posterior likelihood. At each step it pops the best one off the stack. If it is a complete hypothesis it is output. Otherwise the algorithm expands it by one word, trying all

[2] A partial hypothesis accounts for an initial portion of the input speech. A complete hypothesis, or simply hypothesis, accounts for the entire input speech.

possible word extensions, evaluates the resulting (partial) hypotheses with respect to the input speech and re-inserts them in the sorted stack. Any number of N-best hypotheses [59] can be generated in this manner. To avoid an exponential growth in the set of possible word sequences in medium and large vocabulary systems, partial hypotheses are expanded only by a limited set of candidate words at each step. These candidates are identified by a fast match step [6, 7, 8, 20]. Since our experiments have been mostly confined to Viterbi decoding, we do not explore stack decoding in any greater detail.

Tree Structured Lexicons

Even with the beam search heuristic, straightforward Viterbi decoding is expensive. The network of states to be searched is formed by a linear sequence of HMM models for each word in the vocabulary. The number of models actively searched in this organization is still one to two orders of magnitude beyond the capabilities of modern workstations.

Lexical trees can be used to reduce the size of the search space. Since many words share common pronunciation prefixes, they can also share models and avoid duplication. Trees were initially used in fast match algorithms for producing candidate word lists for further search. Recently, they have been introduced in the main search component of several systems [44, 39, 43, 3]. The main problem faced by them is in using a language model. Normally, transitions between words are accompanied by a *prior* language model probability. But with trees, the destination nodes of such transitions are not individual words but entire groups of them, related phonetically but quite unrelated grammatically. An efficient solution to this problem is one of the important contributions of this thesis.

Multipass Search Techniques

Viterbi search algorithms usually also create a *word lattice* in addition to the best recognition hypothesis. The lattice includes several alternative words that were recognized at any given time during the search. It also typically contains other information such as the time *segmentations* for these words, and their posterior acoustic scores (i.e., the probability of observing a word given that time segment of input speech). The *lattice error rate* measures the number of correct words missing from the lattice around the expected time. It is typically much lower than the word error rate[3] of the single best hypotheses produced for each sentence.

Word lattices can be kept very compact, with low *lattice error rate*, if they are produced using sufficiently detailed acoustic models (as opposed to primitive models

[3]Word error rates are measured by counting the number of word substitutions, deletions, and insertions in the hypothesis, compared to the correct reference sentence.

as in, for example, fast match algorithms). In our work, a 10sec long sentence typically produces a word lattice containing about 1000 word instances.

Given such compact lattices with low error rates, one can search them using sophisticated models and search algorithms very efficiently and obtain results with a lower word error rate, as described in [38, 65, 41]. Most systems use such multipass techniques.

However, there has been relatively little work reported in actually creating such lattices efficiently. This is important for the practical applicability of such techniques. Lattices can be created with low computational overhead if we use simple models, but their size must be large to guarantee a sufficiently low lattice error rate. On the other hand, compact, low-error lattices can be created using more sophisticated models, at the expense of more computation time. The efficient creation of compact, low-error lattices for efficient postprocessing is another byproduct of this work.

1.3 Thesis Contributions

This thesis explores ways of improving the performance of speech recognition systems along the dimensions of recognition speed and efficiency of memory usage, while preserving the recognition accuracy of research systems. As mentioned earlier, this is a much harder problem than if we are allowed to trade recognition accuracy for improvement in speed and memory usage.

In order to make meaningful comparisons, the baseline performance of an established "research" system is first measured. We use the CMU Sphinx-II system as the baseline system since it has been extensively used in the yearly ARPA evaluations. It has known recognition accuracy on various test sets, and with similarities to many other research systems. The parameters measured include, in addition to recognition accuracy, the CPU usage of various steps during execution, frequency counts of the most time-consuming operations, and memory usage. All tests are carried out using two vocabulary sizes of about 20,000 (*20K*) and 58,000 (*58K*) words, respectively. The test sentences are taken from the ARPA evaluations in 1993 and 1994 [45, 46].

The results from this analysis show that the search component is several tens of times slower than real time on the reported tasks. (The acoustic output probability computation is relatively smaller since these tests have been conducted using semi-continuous acoustic models [28, 27].) Furthermore, the search time itself can be further decomposed into two main components: the evaluation of HMM models, and carrying out cross-word transitions at word boundaries. The former is simply a measure of the task complexity. The latter is a significant problem since there are cross-word transitions to every word in the vocabulary, and language model probabilities must be computed for every one of them.

1.3.1 Improving Speed

The work presented in this thesis shows that a new adaptation of lexical tree search can be used to reduce both the number of HMMs evaluated and the cost of cross-word transitions. In this method, language model probabilities for a word are computed not when entering that word but upon its exit, if it is one of the recognized candidates. The number of such candidates at a given instant is on average about two orders of magnitude smaller than the vocabulary size. Furthermore, the proportion appears to decrease with increasing vocabulary size.

Using this method, the execution time for recognition is decreased by a factor of about 4.8 for both the 20K and 58K word tasks. If we exclude the acoustic output probability computation, the speedup of the search component alone is about 6.3 for the 20K word task and over 7 for the 58K task. It also demonstrates that the lexical tree search efficiently produces compact word lattices with low error rates that can again be efficiently searched using more complex models and search algorithms.

Even though there is a relative loss of accuracy of about 20% using this method, we show that it can be recovered efficiently by postprocessing the word lattice produced by the lexical tree search. The loss is attributed to suboptimal word segmentations produced by the tree search. However, a new shortest-path graph search formulation for searching the word lattice can reduce the loss in accuracy to under 10% relative to the baseline system with a negligible increase in computation.

If the lattice is first rescored to obtain better word segmentations, all the loss in accuracy is recovered. The rescoring step adds less than 20% execution time overhead, giving an effective overall speedup of about 4 over the baseline system.

We have applied a new *phonetic fast match* step to the lexical tree search that performs an initial pruning of the context independent phones to be searched. This technique reduces the overall execution time by about 40-45%, with a less than 2% relative loss in accuracy. This brings the overall speed of the system to about 8 times that of the baseline system, with almost no loss of accuracy.

The structure of the final decoder is a pipeline of several stages which can be operated in an overlapped fashion. Parallelism among stages, especially the lexical tree search and rescoring passes, is possible for additional improvement in speed.

1.3.2 Reducing Memory Size

The two main candidates for memory usage in the baseline Sphinx-II system, and most of the common research systems, are the acoustic and language models.

The key observation for reducing the size of the language models is that in decoding any given utterance, only a small portion of it is actually used. Hence, we can consider maintaining the language model entirely on disk, and retrieving only the necessary information on demand. Caching schemes can overcome the large disk-access latencies. One might expect the virtual memory systems to perform this function automatically. However, they don't appear to be efficient at managing the language model working set since the granularity of access to the related data structures is much smaller than a pagesize.

We have implemented simple caching rules and replacement policies for bigrams and trigrams, which show that the memory resident portion of large bigram and trigram language models can be reduced significantly. In our benchmarks, the number of bigrams in memory is reduced to about 15-25% of the total, and that of trigrams to about 2-5% of the total. The impact of disk accesses on elapsed time performance is minimal, showing that the caching policies are effective. We believe that further reductions in size can be easily obtained by various compression techniques, such as a reduction in the precision of representation.

The size of the acoustic models is trivially reduced by a factor of 4, simply by reducing the precision of their representation from 32 bits to 8 bits, with no difference in accuracy. This has, in fact, been done in many other systems as in [25]. The new observation is that in addition to memory size reduction, the smaller precision also allows us to speed up the computation of acoustic output probabilities of senones every frame. The computation involves the summation of probabilities—in log-domain, which is cumbersome. The 8-bit representation of such operands allows us to achieve this with a simple table lookup operation, improving the speed of this step by about a factor of 2.

1.4 Summary and Dissertation Outline

In summary, this thesis presents a number of techniques for improving the speed of the baseline Sphinx-II system by about an order of magnitude, and reducing its memory requirements by a factor of 4, without significant loss of accuracy. In doing so, it demonstrates several facts:

- It is possible to build efficient speech recognition systems comparable to research systems in accuracy.
- It is possible to separate concerns of search complexity from that of modelling complexity. By using semi-continuous acoustic models and efficient search strategies to produce compact word lattices with low error rates, and restricting the more detailed models to search such lattices, the overall performance of the system is optimized.
- It is necessary and possible to make decisions for pruning large portions of the search space away with low cost and high reliability. The beam search heuristic

is a well known example of this principle. The phonetic fast match method and the reduction in precision of probability values also fall under this category.

The organization of this thesis is as follows. Chapter 2 contains background material and brief descriptions of related work done in this area. Since recognition speed and memory efficiency has not been an explicit consideration in the research community so far, in the way that recognition accuracy has been, there is relative little material in this regard.

Chapter 3 is mainly concerned with establishing baseline performance figures for the Sphinx-II research system. It includes a comprehensive description of the baseline system, specifications of the benchmark tests and experimental conditions used throughout this thesis, and detailed performance figures, including accuracy, speed and memory requirements.

Chapter 4 is one of the main chapter in this thesis that describes all of the new techniques to speed up recognition and their results on the benchmark tests. Both the baseline and the improved system use the same set of acoustic and language models.

Techniques for memory size reduction and corresponding results are presented in Chapter 5. It should be noted that most experiments reported in this thesis were conducted with these optimizations in place.

Though this thesis is primarily concerned with large vocabulary recognition, it is interesting to consider the applicability of the techniques developed here to smaller vocabulary situations. Chapter 6 addresses the concerns relating to small and extremely small vocabulary tasks. The issues of efficiency are quite different in their case, and the problems are also different. The performance of both the baseline Sphinx-II system and the proposed experimental system are evaluated and compared on the ATIS (Airline Travel Information Service) task, which has a vocabulary of about 3,000 words.

Finally, Chapter 7 concludes with a summary of the results, contributions of this thesis and some thoughts on future directions for search algorithms.

Chapter 2

Background

This chapter contains a brief review of the necessary background material to understand the commonly used modelling and search techniques in speech recognition. Sections 2.1 and 2.2 cover basic features of statistical acoustic and language modelling, respectively. Viterbi decoding using beam search is described in Section 2.3, while related research on efficient search techniques is covered in Section 2.4.

2.1 Acoustic Modelling

2.1.1 Phones and Triphones

The objective of speech recognition is the transcription of speech into text, i.e., word strings. To accomplish this, one might wish to create *word models* from training data. However, in the case of large vocabulary speech recognition, there are simply too many words to be trained in this way. It is necessary to obtain several samples of every word from several different speakers, in order to create reasonable speaker-independent models for each word. Furthermore, the process must be repeated for each new word that is added to the vocabulary.

The problem is solved by creating acoustic models for *sub-word units*. All words are composed of basically a small set of sounds or sub-word units, such as syllables or phonemes, which can be modelled and shared across different words.

Phonetic models are the most frequently used sub-word models. There are only about 50 phones in spoken English (see Appendix A for the set of phones used in Sphinx-II). New words can simply be added to the vocabulary by defining their pronunciation in terms of such phones.

The production of sound corresponding to a phone is influenced by neighbouring phones. For example, the `AE` phone in the word "m*a*n" sounds different from that in "l*a*ck"; the former is more nasal. IBM [4] proposed the use of *triphone* or context-dependent phone models to deal with such variations. With 50 phones, there can be up to 50^3 triphones, but only a fraction of them are actually observed in practice. Virtually all speech recognition systems now use such context dependent models.

2.1.2 HMM modelling of Phones and Triphones

Most systems use hidden Markov models (HMMs) to represent the basic units of speech. The usage and training of HMMs has been covered widely in the literature. Initially described by Baum in [11], it was first used in speech recognition systems by CMU [10] and IBM [29]. The use of HMMs in speech has been described, for example, by Rabiner [52]. Currently, almost all systems use HMMs for modelling triphones and context-independent phones (also referred to as *mono*phones or *base*phones). These include BBN [41], CMU [35, 27], the Cambridge HTK system [65], IBM [5], and LIMSI [18], among others. We will give a brief description of HMMs as used in speech.

First of all, the sampled speech input is usually preprocessed, through various signal-processing steps, into a cepstrum or other *feature stream* that contains one feature vector every *frame*. Frames are typically spaced at 10msec intervals. Some systems produce multiple, parallel feature streams. For example, Sphinx has 4 feature streams—cepstra, Δcepstra, $\Delta\Delta$cepstra, and power—representing the speech signal (see Section 3.1.1).

An HMM is a set of states connected by transitions (see Figure 3.2 for an example). Transitions model the emission of one frame of speech. Each HMM transition has an associated *output probability function* that defines the probability of emitting the input feature observed in any given frame while taking that transition. In practice, most systems associate the output probability function with the source or destination state of the transition, rather than the transition itself. Henceforth, we shall assume that the output probability is associated with the source state. The output probability for state i at time t is usually denoted by $b_i(t)$. (Actually, b_i is not a function of t, but rather a function of the input speech, which is a function of t. However, we shall often use the notation $b_i(t)$ with this implicit understanding.)

Each HMM transition from any state i to state j also has a static *transition probability*, usually denoted by a_{ij}, which is independent of the speech input.

Thus, each HMM state occupies or represents a small subspace of the overall feature space. The shape of this subspace is sufficiently complex that it cannot be accurately characterized by a simple mathematical distribution. For mathematical tractability, the most common general approach has been to model the state output probability by a *mixture Gaussian codebook*. For any HMM state s and feature stream f, the i-th component of such a codebook is a normal distribution with mean vector $\boldsymbol{\mu}_{s,f,i}$ and covariance matrix $\boldsymbol{U}_{s,f,i}$. In order to simplify the computation and also because there is often insufficient data to estimate all the parameters of the covariance matrix, most systems assume independence of dimensions and therefore the covariance matrix becomes diagonal. Thus, we can simply use standard deviation vectors $\boldsymbol{\sigma}_{s,f,i}$ instead of $\boldsymbol{U}_{s,f,i}$. Finally, each such mixture component also has a scalar *mixture coefficient* or *mixture weight* $w_{s,f,i}$.

With that, the probability of observing a given speech input $\mathbf{x}$ in HMM state s is given by:

$$b_s(\mathbf{x}) = \prod_f (\sum_i w_{s,f,i} \mathcal{N}(\mathbf{x}_f, \boldsymbol{\mu}_{s,f,i}, \boldsymbol{\sigma}_{s,f,i})) \tag{2.1}$$

where the speech input $\mathbf{x}$ is the parallel set of feature vectors, and $\mathbf{x}_f$ its f-th feature component; i ranges over the number of Gaussian densities in the mixture and f over the number of features. The expression $\mathcal{N}(.)$ is the value of the chosen component Gaussian density function at $\mathbf{x}_f$.

In the general case of *fully continuous* HMMs, each HMM state s in the acoustic model has its own separate weighted mixture Gaussian codebook. However, this is computationally expensive, and many schemes are used to reduce this cost. It also results in too many free parameters. Most systems group HMM states into clusters that share the same set of model parameters. The sharing can be of different degrees. In *semi-continuous* systems, all states share a single mixture Gaussian codebook, but the mixture coefficients are distinct for individual states. In Sphinx-II, states are grouped into clusters called *senones* [27], with a single codebook (per feature stream) shared among all senones, but distinct mixture weights for each. Thus, Sphinx-II uses semi-continuous modelling with state clustering.

Even simpler *discrete* HMM models can be derived by replacing the mean and variance vectors representing Gaussian densities with a single centroid. In every frame, the single closest centroid to the input feature vector is computed (using the Euclidean distance measure), and individual states weight the codeword so chosen. Discrete models are typically only used in making approximate searches such as in fast match algorithms.

For simplicity of modelling, HMMs can have NULL transitions that do not consume any time and hence do not model the emission of speech. Word HMMs can be built by simply stringing together phonetic HMM models using NULL transitions as appropriate.

2.2 Language Modelling

As mentioned in Chapter 1, a language model (LM) is required in large vocabulary speech recognition for disambiguating between the large set of alternative, confusable words that might be hypothesized during the search.

The LM defines the *a priori* probability of a sequence of words. The LM probability of a sentence (i.e., a sequence of words $w_1, w_2, \ldots, w_n$) is given by:

$$P(w_1)P(w_2|w_1)P(w_3|w_1,w_2)P(w_4|w_1,w_2,w_3)\cdots P(w_n|w_1,\ldots,w_{n-1})$$

$$= \prod_{i=1}^{n} P(w_i|w_1,\ldots,w_{i-1}).$$

In an expression such as $P(w_i|w_1,\ldots,w_{i-1})$, $w_1,\ldots,w_{i-1}$ is the *word history* or simply *history* for w_i. In practice, one cannot obtain reliable probability estimates given arbitrarily long histories since that would require enormous amounts of training data. Instead, one usually approximates them in the following ways:

- Context free grammars or regular grammars. Such LMs are used to define the form of well structured sentences or phrases. Deviations from the prescribed structure are not permitted. Such formal grammars are never used in large vocabulary systems since they are too restrictive.

- Word *unigram*, *bigram*, *trigram*, grammars. These are defined respectively as follows (higher-order n-grams can be defined similarly):

  $$\begin{array}{lcl} P(w) & = & \text{probability of word } w \\ P(w_j|w_i) & = & \text{probability of } w_j \text{ given a one word history } w_i \\ P(w_k|w_i,w_j) & = & \text{probability of } w_k \text{ given a two word history } w_i, w_j \end{array}$$

  A bigram grammar need not contain probabilities for all possible word pairs. In fact, that would be prohibitive for all but the smallest vocabularies. Instead, it typically lists only the most frequently occurring bigrams, and uses a *backoff* mechanism to fall back on unigram probability when the desired bigram is not found. In other words, if $P(w_j|w_i)$ is sought and is not found, one falls back on $P(w_j)$. But a *backoff weight* is applied to account for the fact that w_j is known to be not one of the bigram successors of w_i [30]. Other higher-order backoff n-gram grammars can be defined similarly.

- Class n-gram grammars. These are similar to word n-gram grammars, except that the tokens are entire word classes, such as digit, number, month, proper name, etc. The creation and use of class grammars is tricky since words can belong to multiple classes. There is also a fair amount of handcrafting involved.

- Long distance grammars. Unlike n-gram LMs, these are capable of relating words separated by some distance (i.e., with some intervening words). For example, the *trigger-pair* mechanism discussed in [57] is of this variety. Long distance grammars are primarily used to rescore n-best hypothesis lists from previous decodings.

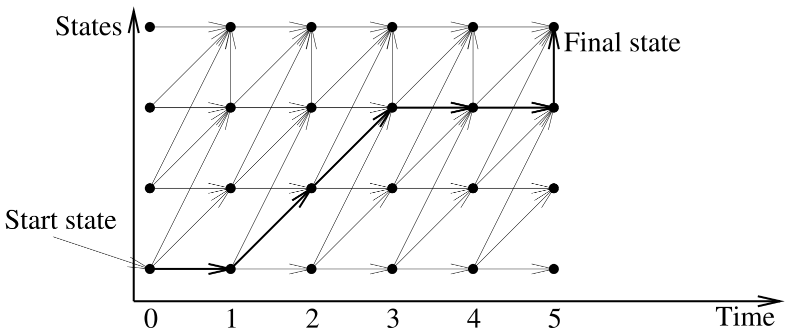


Figure 2.1: Viterbi Search as Dynamic Programming

Of the above, word bigram and trigram grammars are the most commonly used since they are easy to train from large volumes of data, requiring minimal manual intervention. They have also provided high degrees of recognition accuracy. The Sphinx-II system uses word trigram LMs.

2.3 Search Algorithms

The two main forms of decoding most commonly used today are Viterbi decoding using the beam search heuristic, and stack decoding. Since the work reported in this thesis is based on the former, we briefly review its basic principles here.

2.3.1 Viterbi Beam Search

Viterbi search [62] is essentially a dynamic programming algorithm, consisting of traversing a network of HMM states and maintaining the best possible path score at each state in each frame. It is a time-synchronous search algorithm in that it processes all states completely at time t before moving on to time $t+1$.

The abstract algorithm can be understood with the help of Figure 2.1. One dimension represents the states in the network, and the other is the time axis. There is typically one start state and one or more final states in the network. The arrows depict possible state transitions throughout the network. In particular, NULL transitions go vertically since they do not consume any input, and non-NULL transitions always go one time step forward. Each point in this 2-D space represents the best *path probability* for the corresponding state at that time. That is, given a time t and state s, the value at (t, s) represents the probability corresponding to the best state sequence leading from the initial state at time 0 to state s at time t.

The time-synchronous nature of the Viterbi search implies that the 2-D space is traversed from left to right, starting at time 0. The search is initialized at time

$t = 0$ with the path probability at the start state set to 1, and at all other states to 0. In each frame, the computation consists of evaluating all transitions between the previous frame and the current frame, and then evaluating all NULL transitions within the current frame. For non-NULL transitions, the algorithm is summarized by the following expression:

$$P_j(t) = \max_i (P_i(t-1) \cdot a_{ij} \cdot b_i(t)), i \epsilon \text{ set of predecessor states of } j \tag{2.2}$$

where, $P_j(t)$ is the path probability of state j at time t, a_{ij} is the static probability associated with the transition from state i to j, and $b_i(t)$ is the output probability associated with state i while consuming the input speech at t (see Section 2.1.2 and equation 2.1). It is straightforward to extend this formulation to include NULL transitions that do not consume any input.

Thus, every state has a single best predecessor at each time instant. With some simple bookkeeping to maintain this information, one can easily determine the best state sequence for the entire search by starting at the final state at the end and following the best predecessor at each step all the way back to the start state. Such an example is shown by the bold arrows in Figure 2.1.

The complexity of Viterbi decoding is N^2T (assuming each state can transition to every state at each time step), where N is the total number of states and T is the total duration.

The application of Viterbi decoding to continuous speech recognition is straightforward. Word HMMs are built by stringing together phonetic HMM models using NULL transitions between the final state of one and the start state of the next. In addition, NULL transitions are added from the final state of each word to the initial state of all words in the vocabulary, thus modelling continuous speech. Language model (bigram) probabilities are associated with every one of these *cross-word* transitions. Note that a system with a vocabulary of V words has V^2 possible cross-word transitions. All word HMMs are searched in parallel according to equation 2.2.

Since even a small to medium vocabulary system consists of hundreds or thousands of HMM states, the state-time matrix of Figure 2.1 quickly becomes too large and costly to compute in its entirety. To keep the computation within manageable limits, only the most likely states are evaluated in each frame, according to the *beam search heuristic* [37]. At the end of time t, the state with the highest path probability $P^{max}(t)$ is found. If any other state i has $P_i(t) < P^{max}(t) \cdot B$, where B is an appropriately chosen threshold or *beamwidth* < 1, state i is excluded from consideration at time $t + 1$. Only the ones within the beam are considered to be *active*.

The beam search heuristic reduces the average cost of search by orders of magnitude in medium and large vocabulary systems. The combination of Viterbi decoding using beam search heuristic is often simply referred to as *Viterbi beam search*.

2.4 Related Work

Some of the standard techniques in reducing the computational load of Viterbi search for large vocabulary continuous speech recognition have been the following:

- Narrowing the beamwidth for greater pruning. However, this is usually associated with an increase in error rate because of an increase in the number of search errors: the correct word sometimes get pruned from the search path in the bargain.
- Reducing the complexity of acoustic and language models. This approach works to some extent, especially if it is followed by more detailed search in later passes. There is a tradeoff here, between the computational load of the first pass and subsequent ones. The use of detailed models in the first pass produces compact word lattices with low error rate that can be postprocessed efficiently, but the first pass itself is computationally expensive. Its cost can be reduced if simpler models are employed, at the cost of an increase in lattice size needed to guarantee low lattice error rates.

Both the above techniques involve some tradeoff between recognition accuracy and speed.

2.4.1 Tree Structured Lexicons

Organizing the HMMs to be searched as a phonetic tree instead of the flat structure of independent linear HMM sequences for each word is probably the most often cited improvement in search techniques in use currently. This structure is referred to as *tree-structured lexicon* or *lexical tree*. If the pronunciations of two or more words contain the same n initial phonemes, they share a single sequence of n HMM models representing that initial portion of their pronunciation. (In practice, most systems use triphones instead of just basephones, so we should really consider triphone pronunciation sequences. But the basic argument is the same.) Since the word-initial models in a non-tree structured Viterbi search are typically the majority of the total number of active models, the reduction in computation is significant.

The problem with a lexical tree occurs at word boundary transitions where bigram language model probabilities are usually computed and applied. In the flat (non-tree) Viterbi algorithm there is a transition from each word ending state (within the beam) to the beginning of every word in the vocabulary. Thus, there is a *fan-in* at the initial state of every word, with different bigram probabilities attached to every such transition. The Viterbi algorithm chooses the best incoming transition in each case.

However, with a lexical tree structure, several words may share the same root node of the tree. There can be a conflict between the best incoming cross-word transition for different words that share the same root node. This problem has been usually solved by making copies of the lexical tree to resolve such conflicts.

Approximate Bigram Trees

SRI [39] and CRIM [43] augment their lexical tree structure with a flat copy of the lexicon that is activated for bigram transitions. All bigram transitions enter the flat lexicon copy, while the backed off unigram transitions enter the roots of the lexical tree. SRI notes that relying on just unigrams more than doubles the word error rate. They show that using this scheme, the recognition speed is improved by a factor of 2-3 for approximately the same accuracy. To gain further improvements in speed, they reduce the size of the bigram section by pruning the bigram language model in various ways, which adds significantly to the error rate.

However, it should be noted that the experimental set up is based on using discrete HMM acoustic models, with a baseline system word error rate (21.5%), which is significantly worse than their best research system (10.3%) using bigrams, and also worse than most other research systems to begin with.

As we shall see in Chapter 3, bigram transitions constitute a significant portion of cross word transitions, which in turn are a dominant part of the search cost. Hence, the use of a flat lexical structure for bigram transitions must continue to incur this cost.

Replicated Bigram Trees

Ney and others [40, 3] have suggested creating copies of the lexical tree to handle bigram transitions. The leaf nodes at the first level (unigram) lexical tree have secondary (bigram) trees hanging off them for bigram transitions. The total size of the secondary trees depends on the number of bigrams present in the grammar. Secondary trees that represent the bigram followers of the most common *function* words, such as `A`, `THE`, `IN`, `OF`, etc. are usually large.

This scheme creates additional copies of words that did not exist in the original flat structure. For example, in the conventional flat lexicon (or in the auxiliary flat lexicon copy of [39]), there is only one instance of each word. However, in this proposed scheme the same word can appear in multiple secondary trees. Since the short *function* words are recognized often (though spuriously), their bigram copies are frequently active. They are also among the larger ones, as noted above. It is unclear how much overhead this adds to the system.

Dynamic Network Decoding

Cambridge University [44] designed a one-pass decoder that uses the lexical tree structure, with copies for cross-word transitions, but instantiates new copies at every transition, as necessary. Basically, the traditional re-entrant lexical structure is replaced with a non-re-entrant structure. To prevent an explosion in memory space requirements, they reclaim HMM nodes as soon as they become inactive by falling outside the pruning beamwidth. Furthermore, the end points of multiple instances of the same word can be merged under the proper conditions, allowing just one instance of the lexical tree to be propagated from the merged word ends, instead of separately and multiply from each. This system attained the highest recognition accuracy in the Nov 1993 evaluations.

They report the performance under standard conditions—standard 1993 *20K Wall Street Journal* development test set decoded using the corresponding standard bigram/trigram language model using wide beamwidths as in the actual evaluations.

The number of active HMM models per frame in this scheme is actually higher than the number in the baseline Sphinx-II system under similar test conditions (except that Sphinx-II uses a different lexicon and acoustic models). There are other factors at work, but the dynamic instantiation of lexical trees certainly plays a part in this increase. The overhead for dynamically constructing the HMM network is reported to be less than 20% of the total computational load. This is actually fairly high since the time to decode a sentence on an HP735 platform is reported to be about 15 minutes on average.

2.4.2 Memory Size and Speed Improvements in Whisper

The CMU Sphinx-II system has been improved in many ways by Microsoft in producing the Whisper system [26]. They report that memory size has been reduced by a factor of 20 and speed improved by a factor of 5, compared to Sphinx-II under the same accuracy constraints.

One of the schemes for memory reduction is the use of a context free grammar (CFG) in place of bigram or trigram grammars. CFGs are highly compact, can be searched efficiently, and can be relatively easily created for small tasks such as command and control applications involving a few hundred words. However, large vocabulary applications cannot be so rigidly constrained.

They also obtain an improvement of about 35% in the memory size of acoustic models by using run length encoding for senone weighting coefficients (Section 2.1.2).

They have also improved the speed performance of Whisper through a *Rich Get Richer* (RGR) heuristic for deciding which phones should be evaluated in detail, using triphone states, and which should fall back on context independent phone states.

RGR works as follows: Let $P_p(t)$ be the best path probability of any state belonging to basephone p at time t, $P^{max}(t)$ the best path probability over all states at t, and $b_p(t+1)$ the output probability of the context-independent model for p at time $t+1$. Then, the context-dependent states for phone p are evaluated at frame $t+1$ iff:

$$a \cdot P_p(t) + b_p(t+1) > P^{max}(t) - K$$

where, a and K are empirically determined constants. Otherwise, context-independent output probabilities are used for those states. (All probabilities are computed in log-space. Hence the addition operations really represent multiplications in normal probability space.)

Using this heuristic, they report an 80% reduction in the number of context dependent states for which output probabilities are computed, with no loss of accuracy. If the parameters a and K are tightened to reduce the number of context-dependent states evaluated by 95%, there is a 15% relative loss of accuracy. (The baseline test conditions have not be specified for these experiments.)

2.4.3 Search Pruning Using Posterior Phone Probabilities

In [56], Renals and Hochberg describe a method of deactivating certain phones during search to achieve higher recognition speed. The method is incorporated into a fast match pass that produces words and posterior probabilities for their NOWAY stack decoder. The fast match step uses HMM base phone models, the states of which are modelled by neural networks that directory estimate phone posterior probabilities instead of the usual likelihoods; i.e., they estimate $P(phone|data)$, instead of $P(data|phone)$. Using the posterior phone probability information, one can identify the less likely active phones at any given time and prune the search accordingly.

This is a potentially powerful and easy pruning technique when the posterior phone probabilities are available. Stack decoders can particularly gain if the fast match step can be made to limit the number of candidate words emitted while extending a partial hypothesis. In their NOWAY implementation, a speedup of about an order of magnitude is observed on a 20K vocabulary task (from about 150x real time to about 15x real time) on an HP735 workstation. They do not report the reduction in the number of active HMMs as a result of this pruning.

2.4.4 Lower Complexity Viterbi Algorithm

A new approach to the Viterbi algorithm, specifically applicable to speech recognition, is described by Patel in [49]. It is aimed at reducing the cost of the large number of cross-word transitions and has an expected complexity of $N\sqrt{N}T$, instead of N^2T (Section 2.3.1). The algorithm depends on ordering the exit path probabilities and transition bigram probabilities, and finding a threshold such that most transitions can be eliminated from consideration.

The authors indicate that the algorithm offers better performance if every word has bigram transitions to the entire vocabulary. However, this is not the case with large vocabulary systems. Nevertheless, it is worth exploring this technique further for its practical applicability.

2.5 Summary

In this chapter we have covered the basic modelling principles and search techniques commonly used in speech recognition today. We have also briefly reviewed a number of systems and techniques used to improve their speed and memory requirements. One of the main themes running through this work is that virtually none of the practical implementations have been formally evaluated with respect to the research systems on well established test sets under widely used test conditions, or with respect to one another.

In the rest of this thesis, we evaluate the baseline Sphinx-II system under normal evaluation conditions and use the results for comparison with our other experiments.

Chapter 3

The Sphinx-II Baseline System

As mentioned in the previous chapters, there is relatively little published work on the performance of speech recognition systems, measured along the dimensions of recognition accuracy, speed and resource utilization. The purpose of this chapter is to establish a comprehensive account of the performance of a baseline system that has been considered a premier representative of its kind, with which we can make meaningful comparisons of the research reported in this thesis. For this purpose, we have chosen the *Sphinx-II* speech recognition system[1] at Carnegie Mellon that has been used extensively in speech research and the yearly ARPA evaluations. Various aspects of this baseline system and its precursors have been reported in the literature, notably in [32, 33, 35, 28, 1, 2]. Most of these concentrate on the modelling aspects of the system—acoustic, grammatical or lexical—and their effect on recognition accuracy. In this chapter we focus on obtaining a comprehensive set of performance characteristics for this system.

The baseline Sphinx-II recognition system uses *semi-continuous* or tied-mixture hidden Markov models (HMMs) for the acoustic models [52, 27, 12] and word bigram or trigram backoff language models (see Sections 2.1 and 2.2). It is a 3-pass decoder structured as follows:

1. Time synchronous Viterbi beam search [52, 62, 37] in the forward direction. It is a complete search of the full vocabulary, using semi-continuous acoustic models, a bigram or trigram language model, and cross-word triphone modelling during the search. The result of this search is a single recognition hypothesis, as well as a *word lattice* that contains all the words that were recognized during the search. The lattice includes word segmentation and scores information. One of the key features of this lattice is that for each word occurrence, several successive *end* times are identified along with their scores, whereas very often only the single most likely *begin* time is identified. Scores for alternative begin times are usually

[1] The Sphinx-II decoder reported in this section is known internally as FBS6.

not available.

2. Time synchronous Viterbi beam search in the backward direction. This search is restricted to the words identified in the forward pass and is very fast. Like the first pass, it produces a word lattice with word segmentations and scores. However, this time several alternative begin times are identified while typically only one end time is available. In addition, the Viterbi search also produces the best path score from any point in the utterance to the end of the utterance, which is used in the third pass.

3. An A* or stack search using the word segmentations and scores produced by the forward and backward Viterbi passes above. It produces an N-best list [59] of alternative hypotheses as its output, as described briefly in Section 1.2. There is no acoustic rescoring in this pass. However, any arbitrary language model can be applied in creating the N-best list. In this thesis, we will restrict our discussion to word trigram language models.

The reason for the existence of the backward and A* passes, even though the first pass produces a usable recognition result, is the following. One limitation of the forward Viterbi search in the first pass is that it is hard to employ anything more sophisticated than a simple bigram or similar grammar. Although a trigram grammar is used in the forward pass, it is not a complete trigram search (see Section 3.2.2). Stack decoding, a variant of the A* search algorithm[2] [42], is more appropriate for use with such grammars which lead to greater recognition accuracy. This algorithm maintains a stack of several possible partial decodings (i.e, word sequence hypotheses) which are expanded in a best-first manner [9, 2, 50]. Since each partial hypothesis is a linear word sequence, any arbitrary language model can be applied to it. Stack decoding also allows the decoder to output several most likely N-best hypotheses rather than just the single best one. These multiple hypotheses can be postprocessed with even more detailed models. The need for the backward pass in the baseline system has been mentioned above.

In this chapter we review the details of the baseline system needed for understanding the performance characteristics. In order to keep this discussion fairly self-contained, we first review the various knowledge source models in Section 3.1. Some of the background material in Sections 2.1, 2.2, and 2.3 is also relevant. This is followed by a discussion of the forward pass Viterbi beam search in Section 3.2, and the backward and A* searches in Section 3.3. The performance of this system on several widely used test sets from the ARPA evaluations is described in Section 3.4. It includes recognition accuracy, various statistics related to search speed, and memory usage. We finally conclude with some final remarks in Section 3.5.

[2]We will often use the terms *stack decoding* and *A* search* interchangeably.

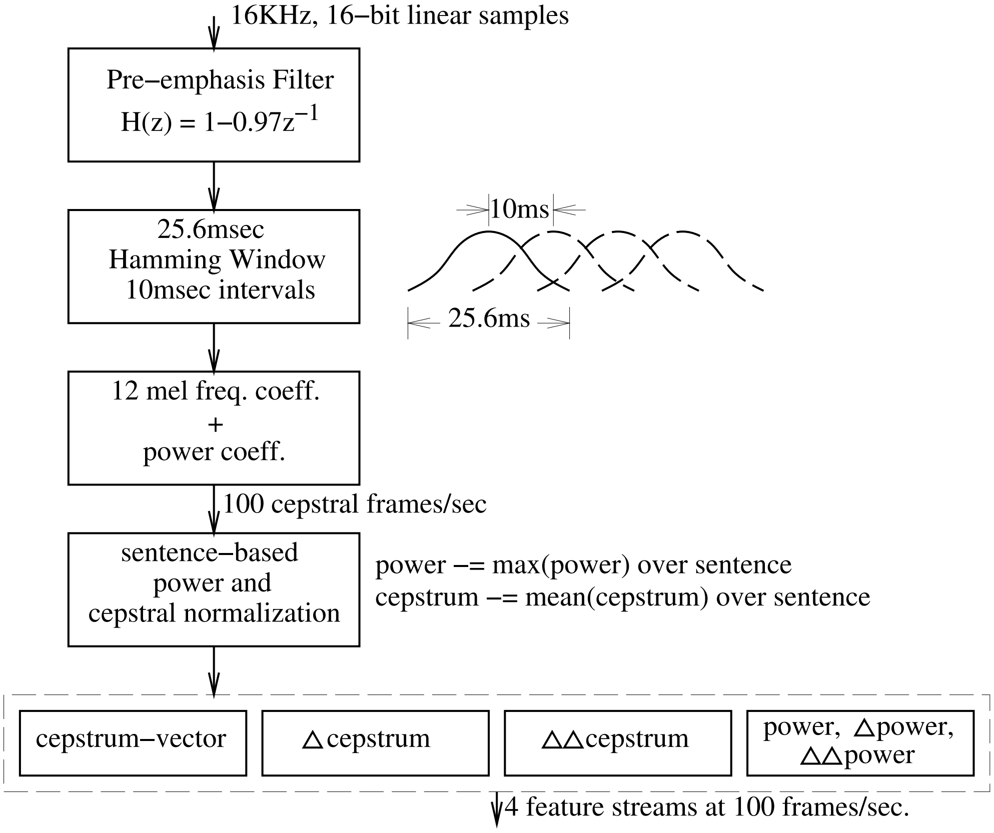


Figure 3.1: Sphinx-II Signal Processing Front End.

3.1 Knowledge Sources

This section briefly describes the various knowledge sources or models and the speech signal processing *front-end* used in Sphinx-II. In addition to the acoustic models and pronunciation lexicon described below, Sphinx-II uses word bigram and trigram grammars. These have been discussed in Section 2.2.

3.1.1 Acoustic Model

Signal Processing

A detailed description of the signal processing front end in Sphinx-II is contained in Section 4.2.1 *Signal Processing* of [27]. The block diagram in Figure 3.1 depicts the overall processing. Briefly, the stream of 16-bit samples of speech data, sampled at 16KHz, is converted into 12-element *mel scale frequency cepstrum* vectors and a *power* coefficient in each 10msec *frame*. We represent the cepstrum vector at time t by $\mathbf{x}(t)$ (individual elements are denoted by $x_k(t), 1 \leq k \leq 12$). The power coefficient

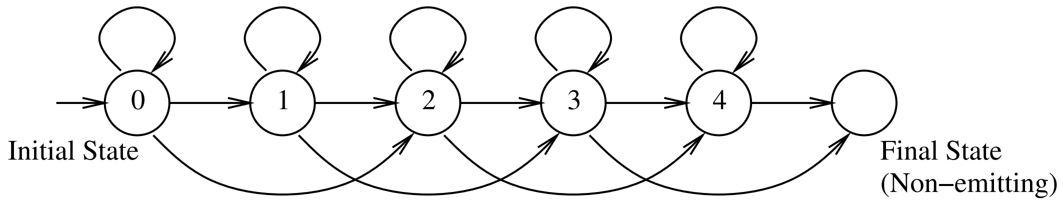


Figure 3.2: Sphinx-II HMM Topology: 5-State Bakis Model.

is simply $x_0(t)$. This cepstrum vector and power streams are first normalized, and four *feature vectors* are derived in each frame by computing the first and second order differences in time:

$$\begin{aligned}
\mathbf{x}(t) &= \text{normalized cepstrum vector} \\
\Delta\mathbf{x}(t) &= \mathbf{x}(t+2) - \mathbf{x}(t-2),\ \Delta_l\mathbf{x}(t) = \mathbf{x}(t+4) - \mathbf{x}(t-4) \\
\Delta\Delta\mathbf{x}(t) &= \Delta\mathbf{x}(t+1) - \Delta\mathbf{x}(t-1) \\
\mathbf{x}_0(t) &= x_0(t), \\
&\quad\ \Delta x_0(t) = x_0(t+2) - x_0(t-2), \\
&\quad\ \Delta\Delta x_0(t) = \Delta x_0(t+1) - \Delta x_0(t-1)
\end{aligned}$$

where the commas denote concatenation. Thus, in every frame we obtain four feature vectors of 12, 24, 12, and 3 elements, respectively. These, ultimately, are the input to the speech recognition system.

Phonetic HMM Models

Acoustic modelling in Sphinx-II is based on hidden Markov models (HMMs) for base-phones and triphones. All HMMs in Sphinx-II have the same 5-state Bakis topology shown in the Figure 3.2. (The background on HMMs has been covered briefly in Section 2.1.2.)

As mentioned in Section 2.1.2, Sphinx-II uses semi-continuous acoustic modelling with 256 component densities in each feature codebook. States are clustered into *senones* [27], where each senone has its own set of 256 mixture coefficients weighting the codebook for each feature stream.

In order to further reduce the computational cost, only the top few component densities from each feature codebook—typically 4—are fully evaluated in each frame in computing the output probability of a state or senone (equation 2.1). The rationale behind this approximation is that the remaining components match the input very poorly anyway and can be ignored altogether. The approximation primarily reduces the cost of applying the mixture weights in computing senone output probabilities in each frame. For each senone and feature only 4 mixing weights have to be applied to the 4 best components, instead of all 256.

3.1.2 Pronunciation Lexicon

The lexicon in Sphinx-II defines the linear sequence of phonemes representing the pronunciation for each word in the vocabulary. There are about 50 phonemes that make up the English language. The phone set used in Sphinx-II is given in Appendix A. The following is a small example of the lexicon for digits:

```
OH          OW
ZERO        Z IH R OW
ZERO(2)     Z IY R OW
ONE         W AH N
TWO         T UW
THREE       TH R IY
FOUR        F AO R
FIVE        F AY V
SIX         S IH K S
SEVEN       S EH V AX N
EIGHT       EY TD
NINE        N AY N
```

There can be multiple pronunciations for a word, as shown for the word `ZERO` above. Each alternative pronunciation is assumed to have the same *a priori* language model probability.

3.2 Forward Beam Search

As mentioned earlier, the baseline Sphinx-II recognition system consists of three passes, of which the first is a time-synchronous Viterbi beam search in the forward direction. In this section we describe the structure of this forward pass. We shall first examine the data structures involved in the search algorithm, before moving on to the dynamics of the algorithm.

3.2.1 Flat Lexical Structure

The lexicon defines the linear sequence of context-independent or base phones that make up the pronunciation of each word in the vocabulary. Since Sphinx-II uses triphone acoustic models [34], these base phone sequences are converted into triphone sequences by simply taking each base phone together with its left and right context base phones. (Note that the phonetic left context at the beginning of a word is the last base phone from the previous word. Similarly, the phonetic right context at the end of the word is the first base phone of the next word. Since the decoder does not know these neighbouring words *a priori*, it must try all possible cases and finally choose the best. This is discussed in detail below.) Given the sequence of triphones for a word, one can construct an equivalent *word-HMM* by simply concatenating the HMMs for the individual triphones, i.e., by adding a NULL transition from the final state of one HMM to the initial state of the next. The initial state of first HMM, and the final state of the last HMM in this sequence become the initial and final states, respectively, of the complete word-HMM. Finally, in order to model continuous speech (i.e., transition from one word into the next), additional NULL transitions are created from the final state of every word to the initial state of all words in the vocabulary. Thus, with a V word vocabulary, there are V^2 possible cross-word transitions.

Since the result is a structure consisting of separate linear sequence of HMMs for each word, we call this a *flat lexical structure*.

3.2.2 Incorporating the Language Model

While the cross-word NULL transitions do not consume any speech input, each of them does have a language model probability associated with it. For a transition from some word w_i to any word w_j, this probability is simply $P(w_j|w_i)$ if a bigram language model is used. A bigram language model fits in neatly with the Markov assumption that given any current state s at time t the probability of transitions out of s does not depend on how one arrived at s. Thus, the language model probability $P(w_j|w_i)$ can be associated with the transition from the final state of w_i to the initial state of w_j and thereafter we need not care about how we arrived at w_j.

The above argument does not hold for a trigram or some other longer distance grammar since the language model probability of transition to w_j depends not only on the immediate predecessor but also some earlier ones. If a trigram language model is used, the lexical structure has to be modified such that for each word w there are several parallel instances of its word HMM, one for each possible predecessor word. Although the copies may score identically acoustically, the inclusion of language model scores would make their total path probabilities distinct. In general, with non-bigram grammars, we need a separate word HMM model for each *grammar state* rather than just one per word in the vocabulary.

Clearly, replicating the word HMM models for incorporating a trigram grammar or some other non-bigram grammar in the search algorithm is much costlier computationally. However, more sophisticated grammars offer greater recognition accuracy and possibly even a reduction in the search space. Therefore, in Sphinx-II, trigram grammars are used in an approximate manner with the following compromise. Whenever there is a transition from word w_i to w_j, we can find the best predecessor of w_i at that point, say w'_i, as determined by the Viterbi search. We then associate the trigram probability $P(w_j|w'_i, w_i)$ with the transition from w_i to w_j. Note, however, that unlike with bigram grammars, trigram probabilities applied to cross-word transitions in this approximate fashion have to be determined dynamically, depending on the best predecessor for each transition at the time in question.

Using a trigram grammar in an approximate manner as described above has the following advantages:

- It avoid any replication of the lexical word-HMM structures and associated increase in computational load.
- In terms of accuracy, it is much better than using a bigram model and is close to that of a complete trigram search. We infer this from the fact that the accuracy of the results from the final A* pass, which uses the trigram grammar correctly, and also has the benefit of additional word segmentations to choose from, is relatively only about 5% better (see Section 3.4.2).
- A trigram grammar applied in this approximate manner is empirically observed to search fewer word-HMMs compared to a bigram grammar, thus leading to a slight improvement in the recognition speed. The reduction in search is a result of sharper pruning offered by the trigram grammar.

3.2.3 Cross-Word Triphone Modeling

It is advantageous to use cross-word triphone models (as opposed to ignoring cross-word phonetic contexts) for continuous speech recognition where word boundaries are unclear to begin with and there are very strong co-articulation effects. Using cross-word triphone models we not only obtain better accuracy, but also greater computational efficiency, at the cost of an increase in the total size of acoustic models. The sharper models provided by triphones, compared to diphones and monophones, leads to greater pruning efficiency and a reduction in computation. However, using cross-word triphone models in the Viterbi search algorithm is not without its complications.

Right Context

The phonetic right context for the last triphone position in a word is the first base phone of the next word. In time-synchronous Viterbi search, there is no way to know the next word in advance. In any case, whatever decoding algorithm is used, there can be several potential successor words to any given word w_i at any given time. Therefore, the last triphone position for each word has to be modelled by a *parallel* set of triphone models, one for each possible phonetic right context. In other words, if there are k basephones $p_1, p_2, \ldots, p_k$ in the system, we have k parallel triphone HMM models $h_{p_1}, h_{p_2}, \ldots, h_{p_k}$ representing the final triphone position for w_i. A cross-word transition from w_i to another word w_j, whose first base phone is p is represented by

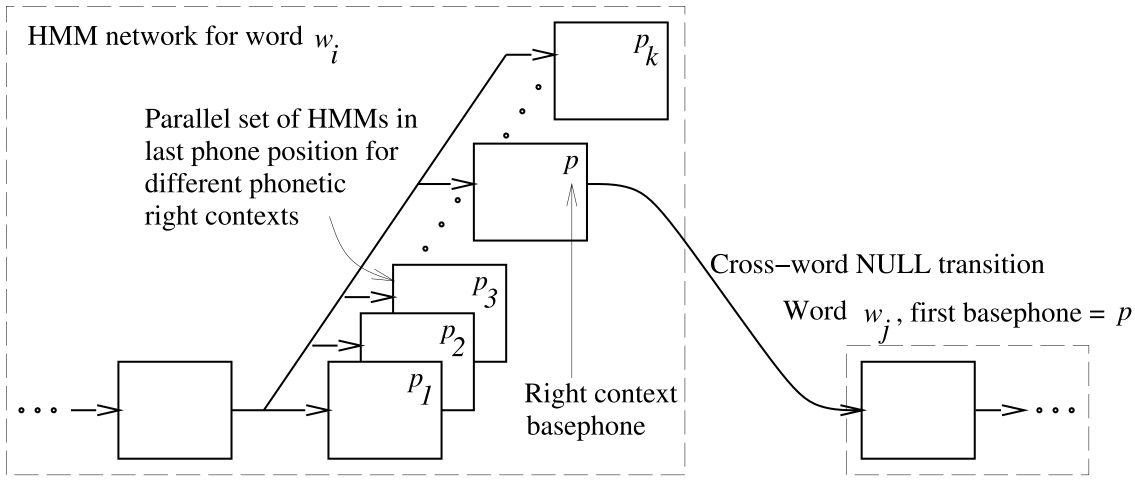


Figure 3.3: Cross-word Triphone Modelling at Word Ends in Sphinx-II.

a NULL arc from h_p to the initial state of w_j. Figure 3.3 illustrates this concept of *right context fanout* at the end of each word w_i in Sphinx-II.

This solution, at first glance, appears to force a large increase in the total number of triphone HMMs that may be searched. In the place of the single last position triphone for each word, we now have one triphone model for each possible phonetic right context, which is typically around 50 in number. In practice, we almost never encounter this apparent explosion in computational load, for the following reasons:

- The dynamic number of rightmost triphones actually evaluated in practice is much smaller than the static number because the *beam* search heuristic prunes most of the words away by the time their last phone has been reached. This is by far the largest source of efficiency, even with the right context fanout.

- The set of phonetic right contexts actually modelled can be restricted to just those found in the input vocabulary; i.e., to the set of first base phones of all the words in the vocabulary.

  Moreover, Sphinx-II uses state clustering into senones, where several states share the same output distribution modelled by a senone. Therefore, the parallel set of models at the end of any given word are not all unique. By removing duplicates, the fanout can be further reduced. In Sphinx-II, these two factors together reduce the right context fanout by about 70% on average.

- The increase is number of rightmost triphones is partly offset by the reduction in computation afforded by the sharper triphone models.

Left Context

The phonetic left context for the first phone position in a word is the last base phone from the previous word. During decoding, there is no unique such predecessor word. In any given frame t, there may be transitions to a word w_j from a number of candidates $w_{i_1}, w_{i_2}, \ldots$ The Viterbi algorithm chooses the best possible transition into w_j. Let us say the winning predecessor is w_{i_k}. Thus, the last base phone of w_{i_k} becomes the phonetic left context for w_j. However, this is in frame t. In the next frame, there may be an entirely different winner that results in a different left context base phone. Since the real best predecessor is not determined until the end of the Viterbi decoding, all such possible paths have to be pursued in parallel.

As with right context cross-word triphone modelling, this problem also can solved by using a parallel set of triphone models for the first phone position of each word—a separate triphone for each possible phonetic left context. However, unlike the word-ending phone position which is heavily pruned by the beam search heuristic, the word-initial position is extensively searched. Most of the word-initial triphone models are alive every frame. In fact, as we shall see later in Section 3.4, they account for more than 60% of all triphone models evaluated in the case of large-vocabulary recognition. A left context fanout of even a small factor of 2 or 3 would substantially slow down the system.

The solution used in the Sphinx-II baseline system is to collapse the left context fanout into a single 5-state HMM with *dynamic triphone mapping* as follows. As described above, at any given frame there may be several possible transitions from words $w_{i_1}, w_{i_2}, \ldots$ into w_j. According to the Viterbi algorithm, the transition with the best incoming score wins. Let the winning predecessor be w_{i_k}. Then the initial state of w_j also *dynamically* inherits the last base phone of w_{i_k} as its left context. When the output probability of the initial state of w_j has to be evaluated in the next frame, its parent triphone identity is first determined dynamically from the inherited left context basephone. Furthermore, this dynamically determined triphone identity is also propagated by the Viterbi algorithm, as the path probability is propagated from state to state. This ensures that any complete path through the initial triphone position of w_j is scored consistently using a single triphone HMM model.

Figure 3.4 illustrates this process with an example, going through a sequence of 4 frames. It contains a snapshot of a word-initial HMM model at the end of each frame. Arcs in bold indicate the winning transitions to each state of the HMM in this example. HMM states are annotated with the left context basephone inherited dynamically through time. As we can see in the example, different states can have different phonetic left contexts associated with them, but a single Viterbi path through the HMM is evaluated with the same context. This can be verified by backtracking from the final state backward in time.

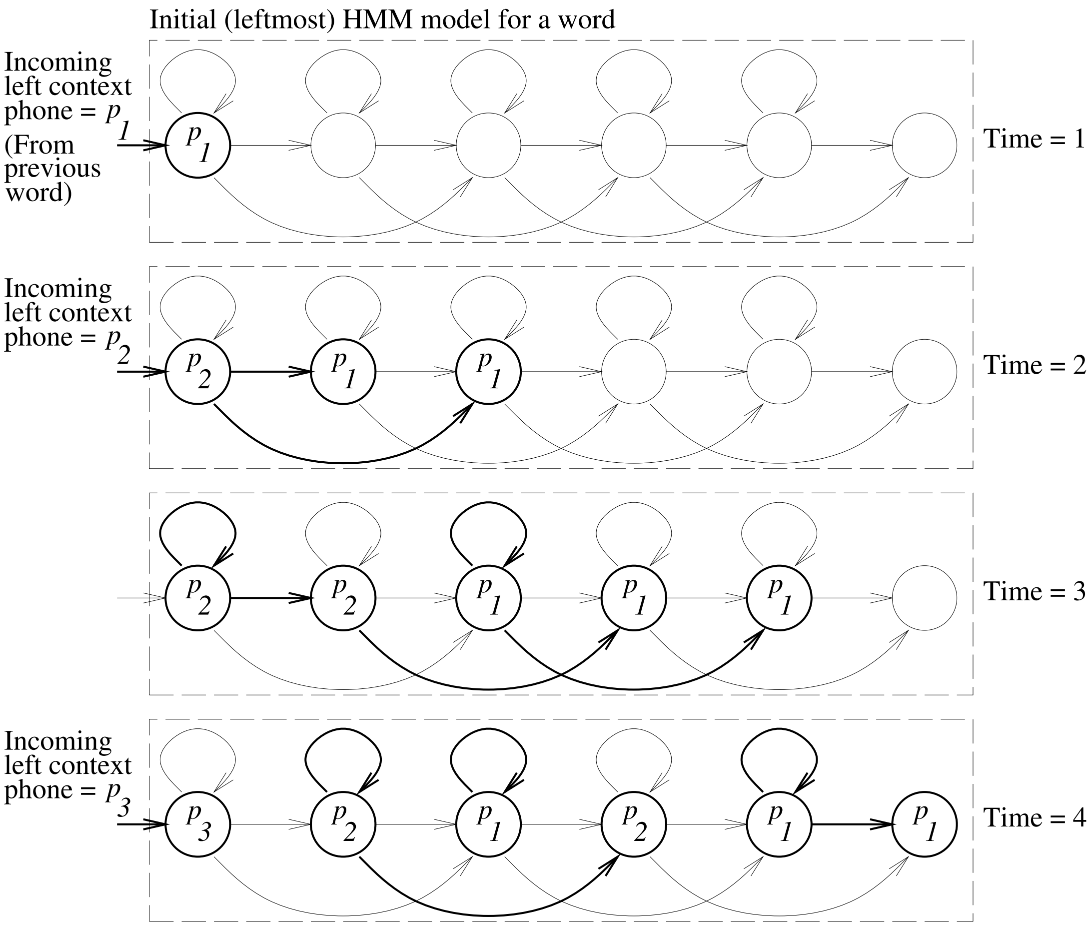


Figure 3.4: Word Initial Triphone HMM Modelling in Sphinx-II.

Single Phone Words

In the case of single-phone words, both their left and right phonetic contexts are derived dynamically from neighbouring words. Thus, they have to be handled by a combination of the above techniques. With reference to Figures 3.3 and 3.4, separate copies of the single phone have to be created for each right phonetic context, and each copy is modelled using the dynamic triphone mapping technique for handling its left phonetic context.

3.2.4 The Forward Search

The decoding algorithm is, in principle, straightforward. The problem is to find the most probable sequence of words that accounts for the observed speech. This is tackled as follows.

The abstract Viterbi decoding algorithm and the beam search heuristic, and its application to speech decoding have been explained in Section 2.3.1. In Sphinx-II, there are two distinguished words, `<s>` and `</s>` , depicting the beginning and ending silence in any utterance. The input speech is expected to begin at the initial state of `<s>` and end in the final state of `</s>` .

We can now described the forward Viterbi beam search implementation in Sphinx-II. It is explained with the help of fragments of pseudo-code. It is necessary to understand the forward pass at this level in order to follow the subsequent discussion on performance analysis and the breakdown of computation among different modules.

Search Outline

Before we go into the details of the search algorithm, we introduce some terminology. A state j of an HMM model m in the flat lexical search space has the following attributes:

- A path score at time t, $P_j^m(t)$, that indicates the probability corresponding to the best state sequence leading from the initial state of `<s>` at time 0 to this state at time t, while consuming the input speech until t.
- A *history* information at time t, $H_j^m(t)$, that allows us to trace back the best preceding word history leading to this state at t. (As we shall see later, this is a pointer to the word lattice entry containing the best predecessor word.)
- The senone output probability, $b_j^m(t)$, for this state at time t (see Section 2.1.2). If m belongs to the first position in a word, the senone identity for state j is determined dynamically from the inherited phonetic left context (Section 3.2.3).

At the beginning of the decoding of an utterance, the search process is initialized by setting the path probability of the start state of the distinguished word `<s>` to 1. All other states are initialized with a path score of 0. Also, an *active HMM list* that identifies the set of active HMMs in the current frame is initialized with this first HMM for `<s>` . From then on, the processing of each frame of speech, given the input feature vector for that frame, is outlined by the pseudo-code in Figure 3.5.

We consider some of the functions defined in Figure 3.5 in a little more detail below. Certain aspects, such as pruning out HMMs that fall below the beam threshold, have been omitted for the sake of simplicity.

VQ: *VQ* stands for *vector quantization*. In this function, the Gaussian densities that make up each feature codebook are evaluated at the input feature vectors. In other words, we compute the Mahalanobis distance of the input feature vector from the mean of each Gaussian density function. (This corresponds to evaluating $\mathcal{N}$ in

```
forward_frame (input feature vector for current frame)
{
    VQ (input feature);    /* Find top 4 densities closest to input feature */
    senone_evaluate ();    /* Find senone output probabilities using VQ results */

    hmm_evaluate ();       /* Within-HMM and cross-HMM transitions */
    word_transition ();    /* Cross-word transitions */
    /* HMM pruning using a beam omitted for simplicity */

    update active HMM list for next frame;
}

hmm_evaluate ()
{
    /* Within-HMM transitions */
    for (each active HMM h)
        for (each state s in h)
            update path probability of s using senone output probabilities;

    /* Within-word cross-HMM transitions and word-exits */
    for (each active HMM h with final state score within beam) {
        if (h is a final HMM for a word w) {
            create word lattice entry for w;  /* word exit */
        } else {
            let h' = next HMM in word after h;
            NULL transition (final-state(h) -> initial-state(h'));
            /* Remember right context fanout if h' is final HMM in word */
        }
    }
}

word_transition ()
{
    let {w} = set of words entered into word lattice in this frame;
    for (each word w' in vocabulary)
        Find the best transition ({w} -> w'), including LM probability;
}
```

Figure 3.5: One Frame of Forward Viterbi Beam Search in the Baseline System.

equation 2.1.) Only the top 4 densities are fully evaluated and used further, since the rest typically contribute very little to the senone output probability.

Since all senones share a single codebook per feature stream in the Sphinx-II semi-continuous model, the VQ step does not have to be repeated for each senone.

Senone Evaluation: (Function `senone_evaluate`.) In this function, we compute the output probability for each senone in the current frame as a weighted sum of the top 4 density values in the frame. There are 4 feature streams in Sphinx-II. The weighting is done independently on each stream and the final result is the product of the four weighted values. (See Sections 3.1.1 and 2.1.2.)

HMM Evaluation: (Function `hmm_evaluate`.) This step includes two cases:

- *Within-HMM transitions*: For each active HMM model m the path score of each state j in m is updated according to:

$$P_j^m(t) = \max_i(P_i^m(t-1) \cdot b_i^m(t) \cdot a_{ij}^m) \tag{3.1}$$

  where, t indicates the current frame, i ranges over all states of m, and a_{ij}^m is a static probability for the arc from i to j in m. (See also Section 2.1.2.) Furthermore, the history pointer $H_j^m(t)$, and the dynamic phonetic left context if applicable, are propagated to j from the state i that maximizes expression 3.1.

- *Within-word cross-HMM transitions and Word Exits*: A cross-HMM NULL transition within a word from HMM m_1 to m_2 causes the path score and history information to be propagated from the final state of m_1 to the start state of m_2 if it results in a better path score at the start state of m_2.

  Words whose final states have a score within the allowed threshold represent potential word recognitions in the current frame. There can be several such words in any given frame. All of them are entered in a *word lattice* along with the path score and history information from the final state. The right context fanout at the end of each word actually results in several entries for each word, one for each possible phonetic right context.

Cross-Word Transition: (Function `word_transition`.) In principle, this step attempts all possible cross-word transitions from the set of words exited to all words in the vocabulary, computing the language model probability in each case. If n words reached their final state in the current frame, and there are V words in the vocabulary, a total of nV transitions are possible. This is an enormous number. However, not all transitions have explicit trigram or even bigram probabilities in the language model.

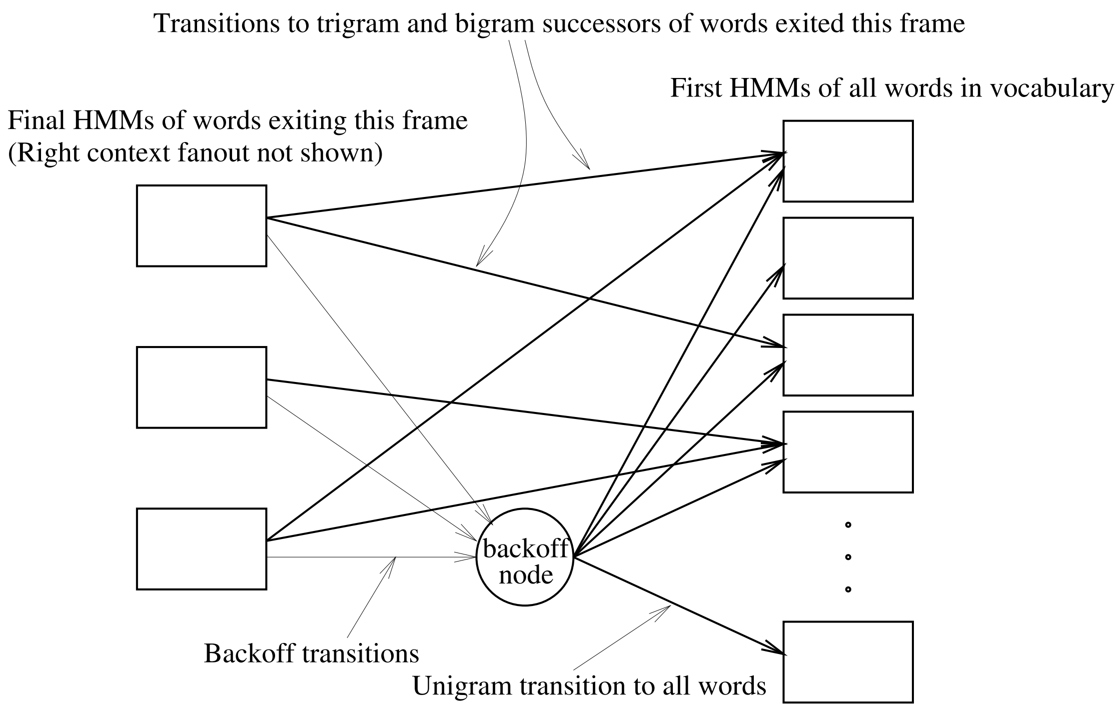


Figure 3.6: Word Transitions in Sphinx-II Baseline System.

Therefore, the computation is approximated by using trigram and bigram transitions that can actually be found in the grammar, and backing off to unigrams through a *backoff node* for the rest[3]. Thus, the total number of transitions evaluated is at most V plus the number of bigrams and trigrams for the n words exited, which is a typically a much smaller number than nV. The scheme is shown in Figure 3.6. For the sake of clarity, details involving cross-word triphone modelling have been omitted.

If a cross-word transition is successful, the history information of the start state of the destination word is updated to point to the word lattice entry corresponding to the "winning" word just exited, i.e., the best predecessor word at this time. The dynamic phonetic left context for the initial phone of the destination word is also set from the best predecessor word.

Result of Forward Beam Search

One result of the forward pass is the word lattice identifying each word recognized during the entire utterance. Each entry in the table identifies a word, its *segmentation* (i.e., start and end points in time), and the acoustic score for that word segmentation.

The second result of the forward Viterbi search is a single recognition hypothesis. It is the word sequence obtained by starting at the final state of `</s>` at the end of

[3]Trigram probabilities are applied using the approximation described in Section 3.2.2.

the utterance and backtracking to the beginning, by following the history pointers in the word lattice.

3.3 Backward and A* Search

As mentioned earlier, the A* or stack search is capable of exactly using more sophisticated language models than bigram grammars, thus offering higher recognition accuracy. It maintains a sorted stack of partial hypotheses which are expanded in a best-first manner, one word length at a time. There are two main issues with this algorithm:

- To prevent an exponential explosion in the search space, the stack decoding algorithm must expand each partial hypothesis only by a limited set of the most likely candidate words that may follow that partial hypothesis.
- The A* algorithm is not time synchronous. Specifically, each partial hypotheses in the sorted stack can account for a different initial segment of the input speech. This makes it hard to compare the path probabilities of the entries in the stack.

It has been shown in [42] that the second issue can be solved by attaching a *heuristic score* with every partial hypothesis H that accounts for the remaining portion of the speech not included in H. By "filling out" every partial hypothesis to the full utterance length in this way, the entries in the stack can be compared to one another, and expanded in a best-first manner. As long as the heuristic score attached to any partial hypothesis H is an upper bound on the score of the best possible complete recognition achievable from H, the A* algorithm is guaranteed to produce the correct results.

The backward pass in the Sphinx-II baseline system provides an approximation to the heuristic score needed by the A* algorithm. Since it is a time-synchronous Viterbi search, run in the backward direction from the end of the utterance, the path score at any state corresponds to the best state sequence between it and the utterance end. Hence it serves as the desired upper bound. It is an approximation since the path score uses bigram probabilities and not the exact grammar that the A* search uses.

The backward pass also produces a word lattice, similar to the forward Viterbi search. The A* search is constrained to search only the words in the two lattices, and is relatively fast.

The word lattice produced by the backward pass has another desirable property. We noted at the beginning of this chapter that for each word occurrence in the forward pass word lattice, several successive *end* times are identified along with their scores, whereas very often only the single most likely *begin* time is identified. The backward pass word lattice produces the complementary result: several beginning times are identified for a given word occurrence, while usually only the single most likely end time is available. The two lattices can be combined to obtain acoustic probabilities for a wider range of word beginning and ending times, which improves the recognition accuracy.

In the following subsections, we briefly describe the backward Viterbi pass and the A* algorithm used in the Sphinx-II baseline system.

3.3.1 Backward Viterbi Search

The backward Viterbi search is essentially identical to the forward search, except that it is completely reversed in time. The main differences are listed below:

- The input speech is processed in reverse.
- It is constrained to search only the words in the word lattice from the forward pass. Specifically, at any time t, cross-word transitions are restricted to words that exited at t in the forward pass, as determined by the latter's word lattice.
- All HMM transitions, as well as cross-HMM and cross-word NULL transitions are reversed with respect to the forward pass.
- Cross word triphone modelling is performed using *left*-context fanout and dynamic triphone mapping for *right* contexts.
- Only the bigram probabilities are used. Therefore, the Viterbi path score from any point in the utterance up to the end is only an approximation to the upper bounds desired by the A* search.

The result of the backward Viterbi search is also a word lattice like that from the forward pass. It is rooted at `</s>` that ends in the final frame of the utterance, and growing backward in time. The backward pass identifies several beginning times for a word, but typically only one ending time. Acoustic scores for each word segmentation are available in the backward pass word lattice.

3.3.2 A* Search

The A* search algorithm is described in [42]. It works by maintaining an ordered stack or list of partial hypotheses, sorted in descending order of likelihood. Hypotheses are word sequences and may be of different lengths, accounting for different lengths of input speed. Figure 3.7 outlines the basic stack decoding algorithm for finding N-best hypotheses.

```
initialize stack with <s>;
while (N > 0) {
    pop best hypothesis H off top of stack;
    if H is a complete hypothesis {
        output H, and decrement N;
    } else {
        find candidate list of successor words to H from backward pass lattice;

        for (each word W in above candidate list) {
            extend H by appending W to it, giving new partial hypothesis H';
            evaluate new score for H' using forward and backward lattices;
            insert H' into the stack in accordance with its new score;
        }
    }
}
```

Figure 3.7: Outline of A* Algorithm in Baseline System

The specific details relevant to the Sphinx-II implementation are covered in [2]. Most of the additional details pertain to two steps: identifying candidate word extensions for a partial hypothesis H, and computing the score for each newly created partial hypothesis H'. Candidate words are located by looking for lattice entries that begin where the partial hypothesis ends. The score for the new hypothesis H' is computed by factoring in the acoustic score for the new word W (obtained from the forward and backward pass word lattices), a new heuristic score to the end of the utterance from the end point of H', and the language model probability for W, given the preceding history, i.e., H.

The hypotheses produced by the A* algorithm are not truly in descending order of likelihood since the heuristic score attached to each partial hypothesis is only an approximation to the ideal. However, by producing a sufficiently large number of N-best hypotheses, one can be reasonably sure that the best hypothesis is included in the list. In our performance measurements described below, the value of N is 150. The best output from that list is chosen as the decoding for the utterance. There is no other post processing performed on the N-best list.

3.4 Baseline Sphinx-II System Performance

The performance of the baseline Sphinx-II recognition system was measured on several large-vocabulary, speaker-independent, continuous speech data sets of read speech[4] from the *Wall Street Journal* and other North American business news domain. These

[4] As opposed to spontaneous speech.

data sets have been extensively used by several sites in the past few years, including the speech group at Carnegie Mellon University. But the principal goal of these experiments has been improving the recognition accuracy. The work reported in this thesis is focussed on obtaining other performance measures for the same data sets, namely execution time and memory requirements. We first describe the experimentation methodology in the following section, followed by other sections containing a detailed performance analysis.

3.4.1 Experimentation Methodology

Parameters Measured and Measurement Techniques

The performance analysis in this section provides a detailed look at all aspects of computational efficiency, including a breakdown by the various algorithmic steps in each case. Two different vocabulary sizes—approximately 20,000 and 58,000 words, referred to as the *20K* and *58K* tasks, respectively—are considered for all experiments. The major parameters measured include the following:

- Recognition accuracy from the first Viterbi pass result and the final A* result. This is covered in detail in Section 3.4.2.
- Overall execution time and its breakdown among the major computational steps. We also provide frequency counts of the most common operations that account for most of the execution time. Section 3.4.3 deals with these measurements. Timing measurements are performed over entire test sets, averaged to per frame values, and presented in multiples of *real time*. For example, any computation that takes 23msec to execute per frame, on average, is said to run in 2.3 *times real time*, since a frame is 10msec long. This makes it convenient to estimate the execution cost and usability of individual techniques. Frequency counts are also normalized to per frame values.
- The breakdown of memory usage among various data structures. This is covered in Section 3.4.4.

Clearly, the execution times reported here are machine-dependent. Even with a single architecture, differences in implementations such as cache size, memory and bus speeds relative to CPU speed, etc. can affect the speed performance. Furthermore, for short events, the act of measuring them itself would perturb the results. It is important to keep these caveats in mind in interpreting the timing results. Having said that, we note that all experiments were carried out on one particular model of Digital Equipment Corporation's Alpha workstations. The Alpha architecture [61] includes a special *RPCC* instruction that allows an application to time very short events of as little as a few hundred machine cycles with negligible overhead. All timing measurements are normalized to an Alpha processor running at 175MHz.

It should also be emphasized that the main computational loops in the Sphinx-II system have been tuned carefully for optimum speed performance. The measurements reported in this work have been limited almost exclusively to such loops.

Test Sets and Experimental Conditions

The test sets used in the experiments have been taken from the various data sets involved in the 1993 and 1994 ARPA hub evaluations. All the test sets consist of clean speech recorded using high quality microphones. Specifically, they consist of the following:

- *Dev93*: The 1993 development set (commonly referred to as *si_dt_20*).
- *Dev94*: The 1994 development set (*h1_dt_94*).
- *Eval94*: The 1994 evaluation set (*h1_et_94*).

The test sets are evaluated individually on the *20K* and the *58K* tasks. This is important to demonstrate the variation in performance, especially recognition accuracy, with different test sets and vocabulary sizes. The individual performance results allow an opportunity for comparisons with experiments performed elsewhere that might be restricted to just some of the test sets. Table 3.1 summarizes the number of sentences and words in each test set.

	Dev93	*Dev94*	*Eval94*	*Total*
Sentences	503	310	316	1129
Words	8227	7387	8186	23800

Table 3.1: No. of Words and Sentences in Each Test Set

The knowledge bases used in each experiment are the following:

- Both the *20K* and the *58K* tasks use semi-continuous acoustic models of the kind discussed in Section 3.1.1. There are 10,000 senones or tied states in this system.
- The pronunciation lexicons in the *20K* tasks are identical to those used by CMU in the actual evaluations. The lexicon for the *58k* task is derived partly from the *20k* task and partly from the 100K-word dictionary exported by CMU.

- The *Dev93* language model for the *20K* task is the standard one used by all sites in 1993. It consists of about 3.5M bigrams and 3.2M trigrams. The *20K* grammar for *Dev94* and *Eval94* test sets is also the standard one used by all sites, and it consists of about 5.0M bigrams and 6.7M trigrams. The grammar for the *58K* task is derived from the approximately 230M words of language model training data that became available during the 1994 ARPA evaluations, and it consists of 6.1M bigrams and 18.0M trigrams. The same grammar is used with all test sets.

The following sections contain the detailed performance measurements conducted on the baseline Sphinx-II recognition system.

3.4.2 Recognition Accuracy

Recognition results from the first pass (Viterbi beam search) as well as the final A* pass are presented for both the *20K* and *58K* task. Table 3.2 lists the word error rates on each of the test sets, individually and overall[5][6]. Errors include substitutions, insertions and deletions.

	Dev93	*Dev94*	*Eval94*	*Mean*
20K(Vit.)	17.6	15.8	15.9	16.4
20K(A*)	16.5	15.2	15.3	15.7
58K(Vit.)	15.1	14.3	14.5	14.6
58K(A*)	13.8	13.8	13.8	13.8

Table 3.2: Percentage Word Error Rate of Baseline Sphinx-II System.

It is clear that the largest single factor that determines the word error rate is the test set itself. In fact, if the input speech were broken down by individual speakers, a much greater variation would be observed [45, 46]. Part of this might be attributable to different out-of-vocabulary (OOV) rates for the sets of sentences uttered by individual speakers. However, a detailed examination of a speaker-by-speaker OOV rate and error rate does not show any strong correlation between the two. The main conclusion is that word error rate comparisons between different systems must be restricted to the same test sets.

[5]The accuracy results reported in the actual evaluations are somewhat better than those shown here. The main reason is that the acoustic models used in the evaluations are more complex, consisting of separate codebooks for individual *phone classes*. We used a single codebook in our experiments instead, since the goal of our study is the cost of the search algorithm, which is about the same in both cases.

[6]Note that in all such tables, the overall mean is computed over all different sets put together. Hence, it is not necessarily just the mean of the means for the individual test sets.

3.4.3 Search Speed

In this section we present a summary of the computational load imposed by the Sphinx-II baseline search architecture. There are three main passes in the system: forward Viterbi beam search, backward Viterbi search, and A* search. The first presents the greatest load of all, and hence we also study the breakdown of that load among its main components: Gaussian density computation, senone score computation, HMM evaluation, and cross-word transitions. These are the four main functions in the forward pass that were introduced in Section 3.2.4. Although we present performance statistics for all components, the following functions in the forward Viterbi search will be the main focus of our discussion:

- HMM evaluation. We present statistics on both execution times as well as the number of HMMs evaluated per frame.
- Cross word transitions. Again, we focus on execution times and the number of cross-word transitions carried out per frame.

The execution time for each step is presented in terms of multiples of real time taken to process that step. As mentioned earlier, the machine platform for all experiments is the DEC Alpha workstation. All timing measurements are carried out using the *RPCC* instruction, so that the measurement overhead is minimized. It should again be emphasized that execution times are heavily influenced by the overall processor, bus, and memory architecture. For this reason, all experiments are carried out on a single machine model. The performance figures presented in this section are normalized to an Alpha processor running at 175MHz.

Overall Execution Times

Table 3.3 summarizes the execution times for both the *20K* and *58K* tasks. As we can see, the forward Viterbi search accounts for well over 90% of the computation. Its four major components can be grouped into two classes: *acoustic model evaluation* and *search*. The former includes the Gaussian density computation and senone output probability evaluation. The latter consists of searching the network of HMMs to find the best decoding—the main body of the Viterbi search algorithm.

Breakdown of Forward Viterbi Search Execution Times

Table 3.4 lists the breakdown of the forward pass execution times for the two vocabularies. The important conclusion is that the *absolute* speed of the search component is several tens of times slower than real time for both tasks. This shows that regardless of other optimizations we may undertake to improve execution speed, the

	Dev93	*Dev94*	*Eval94*	*Mean*	*%Total*
Forward	22.62	21.84	22.14	22.24	92.8%
Backward	0.56	0.57	0.59	0.57	2.4%
A*	1.28	1.02	1.12	1.15	4.8%

(*a*) *20K* Task.

	Dev93	*Dev94*	*Eval94*	*Mean*	*%Total*
Forward	46.71	40.25	39.20	42.43	95.8%
Backward	0.68	0.70	0.70	0.69	1.5%
A*	1.18	1.17	1.24	1.19	2.7%

(*b*) *58K* Task.

Table 3.3: Overall Execution Times of Baseline Sphinx-II System (xRealTime).

	Dev93	*Dev94*	*Eval94*	*Mean*	*%Forward*
VQ	0.16	0.16	0.16	0.16	0.7%
Senone Eval.	3.74	3.72	3.71	3.72	16.7%
HMM Eval.	10.24	9.26	9.41	9.88	44.4%
Word Trans.	8.29	8.49	8.66	8.47	38.1%

(*b*) *20K* Task.

	Dev93	*Dev94*	*Eval94*	*Mean*	*%Forward*
VQ	0.16	0.16	0.16	0.16	0.4%
Senone Eval.	3.81	3.82	3.85	3.82	9.0%
HMM Eval.	19.64	17.22	16.18	17.83	42.0%
Word Trans.	22.90	18.85	18.79	20.41	48.1%

(*b*) *58K* Task.

Table 3.4: Baseline Sphinx-II System Forward Viterbi Search Execution Times (xRealTime).

cost of search must be reduced significantly in order to make large vocabulary speech recognition practically useful.

Since we use semicontinuous acoustic models with just one codebook per feature stream, the cost of computing senone output probabilities is relatively low. In fact, over 80% of the total time is spent in searching the HMM space in the case of the *20K* task. This proportion grows to over 90% for the *58K* task. We can obtain significant speed improvement by concentrating almost solely on the cost of search.

We consider the search component—HMM evaluation and cross-word transitions—in more detail below.

HMMs Evaluated Per Frame in Forward Viterbi Search

Table 3.5 summarizes the average number of HMMs evaluated per frame in each of the test sets and the overall average, for both the *20K* and *58K* tasks. The table also shows the average number of word-initial HMMs computed per frame, in absolute terms and as a percentage of the total number.

	Dev93	*Dev94*	*Eval94*	*Mean*
Total	16957	15315	15411	15985
Word-initial	10849	9431	9299	9940
(%Total)	(63)	(61)	(60)	(62)

(*a*) *20K* Task.

	Dev93	*Dev94*	*Eval94*	*Mean*
Total	33291	26723	26700	29272
Word-initial	24576	18219	17831	20569
(%Total)	(74)	(68)	(67)	(70)

(*b*) *58K* Task.

Table 3.5: HMMs Evaluated Per Frame in Baseline Sphinx-II System.

The most striking aspect of the baseline Sphinx-II system is that in the *20K* task more than 60% of the total number of HMMs evaluated belong to the first position in a word. In the case of the *58K* task, this fraction grows to 70%. The reason for this concentration is simple. Since there are no pre-defined word boundaries in continuous speech, there are cross-word transitions to the beginning of every word in the vocabulary in almost every frame. These transitions keep most of the word-initial triphone models alive or active in every frame.

Cross-Word Transitions Per Frame in Forward Viterbi Search

A similar detailed examination of cross-word transitions shows a large number of unigram, bigram and trigram transitions performed in each frame. As explained in Section 3.2.4, there are three cases to be considered. If w is a word just recognized and w' its best Viterbi predecessor, we have the following sets of cross-word transitions:

1. Trigram followers of (w, w'),

2. Bigram followers of (w), and
3. Unigram transitions to every word in the vocabulary.

Of course, many of them are unsuccessful because their low *a priori* likelihood, as determined by the associated language model probabilities. In Table 3.6 we show the number of *successful* cross-word transitions per frame in the baseline Sphinx-II system.

	Dev93	*Dev94*	*Eval94*	*Mean*	*%Total*
Trigrams	996	662	708	807	6.2%
Bigrams	5845	5035	5079	5364	40.9%
Unigrams	7257	6848	6650	6944	52.9%

(a) *20K* Task.

	Dev93	*Dev94*	*Eval94*	*Mean*	*%Total*
Trigrams	1343	1276	1351	1326	5.1%
Bigrams	10332	7360	7367	8519	32.9%
Unigrams	20090	13792	13305	16087	62.0%

(b) *58K* Task.

Table 3.6: N-gram Transitions Per Frame in Baseline Sphinx-II System.

We conclude that both bigram and unigram transitions contribute significantly to the cost of cross-word transitions.

3.4.4 Memory Usage

It is somewhat hard to measure the *true* memory requirement of any system without delving into the operating system details. There are two measures of memory space: virtual memory image size, and the resident or *working set* size. The former is easy to measure, but the latter is not. We consider both aspects for each of the main data structures in the baseline system.

Acoustic Model

In our experiments with Sphinx-II using semi-continuous acoustic models, the senone mixture weights discussed in Sections 2.1.2 and 3.1.1 constitute the largest portion of the acoustic models. The 10,000 senones occupy 40MBytes of memory, broken down as follows. Each senone contains 256 32-bit weights or coefficients corresponding to

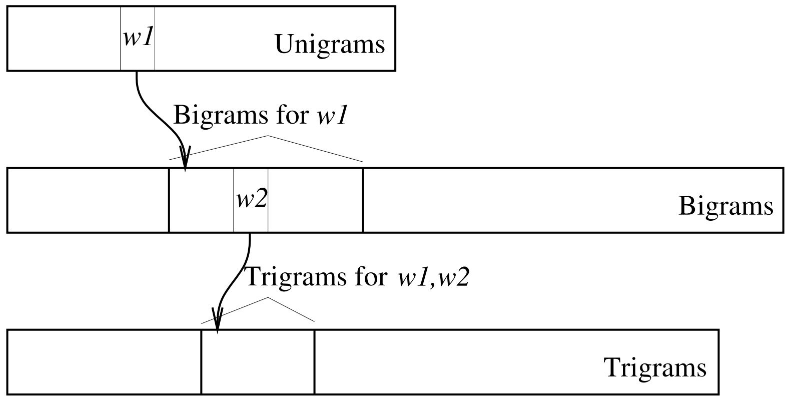


Figure 3.8: Language Model Structure in Baseline Sphinx-II System.

the 256 codewords in a codebook. There are four codebooks for the four feature streams (Section 3.1.1).

The memory resident size of the senone mixture weights is not significantly less than their total size. If all Gaussian densities were fully evaluated in every frame and weighted by the senone coefficients, the entire 40MB data structure would be touched and resident in memory every frame. In practice, only the top 4 densities out of 256 are used in a given frame. Nevertheless, the identity of the top 4 densities varies rapidly from frame to frame. Hence, most of the senone mixture weights are accessed within a short period of time. That is, there isn't very much locality of access to this data structure to be exploited by the virtual memory system.

Language Model

The representation of the language model data structure has been quite well optimized. The sets of unigrams, bigrams, and trigrams are organized into a tree structure wherein each unigram points to the set of its bigram successors, and each bigram points to its trigram successors. Figure 3.8 illustrates this organization.

The memory requirement for unigrams in a large-vocabulary, word trigram grammar is negligible, compared to the higher-order n-grams. A bigram entry includes the following four components:

1. *Word-id.* A bigram is a two-word pair. Since all bigram followers of a single unigram are grouped under the unigram, it is only necessary to record the second word of the bigram in its data structure.

2. Bigram *probability*.

3. A *backoff weight* that is applied when a trigram successor of the bigram is not in the language model and we have to back off to another bigram.

4. Pointer to the *trigram successors* for the bigram.

Similarly, a trigram consists of 2 entries: a *word-id* and its trigram *probability*. By means of a set of indirect lookup tables, each of the components of a bigram or trigram entry is compressed into 2 bytes. In other words, a single bigram requires 8 bytes, and a single trigram 4 bytes.

Based on these figures, the two language models used in the *20K* task (see Section 3.4.1) occupy about 41MB and 67MB of memory, respectively. The *58K* task language model measures at about 121MB.

In this case, the difficulty faced by the virtual memory system in managing the working set is that the granularity of access is usually much smaller than the physical page size of modern workstations. Many words have just a few 10s to 100s of bigram successors. For example, the average number of bigrams per word in the case of the *58K* vocabulary is about 105. Whereas, the page size on a DEC Alpha is 8KB, 16KB, or more. Hence, much of the contents of a page of bigrams might be unused.

Search Data Structures

One of the search data structures is the network of active word HMMs in each frame. It is a dynamically varying quantity. The average number of active HMMs per frame, shown in Table 3.5, is a rough measure of this parameter. Its peak value, however, can be substantially higher. Since all of the active HMMs in a frame have to be evaluated, they are all resident in memory. Other prominent search data structures include the forward and backward pass word lattices, the sizes of which grows approximately proportionately with the utterance length.

All of these data structures are relatively small compared to the acoustic and language models, and we exclude them from further discussion.

Memory Usage Summary

In summary, the virtual memory requirement of the baseline system is well over 100MB for the *20K* tasks and around 200MB for the larger *58K* task, excluding auxiliary data structures used in the three passes. It is worth noting that the *20K* tasks are on the verge of thrashing on a system with 128MB of main memory, indicating that most of the virtual pages are indeed being touched frequently.

3.5 Baseline System Summary

The purpose behind this chapter has been to outline the basic algorithms comprising the baseline Sphinx-II speech recognition system, as well as to evaluate its performance on large-vocabulary, continuous-speech tasks. We have obtained a measure of its efficiency along three basic dimensions: recognition accuracy, speed, and memory requirements. The evaluations were carried out on several test sets of read speech from the *Wall Street Journal* and *North American Business News* domains. The tests are run with two different vocabulary sizes of 20K and 58K words.

The immediate conclusion from these measurements is that the baseline Sphinx-II system cannot be used in practical, large vocabulary speech applications. Its computational and memory requirements are an order of magnitude beyond the capabilities of commonly available workstations. While it is possible to improve the recognition speed by tightening the beam width (i.e., pruning the search more ruthlessly) and using less sophisticated acoustic models to reduce memory requirements, such measures cannot overcome the inherent algorithmic complexities of the system. Moreover, they also result in an unacceptable increase in the recognition error rate.

We summarize our conclusions from this chapter:

- The main search of the full vocabulary, i.e. the forward Viterbi search, is computationally the most expensive. It accounts for over 90% of the total time. Postprocessing the word lattice is relatively inexpensive.
- The *search* component of the forward Viterbi search, even on modern high-end workstations, is several tens of times slower than real time on large vocabulary, continuous speech tasks.
- About half of the search cost is attributable to HMM evaluation. Moreover, the active HMMs to be evaluated during search are concentrated near the beginning of words. Specifically, over 60-70% of the active HMMs are word-initial models. This is not a new result. It has also been pointed out before, for example in [39, 43], although it has not been quantified as systematically.
- The other half of the search cost is attributable to the evaluation of cross-word transitions, along with the need to perform several thousands of language model accesses in each frame. Both bigram and unigram transitions contribute significantly to this cost.
- The memory requirements of large vocabulary speech recognition systems are dominated by the two main databases: acoustic models and language models. For large tasks they can run between 100-200MB.

It is clear that in order for the state-of-the-art speech recognition systems to become useful, we must address all of the above issues.

Chapter 4

Search Speed Optimization

4.1 Motivation

Most of the research effort on large vocabulary continuous speech recognition has primarily been in improving recognition accuracy, exemplified by the baseline Sphinx-II system. We have seen in the previous chapter that the Sphinx-II system is several tens of times too slow and requires 100-200MB of memory for large vocabulary tasks. In order to be practically useful, speech recognition systems have to be efficient in their usage of computational resources as well.

There clearly are several real-time recognition systems around in the ARPA speech research community [23, 60, 55, 24]. However, the published literature is relatively bare regarding them. Their performance has never been formally evaluated with respect to the research systems or with respect to one another, in the way that the accuracy of research systems has been. One goal of this thesis is to demonstrate that it is possible to achieve near real-time performance on large-vocabulary, continuous speech recognition tasks without compromising the recognition accuracy offered by research systems. This is a way of lending validity to the ongoing research on improving accuracy.

We can also look at the current focus of speech research from the following angle. Speech recognition systems consist of two main components:

- *Modelling* structure, consisting of acoustic and language models.
- *Algorithmic* or *search* structure. For example, the forward pass Viterbi beam search algorithm described in the previous chapter.

Clearly, both components contribute to the various dimensions of efficiency of the system—accuracy, speed, memory usage. But much of speech research has been focussed on the modelling aspect, specifically towards improving recognition accuracy.

This chapter concentrates on improving the algorithmic structure of search, while preserving the gains made in the modelling arena.

Another reason for concentrating on the search problem is the following. The complexity of speech tasks is constantly growing, outpacing the growth in the power of commonly available workstations. Since the late 1980s, the complexity of tasks undertaken by speech researchers has grown from the 1000-word *Resource Management* (*RM*) task [51] to essentially unlimited vocabulary tasks such as transcription of radio news broadcast in 1995 [48]. The *RM* task ran about an order of magnitude slower than real time on processors of that day. The unlimited vocabulary tasks run about two orders of magnitude slower than real time on modern workstations. At least part of the increase in the computational load is the increase in the search space. It seems reasonable to expect that task complexity will continue to grow in the future.

In this chapter, we discuss several algorithms and heuristics for improving the efficiency of a recognition system. We use the Sphinx-II research system described in Chapter 3 as a baseline for comparison. Since the focus of this work is in improving search algorithms, we use the same acoustic and language models as in the baseline system. As mentioned above, there are two variables in speech recognition systems, modelling and search algorithms. By keeping one of them constant, we also ensure that comparisons of the performance of proposed search algorithms with the baseline system are truly meaningful.

Though the work reported in this thesis has been carried out in the context of semi-continuous acoustic models, it is also relevant to systems that employ fully continuous models. At the time that this work was begun, the Sphinx-II semi-continuous acoustic models were the best available to us. Over the last two years fully continuous acoustic models [66, 5, 18] have become much more widely used in the speech community. They reduce the word error rate of recognition systems by a relative amount of about 20-30% compared to semi-continuous acoustic models[1] [46, 47]. The use of fully continuous models does not eliminate the search problem. On the other hand, the cost of computing output probabilities for each state in each frame becomes much more significant that in the semi-continuous system. Hence, improving the speed of search alone is not sufficient. We demonstrate that the proposed search algorithms using semi-continuous models generate compact word lattices with low lattice error rate. Such lattices can be postprocessed efficiently using more complex acoustic models for higher accuracy.

The outline of this chapter is as follows:

- In Section 4.2 we discuss *lexical tree* Viterbi search and all its design ramifica-

[1]The contribution of acoustic modelling in different systems to recognition accuracy is hard to estimate since some systems use not one but several sets of acoustic models, particularly for speaker adaptation [64]. The overall accuracy resulting from the use of continuous HMM models plus several cycles of mean and variance adaptation was about 50% better than semi-continuous HMM modelling with little or no adaptation.

tions. We show how the lexical tree can be used to not only take advantage of the reduction in the number of active HMMs, but also to significantly reduce the number of language model operations during cross-word transitions. Tree-structured lexicons are increasingly being used in all speech recognition systems to take advantage of the sharing of HMMs across words, but this is the first instance of reducing the language model operations significantly. The section includes detailed performance measurements and comparisons to the baseline Sphinx-II system.

- In Section 4.3 we present an efficient word lattice search to find a globally optimum path through the lattice using a trigram grammar. Even though the lexical tree search is about 20% worse in recognition accuracy relative to the baseline system, most of the loss is recovered with this step. The global word lattice search improves the recognition accuracy by considering alternative paths that are discarded during the lexical tree Viterbi search.

- In Section 4.4 we show that by *rescoring* the word lattice output of the tree search using the conventional search algorithm of the baseline system, we essentially regain the recognition accuracy of the baseline system. Though our rescoring experiments are restricted to semi-continuous acoustic models, clearly more sophisticated models can be used as well.

- In Section 4.5 we propose a *phonetic fast match* heuristic that can be easily integrated into the lexical tree search algorithm to reduce the search, with virtually no loss of accuracy. The heuristic uses senone output probabilities in each frame to predict a set of active basephones near that frame. All others are considered inactive and pruned from search.

- There is a good deal of inherent parallelism at various levels in a speech recognition system. As commercial processor architectures and operating systems become capable of supporting multithreaded applications, it becomes possible to take advantage of the applications' inherent concurrency. In Section 4.6 we explore the issues involved in exploiting them.

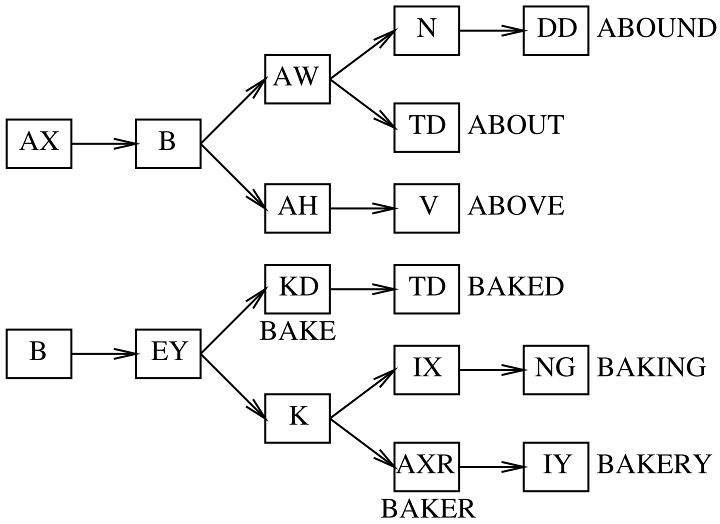
This chapter concludes with Section 4.7 that summarizes the performance of all of the techniques presented in this chapter.

4.2 Lexical Tree Search

The single largest source of computational efficiency in performing search is in organizing the HMMs to be searched as a phonetic tree, instead of the flat structure described in Section 3.2.1. It is referred to as a *tree-structured lexicon* or *lexical tree*

```
ABOUND     AX B AW N DD
ABOUT      AX B AW TD
ABOVE      AX B AH V
BAKE       B EY KD
BAKED      B EY KD TD
BAKER      B EY K AXR
BAKERY     B EY K AXR IY
BAKING     B EY K IX NG
```

(a) Example Pronunciation Lexicon.



(b) Basephone Lexical Tree.

Figure 4.1: Basephone Lexical Tree Example.

structure. In such an organization, if the pronunciations of two or more words contain the same n initial phonemes, they share a single sequence of n HMM models representing that initial portion of their pronunciation. Tree-structured lexicons have often been used in the past, especially in *fast-match* algorithms as a precursor step to a stack-decoding algorithm. More recently, tree search has come into widespread use in the main decoding process [43, 39]. Figure 4.1 shows a simple base-phone lexical tree[2] for a small example lexicon.

The lexical tree offers a potential solution to the two main sources of computational cost in the baseline system:

- By introducing a high degree of sharing at the root nodes, it reduces the number

[2]Strictly speaking, the so-called lexical tree is actually a *collection* of trees or a *forest*, rather than a single tree. Nevertheless, we will continue to use the term *lexical tree* to signify the entire collection.

of word initial HMMs that need to be evaluated in each frame. As we saw in Section 3.4.3, word-initial HMMs are the most frequently evaluated HMMs in the baseline system.

- The tree structure also greatly reduces the number of cross-word transitions, which is again a dominant part of search in the baseline system (see Section 3.4.3, Table 3.6).

Another advantage of the tree organization is that both the number of active HMMs and the number of cross-word transitions grow much more slowly with increasing vocabulary size than in the case of a flat lexical structure. On a per active HMM basis, however, there is more work involved in the lexical tree search, since each active HMM makes NULL transitions to several successor nodes, rather than just a single node as in the baseline system.

The main impediment to the full realization of the above advantages of tree search is the incorporation of a language model into the search. In the flat lexical structure, each cross-word transition from word w_i to w_j is accompanied by a language model probability $P(w_j|w_i)$, assuming a bigram grammar. The difficulty with the tree structure is that individual words are not identifiable at the roots of the tree. The root nodes represent the beginning of several different words (and hence multiple grammar states), which are related phonetically, but not grammatically. This can lead to conflicts between different cross-word transitions that end up at the same root node.

Most of the current solutions rely on creating additional word HMM networks to handle such conflicts. The prominent ones have been reviewed in Section 2.4. The obvious drawback associated with these solutions is an increase in the number of operations that the lexical tree structure is supposed to solve in the first place.

In this work we present a coherent solution that avoids the replication by postponing the computation of language model probability for a word until the end of the word is reached. We show that this strategy improves the computational efficiency of search as it takes full advantage of the tree structure to dramatically reduce not only the number of HMMs searched but also the number of cross-word transitions and language model probabilities to be evaluated.

We first present the structure of the lexical tree in Section 4.2.1, followed by the main issue of treating language model probabilities across word transitions in Section 4.2.2. The overall tree search algorithm is discussed in Section 4.2.3. Section 4.2.4 contains a detailed performance analysis of this algorithm, and we finally conclude with a summary in Section 4.2.5.

4.2.1 Lexical Tree Construction

In Figure 4.1 we saw the construction of a lexical tree of base phone nodes. However, we wish to use triphone acoustic models rather than simple base phone models for high recognition accuracy. Hence, the lexical tree has to be built out of triphone nodes rather than basephone nodes. This basically requires a trivial change to Figure 4.1, except at the roots and leaf positions of the tree (corresponding to word beginnings and endings), which have to deal with cross-word triphone models.

The issues that arise in dealing with cross-word triphone modelling have been discussed in Section 3.2.3. The Sphinx-II tree-structured decoder also uses similar strategies[3]. To summarize:

- In a time-synchronous search, the phonetic right contexts are unknown since they belong to words that would occur in the future. Therefore, all phonetic possibilities have to be considered. This leads to a *right context fanout* at the leaves of the lexical tree.

- The phonetic left context at the roots of the lexical tree is determined dynamically at run time, and there may be multiple contexts active at any time. However, a fanout at the roots, similar to that at the leaves, is undesirable since the former are active much more often. Therefore, cross-word triphones at the root nodes are modelled using the *dynamic triphone mapping* technique described in Section 3.2.3. It multiplexes the states of a single root HMM between triphones resulting from different phonetic left contexts.

Figure 4.2 depicts the earlier example shown in Figure 4.1, but this time as a triphone lexical tree. The notation $b(l,r)$ in this figure refers to a triphone with basephone b, left context phone l, and right context phone r. A question-mark indicates an unknown context that is instantiated dynamically at run time.

The degree of sharing in a triphone lexical tree is not as much as in the basephone version, but it is still substantial at or near the root nodes. Table 4.1 lists the number of tree nodes at various levels, the corresponding number of nodes in the flattened lexicon (i.e., if there were no sharing), and the ratio of the former to the latter as a percentage. Leaf nodes were not considered in these statistics since they have to be modelled with a large right context fanout. The degree of sharing is very high at the root nodes, but falls off sharply after about 3 levels into the tree.

In our implementation, the entire lexical tree, except for the leaf nodes with their right context fanout, is instantiated as a data structure in memory. If the leaf nodes were also allocated statically, their right context fanout would increase the total

[3] Unlike the baseline system, however, single-phone words have been modelled more simply, by modelling different left contexts but ignoring the right context.

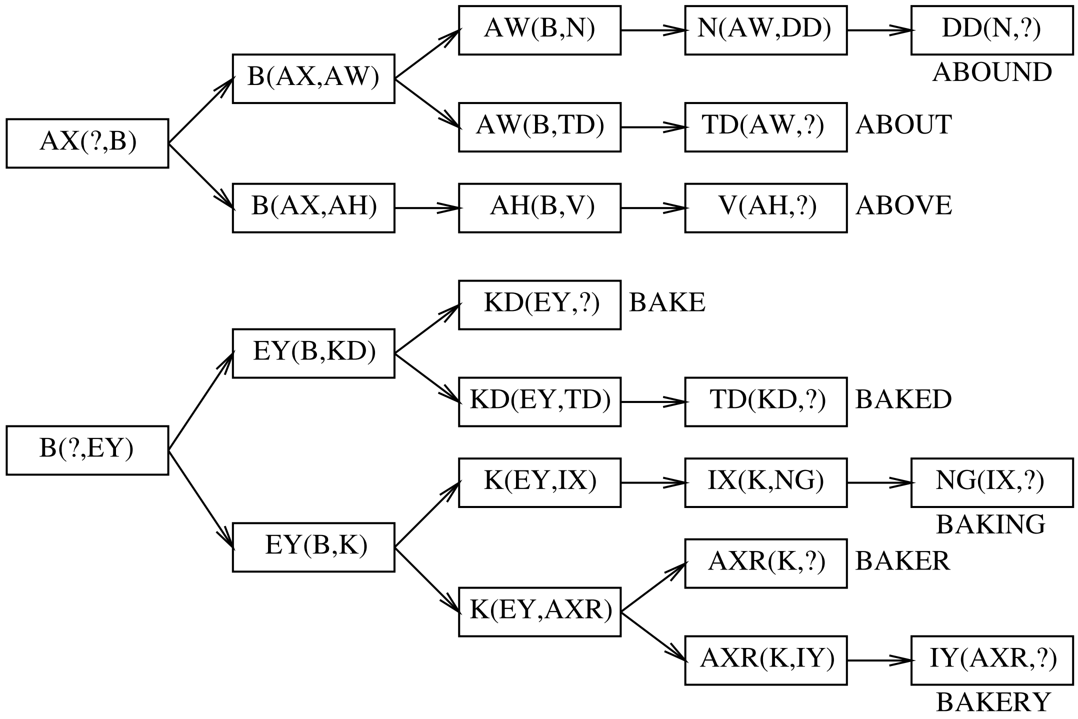


Figure 4.2: Triphone Lexical Tree Example.

	20K(Dev93)			20K(Dev94/Eval94)			58K		
Level	Tree	Flat	Ratio	Tree	Flat	Ratio	Tree	Flat	Ratio
1	656	21527	3.0%	690	21725	3.2%	851	61657	1.4%
2	3531	21247	16.6%	3669	21430	17.1%	5782	61007	9.5%
3	8047	19523	41.2%	8339	19694	42.3%	18670	57219	32.6%
4	9455	16658	56.8%	9667	16715	57.8%	26382	49390	53.4%
5	8362	12880	64.9%	8493	12858	66.1%	24833	38254	64.9%
6	6359	9088	70.0%	6388	8976	71.2%	18918	26642	71.0%
7	4429	5910	74.9%	4441	5817	76.3%	13113	17284	75.9%
8	2784	3531	78.8%	2777	3448	80.5%	8129	10255	79.3%

Table 4.1: No. of Nodes at Each Level in Tree and Flat Lexicons.

number of triphone models enormously. Therefore, leaf nodes are only allocated on demand; i.e., when these HMMs become active during the search.

4.2.2 Incorporating Language Model Probabilities

The application of language model probabilities at word boundaries presents an interesting dilemma. Traditionally, the language model probability for a transition from word w_i to w_j is computed and accumulated during the transition into the initial state of w_j. See, for example, the baseline system description in Section 3.2.4. As a result, the initial score for the new word w_j is "primed" with the appropriate expectation for that word in the context of the preceding history. This approach fits neatly into the Markov model and the Viterbi search algorithm, and has two main advantages:

- By using the language model probability upon word entry, the search process is biased in favour of the grammatically more likely words, and against the less likely ones. This bias serves to prune away the less likely words, reducing the dynamic search space.
- Frequently occurring short words or function words, such as *a*, *the*, *an*, *of*, etc., which are generally poorly articulated, are given an initial boost by the language model at the appropriate moments[4]. Thus, even though their poor articulation might result in a poor acoustic match subsequently, the initial priming by the language model often allows them to survive the beam search without getting pruned.

The disadvantage of computing language model probabilities upon word entry is, of course, the computational cost of evaluating a very large number of them in each frame. This was seen in the previous chapter in Section 3.4.3, Table 3.4, making the execution of cross-word transitions one of the most costly steps in the search process.

One would like to retain the advantages stated above, without incurring the associated cost, if possible. The immediate problem with a tree-structured lexicon is that one does not have distinct, identifiable initial states for each word in the lexicon. The tree structure implies that the root nodes are shared among several words, related phonetically, but quite unrelated grammatically. Hence it is not possible to determine a meaningful language model probability upon transitioning to a root node.

The Language Modelling Problem

Let us see the problem in detail by referring to Figure 4.3(a) and the original algorithm for cross-word transitions in the baseline system in Figure 3.6. Figure 4.3(a) depicts cross-words NULL transitions attempted from the final states of two words p_1 and p_2 to the initial states of words w_1 and w_2 at time t. Let us represent the path scores

[4]The correct thing to do is, of course, to improve the *acoustic* modelling of such events rather than relying on the language model to overcome the former's shortcomings. However, every bit helps!

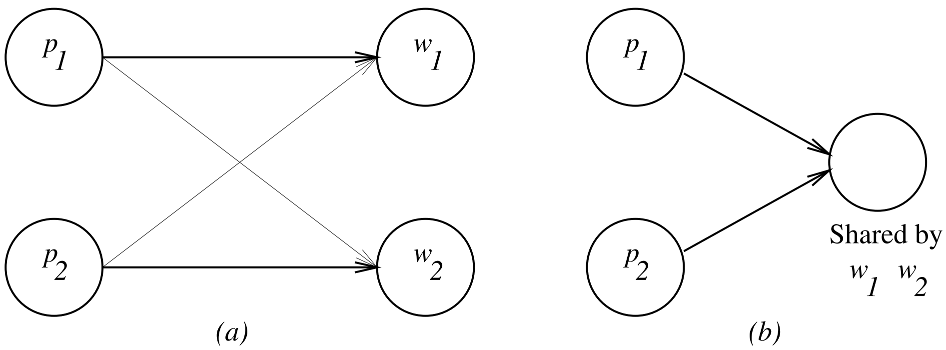


Figure 4.3: Cross-Word Transitions With Flat and Tree Lexicons.

at the end of any p_i at time t by $P_{p_i}(t)$, and the bigram probability for the transition from p_i to w_j by $P_{LM}(w_j|p_i)$. In the flat-lexical search of the baseline system, the path score entering w_j from p_i at time t is:

$$P_{p_i}(t) \cdot P_{LM}(w_j|p_i) \tag{4.1}$$

The Viterbi algorithm chooses the better of the two arcs entering each word w_1 and w_2, and their history information is updated accordingly. In Figure 4.3(a), the "winning" transitions are shown by bold arrows.

In particular, the presence of separate word-HMM models for w_1 and w_2 allows them to capture their distinct best histories. However, if w_1 and w_2 share the same root node in the tree lexicon, as shown in Figure 4.3(b), it is no longer possible to faithfully retain the distinctions provided by the grammar. It should be emphasized that the bigram grammar *is* the source of the problem. If only unigram probabilities are used, $P_{LM}(w_j|p_i)$ is independent of p_i and the best incoming transition is the same for all words w_j.)

Suggested Solutions to Language Modelling Problem

Several attempts have been made to resolve this problem, as mentioned in Section 2.4. One solution to this problem has been to augment the lexical tree with a separate flat bigram section. The latter is used for all bigram transitions and the lexical tree only for unigram transitions [39]. The scheme is shown in Figure 4.4. Bigram transitions, from the leaves of either the lexical tree or flat structure, always enter the flat structure, preserving the grammar state distinctions required, for example, in Figure 4.3(a). Unigram transitions enter the roots of the lexical tree. This solution has two consequences for the speed performance:

- The addition of the flat lexicon increases the dynamic number of HMM models to be searched.

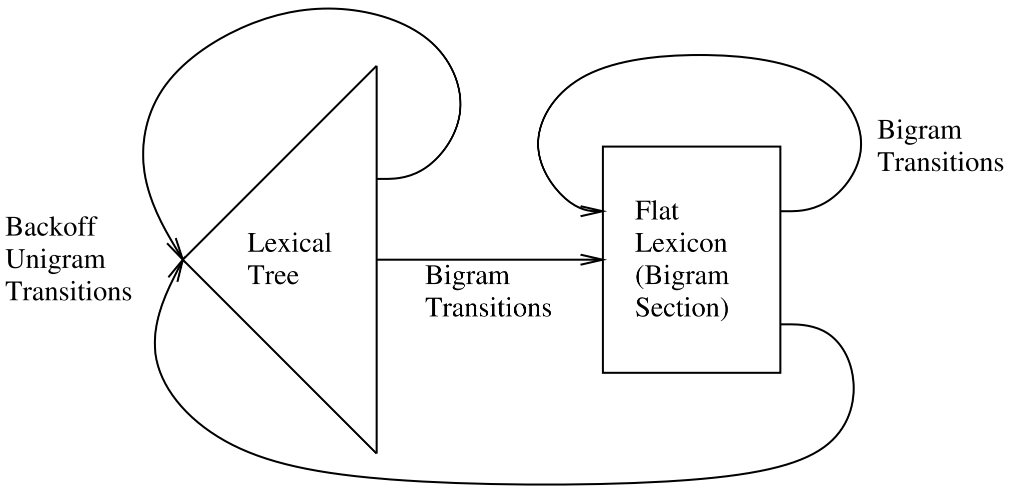


Figure 4.4: Auxiliary Flat Lexical Structure for Bigram Transitions.

- The number of unigram transitions is reduced significantly because of the tree structure. However, the number of bigram transitions is similar to that of the baseline system (Section 3.4.3, Table 3.6), which still constitutes a significant computational load.

Alternative solutions are proposed in [40, 3] that construct separate secondary trees for the bigram section, instead of the flat lexical structure of Figure 4.4. Both of them report results on 10,000 word vocabulary experiments. In the former, the number of tree copies that are active range between 20 and 30, causing an increase in the number of active states by about a factor of 2. The latter have reported near real-time performance on the 10,000 word task with a language model of perplexity 27. It is not clear how the performance extends to tasks with a larger vocabulary and grammars.

Computing Language Model Probability Upon Word Exit

The difficulties can be overcome simply by deferring the computation of the language model probability for a word until we reach a node in the lexical tree that uniquely represents that word; i.e., it is not shared with any other word. If there are multiple words with identical pronunciations (homophones), they can still be forced to become distinct at the leaf nodes. Therefore, we can defer the computation of language model score for a word until it exits its leaf node in the lexical tree. The advantage of this approach is that the total number of such computations per frame is very small. The number of words that survive the beam search all the way to their final state, on average, is about two orders of magnitude smaller than the vocabulary size.

Let us see how this scheme works with respect to the example in Figure 4.3(b).

Extract H_{temp} from the final state of w_j;
From H_{temp} find p_{temp}, the initially chosen predecessor word;
Obtain t, the end time of p_{temp};
Find all word lattice entries p_i that ended at t;
    (In this example, they are p_1 and p_2.)
Find: $f = \max_i((P_{p_i}(t)/P_{p_{temp}}(t)) \cdot P_{LM}(w_j|p_i))$;

Figure 4.5: Path Score Adjustment Factor f for Word w_j Upon Its Exit.

Since the transitions to the root node shared by w_1 and w_2 no longer include their language model probabilities, the incoming score into the root at time t is simply:

$$\max_i(P_{p_i}(t))$$

The root node also inherits a history information that points to the word lattice entry for the best predecessor word, as described in Section 3.2.4. However, it is a *temporary* value since it does not include a language model probability for the transition. Let's call this history H_{temp}. It is propagated as usual by the Viterbi algorithm and eventually reaches the final states of w_1 or w_2 (assuming they are not pruned by the beam search). By our earlier assumption, the final states of the leaf nodes belong to distinct words and are not shared. Therefore, their language model probabilities can now be included, and the path scores and history information h_{temp} updated if necessary. Figure 4.5 summarizes this *path score adjustment* at the end of word w_j. The value f computed in the figure is the adjustment factor applied to the path score at the end of w_j. The word lattice entry that maximizes f becomes the adjusted history at the end of w_j, replacing H_{temp}. (In our discussions we have neglected to deal with cross-word triphone models. However, it is straightforward to accommodate it into the expression $P_{p_i}(t)$.)

There are some disadvantages that stem from deferring the accumulation of language model probabilities until word exit:

- The initial "priming" or guidance provided by the language model is absent. Since all words are grammatically equal until their final state is reached, the search pruning provided by the language model is lost and the number of HMM models to be searched increases.

- Short function words which occur frequently but are poorly articulated, are more likely to be pruned by the beam search before their final state is ever reached. If their language model probabilities had been included upon word entry, on the other hand, they would have received an initial boost allowing them to survive the beam pruning.

These concerns are addressed below.

Computing Language Model Probability When Entering Leaf Nodes

One obvious solution to the lack of guidance from a language model is to adopt an intermediate solution between computing the language model probability at the very beginning upon word entry and at the very end upon word exit. Since the degree of sharing in the lexical tree drops rapidly beyond 3 or 4 phone positions, one might as well flatten the lexical structure completely beyond that depth. For example, the lexical tree of Figure 4.2 is essentially flat beginning at a depth of 4.

Since individual words are identifiable beyond a level of 3 or 4 from the roots, language model scores can be computed during the NULL transitions at these points in a similar fashion to that described above. Thus, the computational savings afforded by the tree structure are retained near the root where it matters most, and the guidance and search pruning provided by the language model is available when the tree structure ceases to be as effective.

However, the above solution still doesn't address the problem of poorly articulated function words which are typically just 1-3 phones long. Secondly, the shorter we make the depth of the actual tree structure and the earlier we compute language model probabilities, the more HMMs are actively being searched at that point, increasing the cost of the path score adjustments.

For these reasons, in the final implementation of our tree search algorithm, the language model probability for a word is computed upon *entering the final leaf node*[5] for that word, rather than when exiting it. The algorithm is basically identical to that shown in Figure 4.5, except that the path score adjustment is performed when entering the final phone of w_j, rather than exiting w_j. Furthermore, it does not apply to single-phone words, which have to be treated essentially as in the baseline Sphinx-II system, outside the lexical tree structure. But this is not a major issue since the number of single-phone words in the vocabulary is only about 10.

For a short function word, this organization has the effect of accumulating the language model probability early into the word, reducing the chances of its having been pruned because of poor acoustic match. In particular, in the case of single phone words, the language model probability is computed and accumulated upon word entry. This compromise partly retains the guidance provided by the language model for poorly articulated short function words, while preserving computational efficiency for the vast majority of the remaining words.

[5]Note that there are really several leaf nodes for any given word, caused by the right context fanout for cross-word triphone modelling. However, we shall continue to speak loosely of a *leaf node* in the singular, with the right context fanout being implicitly understood.

Optimization of Path Score Adjustment

We can further reduce the cost of path score adjustment operations shown in Figure 4.5 using the following optimization.

We observe that in the Viterbi algorithm, if a triphone model survives the beam pruning through to its exit state at a certain frame, it is very likely to continue to survive in the next several frames. This is particularly true of triphones near word-ends. Thus, if we make a transition into a given leaf node of the lexical tree at time t_e, we are likely to make that transition again at $t_e + 1$. This is because speech corresponding to a phone lingers for several frames.

We note that the path score adjustment for a transition into a leaf node at t_e+1 is identical to the adjustment at t_e, provided the temporary history information H_{temp} is identical in both cases (see previous discussion and Figure 4.5). This is obvious because the final expression in Figure 4.5 for the adjustment factor f:

$$\max_i((P_{p_i}(t)/P_{p_{temp}}(t)) \cdot P_{LM}(w_j|p_i))$$

is independent of t_e; all the variables involved depend only on H_{temp}.

Therefore, we can eliminate many path score adjustment operations as follows. When we enter the leaf node of a word w_j with a new temporary history information H_{temp} for the first time, we compute the complete path score adjustment factor and cache the result. If we transition to the leaf node again in subsequent frames with the same history, we simply re-use the cached result. Some rough measurements indicate that this optimization eliminates approximately 50% of the adjustment operations in our benchmarks.

4.2.3 Outline of Tree Search Algorithm

The lexical tree search is implemented as a time-synchronous, Viterbi beam search algorithm[6]. It is similar to the baseline Sphinx-II decoder in many ways:

- It uses the same signal processing front end and semi-continuous phonetic HMM models as the baseline system, described in Section 3.1.1. The HMM topology is the 5-state Bakis model shown in Figure 3.2.
- The pronunciation lexicon is also identical to that used in the baseline system (Section 3.1.2).
- It uses backed off word trigram language models.

[6] The tree-search decoder is known within CMU as FBS8.

- Cross-word modelling at word ends is accomplished by right context fanout, and at word beginnings by multiplexing a single HMM with *dynamic triphone mapping* (Section 3.2.3).
- The *vector quantization* step is identical to the baseline system (Section 3.2.4). In particular, only the top 4 densities in each feature codebook are fully evaluated and used.
- Senone output probability evaluation is similar to the baseline system, except that we have the option of evaluating only the active senones in a given frame. These are identified by scanning the active HMMs in that frame. It is not worthwhile in the baseline system because of the overhead of scanning the much larger number of active HMMs.
- The result of the Viterbi search is a single recognition hypothesis, as well as a word lattice that contains all the words recognized during the decoding, their time segmentations, and corresponding acoustic scores. The word lattice typically contains several alternative end times for each word occurrence, but usually only a single beginning time.

As with the baseline system, the decoding of each utterance is begun with the path probability at the start state of the distinguished word `<s>` set to 1, and 0 everywhere else. An *active HMM list* that identifies the set of active HMMs in the current frame is initialized with this first HMM of `<s>` . From then on, the processing of each frame of speech, given the input feature vector for that frame, is outlined by the pseudo-code in Figure 4.6. Some of the details, such as pruning out HMMs that fall below the beam threshold, have been omitted for the sake of clarity.

The Viterbi recognition result is obtained by backtracking through the word lattice, starting from the lattice entry for the distinguished end symbol `</s>` in the final frame and following the history pointers all the way to the beginning.

4.2.4 Performance of Lexical Tree Search

The lexical tree search implementation was evaluated on the same large vocabulary, continuous speech test sets of read speech from the *Wall Street Journal* and *North American Business News* domains as the baseline Sphinx-II system. To recapitulate, they include the clean speech development test sets from the Dec.'93 and Dec.'94 DARPA speech evaluations, as well as the evaluation test set of the latter.

The experiments are carried out on two different vocabulary sizes of *20K* and *58K* words. The main parameters measured include the following:

- Overall execution time and its breakdown among major components, as well as

```
tree_forward_frame (input feature vector for current frame) {
    VQ (input feature);     /* Find top 4 densities closest to input feature */
    senone_evaluate ();     /* Find senone output probabilities using VQ results */

    hmm_evaluate ();        /* Within-HMM/cross-HMM (except leaf transitions) */
    leaf_transition ();     /* Transitions to tree leaf nodes, with LM adjustment */
    word_transition ();     /* Cross-word transitions */
    /* HMM pruning using a beam omitted for simplicity */

    update active HMM list for next frame;
}
hmm_evaluate () {
    for (each active HMM h)
        for (each state s in h)
            update path probability of s using senone output probabilities;

    /* Cross-HMM and tree-exit NULL transitions */
    L = NULL;  /* List of leaf transitions in this frame */
    for (each active HMM h with final state score within beam) {
        if (h is leaf node or represents a single phone word) {
            create word lattice entry for word represented by h;  /* word exit */
        } else {
            for (each descendant node h' of h) {
                if (h' is NOT leaf node)
                    NULL transition (final-state(h) -> start-state(h'));
                else
                    add transition h->h' to L;
            }
        }
    }
}
leaf_transition () {
    for (each transition t in L) {
        let transition t be from HMM h to h', and w the word represented by h';
        compute path score adjustment entering h', INCLUDING LM probability of w;
        update start state of h' with new score and history info, if necessary;
    }
}
word_transition () {
    let {w} = set of words entered into word lattice in this frame;
    for (each single phone word w')
        compute best transition ({w} -> w'), INCLUDING LM probabilities;
    for (each root node r in lexical tree)
        compute best transition ({w} -> r), EXCLUDING LM probabilities;
}
```

Figure 4.6: One Frame of Forward Viterbi Beam Search in Tree Search Algorithm.

frequency counts of the most common operations that account for most of the execution time.

- Word error rates for each test set and vocabulary size.

The experimentation methodology is also similar to that reported for the baseline system. In particular, the execution times are measured on DEC's Alpha workstations using the *RPCC* instruction to avoid measurement overheads. See Section 3.4.1 for complete details.

Recognition Speed

Table 4.2[7] lists the execution times of the lexical tree search on the *20K* and *58K* tasks, and also shows the overall speedup obtained over the baseline Sphinx-II recognition system (see Table 3.3 for comparison). Clearly, tree search decoding is several

Task	*Dev93*	*Dev94*	*Eval94*	*Mean*
20K	4.68	4.66	4.75	4.70
58K	8.93	8.36	8.68	8.69

(*a*) Absolute Speeds (xRealTime).

Task	*Dev93*	*Dev94*	*Eval94*	*Mean*
20K	4.8	4.7	4.7	4.7
58K	5.2	4.8	4.5	4.9

(*b*) Speedup Over Forward Viterbi Pass of Baseline System.

Table 4.2: Execution Times for Lexical Tree Viterbi Search.

times faster than the baseline system on the given *20K* and *58K* tasks. As mentioned at the beginning of this chapter, however, there are two main aspects to the decoding procedure, *acoustic model evaluation*, and *searching* the HMM space, of which the latter has been our main emphasis. Therefore, it is instructive to consider the execution speeds of individual components of the lexical tree search implementation.

Table 4.3 shows the breakdown of the overall execution time of the lexical search algorithm into five major components corresponding to the main functions listed in Figure 4.6. It is also instructive to examine the number of HMMs and language model operations evaluated per frame. These are contained in Tables 4.4 and 4.5, respectively.

[7]Note that in all these tables, the mean value is computed over all test sets put together. Hence, it is not necessarily just the mean of the means for the individual test sets.

	Dev93	*Dev94*	*Eval94*	*Mean*	*%Total*
VQ	0.15	0.15	0.16	0.15	3.2%
Senone Eval.	1.73	1.70	1.73	1.72	36.6%
HMM Eval.	2.02	1.98	2.02	2.01	42.8%
Leaf Trans.	0.56	0.58	0.60	0.58	12.3%
Word Trans.	0.20	0.23	0.24	0.22	4.7%

(*a*) *20K* System.

	Dev93	*Dev94*	*Eval94*	*Mean*	*%Total*
VQ	0.16	0.17	0.17	0.16	1.8%
Senone Eval.	2.39	2.27	2.32	2.33	26.8%
HMM Eval.	3.78	3.57	3.71	3.70	42.6%
Leaf Trans.	2.08	1.81	1.91	1.94	22.3%
Word Trans.	0.51	0.56	0.57	0.54	6.2%

(*b*) *58K* System.

Table 4.3: Breakdown of Tree Viterbi Search Execution Times (xRealTime).

	Dev93	*Dev94*	*Eval94*	*Mean*	*%Baseline*
Total	4298	4181	4281	4259	26.6%
Word-initial	551	556	557	554	5.6%
(%Total)	(12.8)	(13.3)	(13.0)	(13.0)	

(*a*) *20K* System.

	Dev93	*Dev94*	*Eval94*	*Mean*	*%Baseline*
Total	7561	7122	7358	7369	25.2%
Word-initial	711	680	683	693	3.4%
%Total	(9.4)	(9.5)	(9.3)	(9.4)	

(*b*) *58K* System.

Table 4.4: No. of HMMs Evaluated Per Frame in Lexical Tree Search.

	Dev93	*Dev94*	*Eval94*	*Mean*	*%Baseline*
20K	663	591	609	625	4.8%
58K	1702	1493	1558	1595	6.2%

Table 4.5: No. of Language Model Operations/Frame in Lexical Tree Search.

In summary, the lexical tree search has reduced the total number of HMMs evaluated per frame to about a quarter of that in the baseline system. More dramatically, the number of language model operations have been reduced to about 5-6%, mainly because of the decision to defer the inclusion of language model probabilities until the leaves of the lexical tree.

Accuracy

Table 4.6(a) shows the word error rates resulting from the lexical tree search on the different test sets individually and overall. Table 4.6(b) provides a comparison with the baseline system results from both the forward Viterbi search and the final A* algorithm.

	Dev93	*Dev94*	*Eval94*	*Mean*
20K	21.2	18.9	18.0	19.4
58K	19.2	17.5	17.1	18.0

(a) Absolute Word Error Rates(%).

Baseline	*Dev93*	*Dev94*	*Eval94*	*Mean*
20K(Vit.)	20.6	19.6	13.4	17.8
20K(A*)	28.8	24.3	18.0	23.7
58K(Vit.)	27.9	22.4	18.3	23.1
58K(A*)	39.4	26.6	24.1	30.3

(b) %Degradation w.r.t. Baseline System Error Rates.

Table 4.6: Word Error Rates for Lexical Tree Viterbi Search.

The relative increase in recognition errors, compared to the baseline system, is unquestionably significant. The appropriate baseline for comparison is the output of the first Viterbi search pass, for obvious reasons, but even then the tree search is about 20% worse in relative terms. However, it can be argued that the nearly five fold speedup afforded by the lexical tree search is well worth the increase in error rate. In practical terms, the absolute word error rate translates, very roughly, into 1 error about every 5 words, as opposed to the baseline case of 1 error about every 6 words.

More importantly, we shall see in the subsequent sections of this chapter that the loss in accuracy can be completely recovered by efficiently postprocessing the word lattice output of the tree Viterbi search.

We attribute the increase in word error rate to occasionally poorer word segmentations produced by the tree search, compared to the baseline system. One problem is that Viterbi search is a greedy algorithm that follows a local maximum. In each frame, the root nodes of the lexical tree receive several incoming, cross-word transitions (from the final states of the leaves of the tree), of which the best is chosen by the Viterbi algorithm. The same is true in the case of the baseline system with a flat lexical structure. However, in the latter, each cross-word transition is augmented with a grammar probability so that the *effective* fan-in is reduced. This is not possible with the tree structure, with the result that the Viterbi pruning behaviour at tree roots is modified.

4.2.5 Lexical Tree Search Summary

Clearly, the results from the lexical tree search algorithm are mixed. On the one hand, there is a nearly 5-fold overall increase in recognition speed, but it is accompanied by an approximately 20% increase in word error rate, relative to the baseline system. We shall see in subsequent sections that we can, not surprisingly, recover from the loss in accuracy by postprocessing the word lattice output of the tree search algorithm. Some of the other conclusions to be drawn in this section are the following:

- While the overall speedup is slightly under 5, the search speed alone, excluding senone output probability computation, is over 6 times faster than the baseline case (comparing Tables 4.3 and 3.4). This is an important result since our focus in this section has been improving the speed of searching the HMM space.

- It should be pointed out that the reduction in search speed is irrelevant if the cost of computing state output probabilities is overwhelming. Thus, it is appropriate to rely on a detailed tree search if we are using semi-continuous or even discrete acoustic models, but it is less relevant for fully continuous ones.

- Using semi-continuous acoustic models, we obtain a word lattice that is extremely compact. The total number of words in the lattice is, on average, several hundreds to a few thousand for an average sentence of 10sec duration (1000 frames). Furthermore, the *lattice error rate*—the fraction of correct words not found in the lattice around the expected time—is extremely small. It is about 2%, excluding out-of-vocabulary words. This is substantially the same as the lattice error rate of the baseline Sphinx-II system, and similar to the results reported in [65]. The compact nature of the word lattice, combined with its low error rate, makes it an ideal input for further postprocessing using more detailed acoustic models and search algorithms.

The lexical tree described in this section can be contrasted to those described in [40, 3, 39, 43] in their treatment of the language model. By deferring the application of language model probabilities to the leaves of the tree, we gain a significant reduction in computation.

4.3 Global Best Path Search

In Section 4.2.4 we saw that although the lexical tree search algorithm improves the execution efficiency of large vocabulary continuous speech recognition, there is also a significant degradation in the recognition accuracy of about 20% relative to the baseline Sphinx-II system using the same acoustic, lexical and grammar models. We also observed that much of this degradation could be attributed to the following factors:

- Greedy nature of the Viterbi algorithm in following a locally optimum path that is globally suboptimal; and more so than in the case of the baseline system.
- Poorer word segmentations along the best Viterbi decoding.

However, the lexical tree search algorithm produces not only the single best Viterbi decoding, but also a word lattice containing other candidate words recognized. An examination of the word lattices from both the lexical tree Viterbi search and the flat-lexical Viterbi search in the baseline system reveals that the correct words are predominantly present in both lattices at the expected times. Therefore, it is possible to extract a more accurate recognition result from the word lattice.

In this section we present a simple and efficient algorithm to search the word lattice produced by the lexical tree search for a *globally optimum* decoding. This is accomplished by casting the word lattice as a directed acyclic graph (DAG) such that the problem is reduced to that of finding the least-cost path from end to end. Therefore, any of the well-known and efficient shortest-path graph search algorithms can be used. We show that the algorithm brings the recognition accuracy significantly closer to that of the baseline system, at an almost negligible computational cost.

4.3.1 Best Path Search Algorithm

Global Best Path Search Using Bigram Grammar

The word lattice output from the lexical tree Viterbi search algorithm contains instances of all candidate words that were recognized during the search. In particular, there may be several candidates at any point in time. Each unique *word instance* is identified by two quantities: the word itself, and a start time for that instance of the word. Figure 4.7 shows an example of such a word lattice, where each word instance is identified by one of the line segments representing a word starting at a specific time frame. Note that the Viterbi search algorithm produces a range of end times for each word instance, as observed earlier at the beginning of Chapter 3 and in Section 4.2.3.

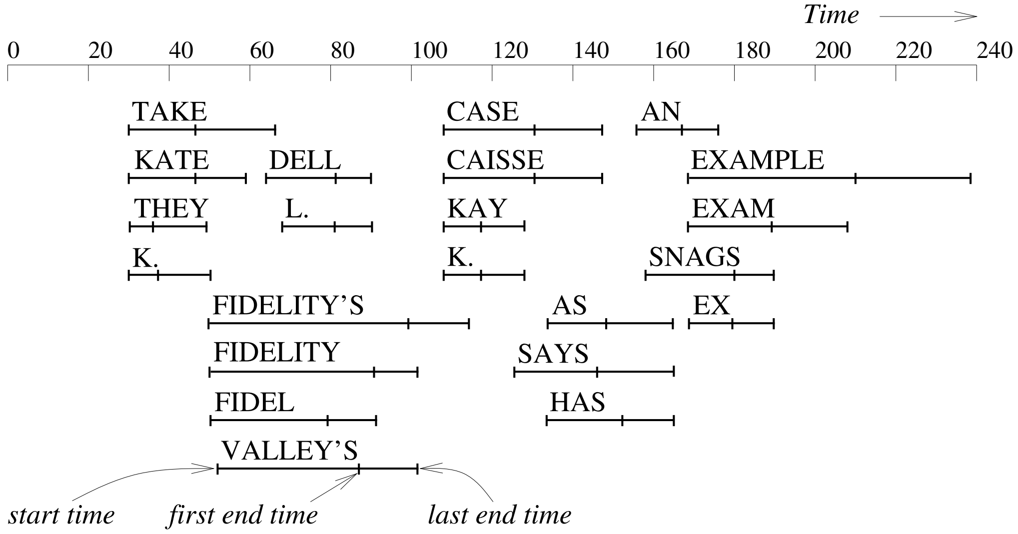


Figure 4.7: Word Lattice for Utterance: *Take Fidelity's case as an example.*

Thus, the information contained in the word lattice can be converted into as a DAG as follows: Each word instance represented by a pair, (w, t), is a DAG node, where w is a word-id and t the start time corresponding to this instance of w. There can be a range of end-times for this word instance or DAG node, as just mentioned. We create an edge from a node (w_i, t_i) to node (w_j, t_j) iff $t_j - 1$ is one of the end times of (w_i, t_i); i.e., there is a word lattice entry for w_i at $t_j - 1$ and so it is possible for (w_j, t_j) to follow (w_i, t_i) in time. Such a DAG representation of the example in Figure 4.7 is shown in Figure 4.8. It is easy to see that the graph is indeed a DAG:

- The edges are directed.
- The DAG cannot contain any cycles since edges always proceed in the direction of increasing start time.

The DAG is rooted at (`<s>` ,0), since the Viterbi search algorithm is initialized to start recognition from the beginning silence `<s>` at time 0. We can also identify a *final node* in the DAG which must be an instance of the end silence word `</s>` that has an end time of T, where T is the end time for the entire utterance[8].

We can now associate a cost with each edge in the DAG. Consider an edge from a node (w_i, t_i) to (w_j, t_j). The cost for this edge is the product of two components: an acoustic score or probability and a grammar probability[9]. The acoustic score is obtained as follows. The edge represents a time segmentation of w_i from frame t_i

[8]We can be sure that there will only be one such instance of `</s>` , since there can only be one entry for `</s>` ending at T in the word lattice.

[9]Actually, the cost is computed from the reciprocal of the probabilities, since an increase in the latter implies a reduction in the former.

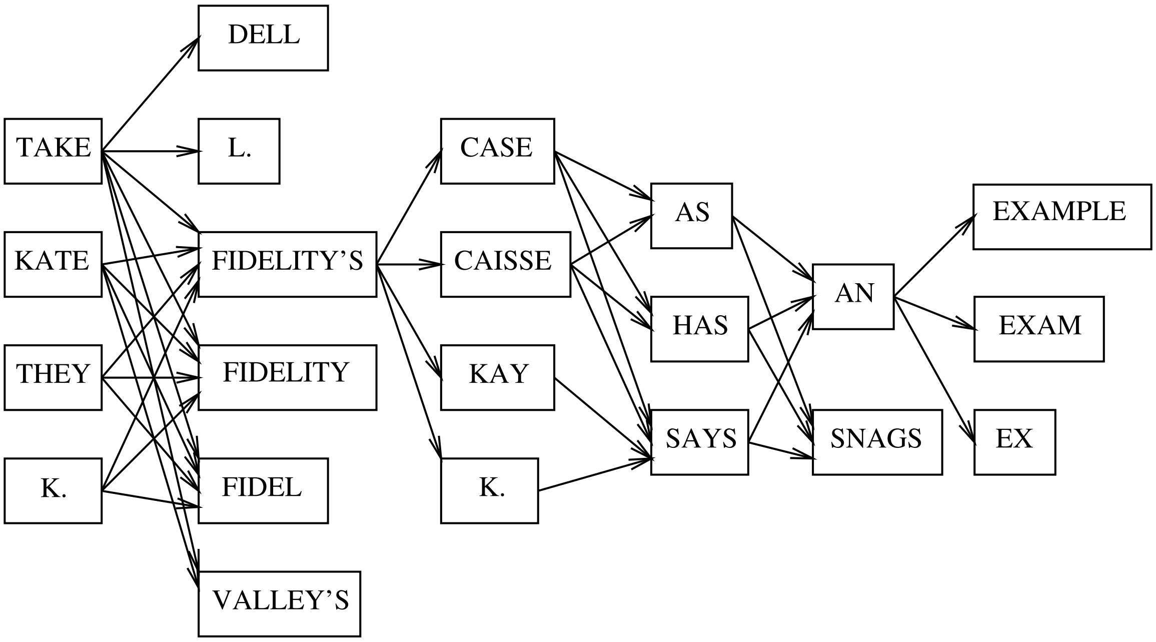


Figure 4.8: Word Lattice Example Represented as a DAG.

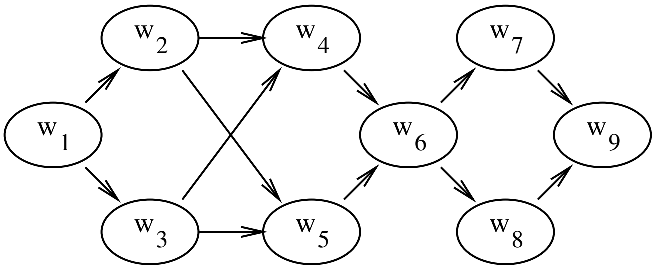
and ending at $t_j - 1$, as discussed above. Since the word lattice produced by the Viterbi search contains all the word-ending scores of interest, we can easily compute the acoustic score for this segmentation of w_i. In fact, the word lattice contains path scores for all possible phonetic right contexts of w_i, and we can choose exactly the right one depending on the first base phone of w_j.

As for the language model probability component for the edge, let us first consider the case of a simple bigram grammar. The grammar probability component for the edge under consideration is just $P(w_j|w_i)$. In particular, it is independent of the path taken through the DAG to arrive at (w_i, t_i).

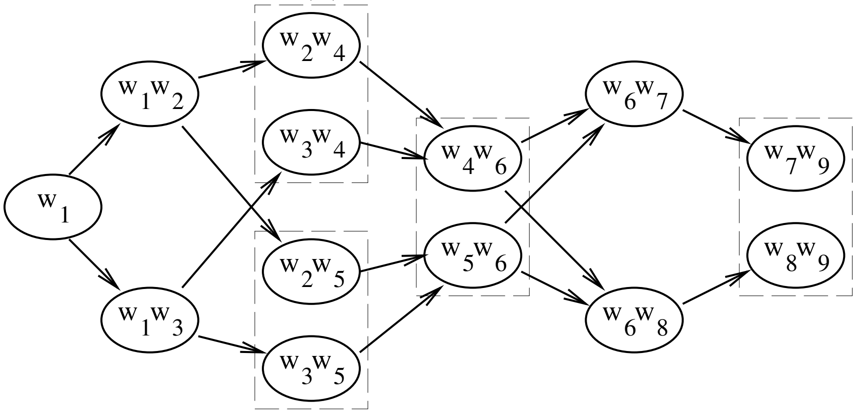
We have now obtained a cost for each edge of the graph. The cost of any path through the DAG from the root node to the final node is just the product of the costs of the individual edges making up the path. The path that has the least cost is the globally optimum one, given the input word lattice, acoustic models and (bigram) grammar. The word sequence making up this path has to be the globally optimum one. Given the above formulation of the problem, any of the textbook algorithms for finding the least-cost path can be applied [17]. Given a graph with N nodes and E edges, the least-cost path can be found in time proportional to $N + E$.

Global Best Path Search Using Trigram Grammar

The above formulation of edge costs is no longer valid if we use a trigram grammar since the grammar probability of an edge is not solely dependent on the edge. Con-



(a) Original DAG.



(b) Modified DAG for Trigram Grammar.

Figure 4.9: Word Lattice DAG Example Using a Trigram Grammar.

sider an edge from node (w_i, t_i) to (w_j, t_j) again. The trigram probability for the transition depends also on the predecessor of w_i. Since there can be more than one such predecessor in the DAG, the grammar probability for the edge under consideration is not uniquely determined.

The difficulty is easily resolved by the usual method of replicating a node for each distinct predecessor, i.e., creating distinct grammar states in the DAG. We illustrate this process with an example in Figure 4.9. (The start time information at each node has been omitted from the figure since it is superfluous and only clutters up the picture, as long as it is understood that each node has a specific start time associated with it. We shall also omit the time component in labelling nodes below, under the assumption that nodes can be identified uniquely even after this omission.) Modification to the DAG is straightforward:

1. If a node (w) has n distinct predecessors $(w_i), i = 1, 2, \ldots, n$ in the original DAG, it is replicated n times in the new DAG, labelled $(w_i w), i = 1, 2, \ldots, n$ respectively; i.e., the first component of the label identifies a predecessor word. Instances of such replication where $n > 1$ are marked by dashed rectangles in Figure 4.9(b).

2. If there was an edge from (w_i) to (w_j) in the original DAG, the new DAG has an edge from every replicated copy of (w_i) to $(w_i w_j)$. Note that by the replication and labelling process, if the new DAG has an edge from $(w_i w_j)$ to $(w_k w_l)$, then $w_j = w_k$.

3. The acoustic score component of the cost of an edge from node $(w_i w_j)$ to $(w_j w_k)$ in the new DAG is the same as that of edge (w_j) to (w_k) in the original DAG.

4. The language model component of the cost of an edge from node $(w_i w_j)$ to $(w_j w_k)$ in the new DAG is the trigram probability: $P(w_k|w_i w_j)$.

In particular, it should be noted that the language model probability component of the cost of an edge in the new DAG is no longer dependent on other edges in the DAG.

It should be easy to convince ourselves that the new DAG is equivalent to the original one. For any path from the root node to the final node in the original DAG, there is a corresponding path in the new DAG, and *vice versa*. Thus, we have again reduced the task to the canonical shortest path graph problem and the standard methods can be applied.

It is, in principle, possible to extend this approach to arbitrary language models, but it quickly becomes cumbersome and expensive with higher order n-gram grammars. With n-gram grammars, the size of the graph grows exponentially with n, and this is one of the drawbacks of this approach. Nevertheless, it is still valuable since bigram and trigram grammars are the most popular and easily constructed for large vocabulary speech recognition.

Suboptimality of Viterbi Search

At this point we consider the question of why we should expect the global best path algorithm to find a path (i.e., word sequence) that is any better than that found by the Viterbi search. One reason has to do with the approximation in applying the trigram grammar during Viterbi search as explained in Section 3.2.2. The same approximation is also used in the lexical tree Viterbi search. The suboptimal nature of this approximation can be understood with the help of Figure 4.10.

Let us say that at some point in the Viterbi tree search, there were two possible transitions into the root node for word w_4 from the final states of w_2 and w_3. And let us say that the Viterbi algorithm deemed the path $w_1 w_2 w_4$ (including acoustic and grammar probabilities) to be more likely and discarded transition $w_3 w_4$, shown by the dashed arrow in the figure. It could turn out, later when we reach word w_5, that perhaps $w_3 w_4 w_5$ is a more likely trigram than $w_2 w_4 w_5$, and in the *global* picture transition $w_3 w_4$ is a better choice. However, given that the Viterbi algorithm has already discarded transition $w_3 w_4$, the global optimum is lost. The shortest

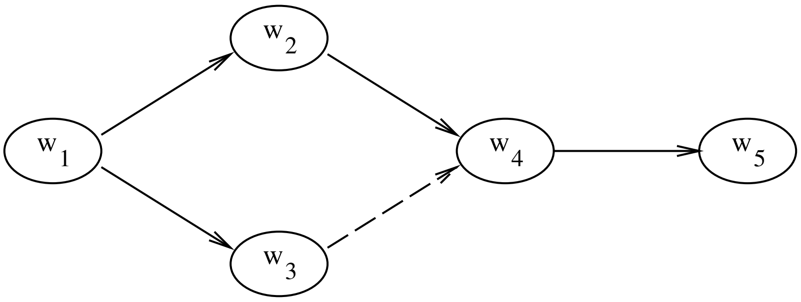


Figure 4.10: Suboptimal Usage of Trigrams in Sphinx-II Viterbi Search.

path algorithm described here considers all such alternatives discarded by the Viterbi algorithm and finds the globally optimum path.

The second reason for the improvement in accuracy follows from the above. We noted in Section 4.2.4 that the Viterbi algorithm, owing to its greedy nature, produces suboptimal word segmentations along the best Viterbi path. However, the word lattice often also contains other word segmentations that have been discarded along the best Viterbi path. The global DAG search uncovers such alternatives in finding a global optimum, as described above.

4.3.2 Performance

We now summarize the improvement in recognition accuracy obtained by applying the global best path search algorithm to the word lattice produced by the lexical tree search. We also examine the computational overhead incurred because of this additional step.

Accuracy

Table 4.7 shows the word error rate figures on our benchmark test sets resulting from applying the best path DAG search algorithm to the word lattice output of the lexical tree Viterbi search.

As we can see from Table 4.7(b), there is a significant improvement of over 10% in accuracy relative to the tree search. Correspondingly, Table 4.7(c) shows that compared to the first pass of the baseline Sphinx-II system, the word error rate is now less than 10% worse, in relative terms. In practical terms, this is almost insignificant given their absolute word error rates of 15%. We surmise that this difference in recognition accuracy is partly attributable to incorrect word segmentations for which no alternatives were available in the word lattice, and partly to pruning errors during the tree search.

	Dev93	*Dev94*	*Eval94*	*Mean*
20K	18.5	16.9	16.5	17.3
58K	16.5	15.4	15.2	15.7

(*a*) Absolute Word Error Rates.

	Dev93	*Dev94*	*Eval94*	*Mean*
20K	12.4	10.9	8.4	10.5
58K	14.3	12.1	11.2	12.5

(*b*) %Improvement Over Lexical Tree Search.

	Dev93	*Dev94*	*Eval94*	*Mean*
20K(Vit.)	5.3	7.0	4.0	5.4
20K(A*)	12.4	11.2	8.1	10.7
58K(Vit.)	9.9	7.7	5.2	7.7
58K(A*)	19.8	11.4	10.3	14.0

(*c*) %Degradation w.r.t. Baseline System.

Table 4.7: Word Error Rates from Global Best Path Search of Word Lattice Produced by Lexical Tree Search.

Recognition Speed

Table 4.8 summarizes the average execution times of the shortest path algorithm. The computational overhead associated with this step is negligible. This is to be expected since the DAG size is usually small. For a 10sec long sentence, it typically consists of a few hundred nodes and a few thousand edges.

	Dev93	*Dev94*	*Eval94*	*Mean*
20K	0.04	0.05	0.05	0.05
58K	0.07	0.08	0.09	0.08

Table 4.8: Execution Times for Global Best Path DAG Search (x RealTime).

4.3.3 Best Path Search Summary

Performing a global best path search over the word lattice output of the lexical tree Viterbi search is unquestionably advantageous. The resulting accuracy is not significantly worse than that of the baseline system, and it is achieved at an almost negligible cost.

The DAG search is similar to the A* search of the baseline system in that it finds a globally optimum path, but it does so at a much lower computational cost, as seen by comparing Tables 4.8 and 3.3. On the other hand, the A* pass in the baseline system is not restricted to low-order n-gram grammars, and uses additional word segmentation information from the backward Viterbi pass, which is the main reason for its superior word accuracy. We note that we are not precluded from applying the latter passes of the baseline system to the word lattice output of the lexical tree Viterbi search. Likewise, we can apply the shortest path algorithm to the word lattice output of the forward Viterbi pass of the baseline system. One reason to avoid a backward pass is that many practical systems need *online operation*, and it is desirable not wait for the end of the utterance before beginning a search pass.

The DAG search algorithm, while sufficiently fast to be unobservable for short sentences, can cause noticeable delay for a long sentence. But this can be avoided by overlapping the search with the forward pass. The nodes of the DAG can be built incrementally as the forward Viterbi pass produces new word lattice entries. The addition of a node only affects existing nodes that immediately precede the new one and nodes that occur later in time. Since the Viterbi search is a time-synchronous algorithm, it is likely to add new nodes towards the end of the DAG, and hence the update to the DAG for the addition of each new node is minimal.

There are several other uses for the DAG structure. For example, once the word lattice is created, it is possible to search it efficiently using several different parameters for language weight, word insertion penalties, etc. in order to tune such parameters. One can also search the DAG several times using different language models, effectively in parallel. The result is a measure of the posterior probability of each language model, given the input speech. This can be useful when users are allowed speak phrases from different domains without explicit prior notification, and the system has to automatically identify the intended domain by trying out the associated language models in parallel.

The creation of word graphs and some of their uses as described here, has also been reported in [65]. However, they do not report on the performance issues or on the use of the shortest path algorithm to find a global optimum that overcomes the suboptimality of the Viterbi search algorithm. A final word on the use of the global path search is that it essentially eliminates the need for a full trigram search during the first pass Viterbi search. Thus, the approximate use of a trigram grammar in the forward pass, in the manner described in Section 3.2.2 is quite justified.

4.4 Rescoring Tree-Search Word Lattice

4.4.1 Motivation

We noted in Section 4.2.5 that the word lattice output of the lexical tree Viterbi search is quite compact, consisting of only several hundreds or thousands of words for a 10sec sentence, on average, and that its low lattice error rate makes it an ideal input for postprocessing with models and algorithms of higher sophistication.

The main purpose of this section is to obtain a measure of the quality of the word lattice produced by the lexical tree search. This is relevant for understanding what we lose or gain by postprocessing the word lattice with detailed models instead of performing a complete search with such models. The parameters that determine the quality of the word lattice include its size and the lattice error rate. The former measures the work needed to search the lattice, while the latter sets an upper bound on the recognition accuracy.

We measure these parameters indirectly by *rescoring* the lexical tree word lattice with the forward pass Viterbi search of the baseline Sphinx-II system. The main difference between the two is that the rescoring pass is restricted to searching words in the lattice. By comparing the recognition accuracy and search time overhead with the baseline system results in Section 3.4, we ascertain the quality of the lexical tree word lattice output.

We believe that the suboptimality of the tree search word lattice manifests itself as occasionally poor word segmentations that we observe by manually comparing them with the baseline Sphinx-II system. To overcome this shortcoming, we allow the word boundaries in the input lattice to be treated in a fuzzy manner. In other words, at any time t in the rescoring pass, we allow cross-word transitions to those words in the tree search word lattice that begin within a given *window* of t. We can afford to be generous with the window since the size of the input word lattice is small anyway. In our experiments we use a window of 25 frames although that is probably an order of magnitude larger than necessary.

The output of the rescoring pass is another word lattice that presumably has correct word segmentations. This lattice is then searched using the global best path algorithm described in Section 4.3 to produce the final recognition result.

4.4.2 Performance

The new configuration consisting of three passes—lexical tree search, rescoring its word lattice output, and global best path search of the rescored word lattice—has been tested on our benchmark test sets. We use the same set of acoustic and language models in the rescoring pass as in the lexical tree search and the baseline system.

	Dev93	*Dev94*	*Eval94*	*Mean*
20K	17.4	16.0	15.9	16.4
58K	15.0	14.4	14.5	14.7

(a) Absolute Word Error Rates(%).

	Dev93	*Dev94*	*Eval94*	*Mean*
20K(Vit.)	-1.0	1.3	0.2	0.1
20K(A*)	5.7	5.2	4.2	5.0
58K(Vit.)	-0.1	0.7	0.3	0.4
58K(A*)	8.9	4.2	5.2	6.2

(b) %Degradation w.r.t. Baseline System. (Negative values indicate *improvement* over baseline system.)

Table 4.9: Word Error Rates From Lexical Tree+Rescoring+Best Path Search.

	Dev93	*Dev94*	*Eval94*	*Mean*
20K	0.72	0.76	0.80	0.76
58K	1.25	1.26	1.35	1.28

(a) Rescoring Pass Alone (xRealTime).

	Dev93	*Dev94*	*Eval94*	*Mean*	Speedup Over Baseline Fwd.Vit.
20K	5.4	5.5	5.5	5.5	4.04
58K	10.2	10.0	10.0	10.0	4.24

(b) Lexical Tree+Rescoring+Best Path Search (xRealTime).

Table 4.10: Execution Times With Rescoring Pass.

Table 4.9 summarizes the word error rates obtained by using all three passes, and compares them to the baseline system. The bottom line is that there is no difference in accuracy between the new three-pass search and the baseline one-pass search. One reason the three-pass baseline system is still better is that the backward pass provides additional word segmentations to the A* pass, that are lacking in the lexical tree-based system.

Table 4.10 shows the execution times with the rescoring pass. We note that our implementation has not been optimized or tuned. Hence, the costs shown in the table are somewhat on the higher side.

4.4.3 Summary

The main conclusion from this section is that the word lattice output of the lexical tree is compact and has a low lattice-error rate. We are able to recover the baseline system recognition accuracy at the cost of about 15% increase in computation. The computational overhead is less than 5% compared to the forward pass of the baseline system. The 3-pass tree-based recognizer is about 4.1 times faster than the forward pass of the baseline system on our benchmarks.

We also conclude that though the tree search is sub-optimal in that the output word segmentations are occasionally incorrect, it is possible to recover the loss using a second rescoring pass similar to the first pass of the baseline Sphinx-II system. The A* accuracy of the baseline system is still slightly better because of the additional word segmentations provided by its backward pass.

4.5 Phonetic Fast Match

4.5.1 Motivation

We have described the senone as the acoustic model shared by a cluster of phonetic HMM states (Sections 2.1.2 and 3.1.1). Mei-Yuh Hwang in her dissertation [27] has pointed out that "...each senone describes a very short distinct acoustic event (shorter than a phoneme)...," and "...it can be used to construct models of all kinds of acoustic phenomena." One of the phenomena modelled by senones is the relative activity of the different phonemes at any given time.

Let us see how senones can be used to predict the presence or absence of a given basephone at a given point in the input speech. In Sphinx, clusters of HMM states that form a senone can only belong to a single parent basephone. That is, senones are partitioned among basephones. Consider all basephones $p_i, i = 1, 2, \ldots$. Let $CD(p_i)$ represent the collection of *c*ontext *d*ependent triphones derived from p_i as well as p_i itself. We say that senone $s \epsilon CD(p_i)$, if s models any state of any phone in $CD(p_i)$. (Similarly, $s \epsilon p_i$, if s models any of the states of the basephone p_i.) At each time instant t, we compute *base phone scores* given by:

$$P_{p_i}(t) = \max_{s \; \epsilon \; CD(p_i)} (b_s(t)), i = 1, 2, \ldots \tag{4.2}$$

where $b_s(t)$ is the output probability of senone s at t. That is, $P_{p_i}(t)$ is the output probability, at time t, of the best scoring senone belonging to basephone p_i or any triphone derived from it. Equation 4.2 defines an ordering or ranking among all basephones, as well as the acoustic separation between them, in each frame.

We can use the above ranking as a measure of the relative activity of the individual basephones at any point in time. The basic understanding is that if $P_{p_i}(t) >> P_{p_j}(t)$

for some two basephones p_i and p_j, then none of the states derived from p_j will score well at t, and all instances of HMMs derived from p_j can be pruned from search. In other words, by setting a pruning threshold or beam width relative to the best scoring basephone, we can limit the search at time t to just those falling within the threshold. The phones within the beamwidth at t are the list of *active candidates* to be searched at t. Because of this similarity to word-level fast match techniques that identify candidate words active at a given time, we call this the *phonetic fast match* heuristic.

The proposed heuristic raises the following issues:

- Like other fast match techniques, the phonetic fast match can cause pruning errors during search. This occurs because senone scores are noisy, as we shall see later in Section 4.5.2, and they occasionally mispredict the active phones. We explore efficient ways of minimizing pruning errors later under this section.

- Equation 4.2 requires the computation of all senone output probabilities in order to determine the base phone scores. That is relatively expensive for a fast match heuristic. We can also obtain base phone scores from just the context *in*dependent senones; i.e., $s \in p_i$ instead of $s \in CD(p_i)$ in equation 4.2. However, by omitting the more detailed context dependent senones from the heuristic, we make the phone scores and ranking less reliable, and the beamwidth must be increased to avoid pruning errors. We explore the trade-offs presented by these alternatives.

A somewhat similar approach to search pruning has also been suggested in [56, 31]. In their work, phones are pruned from the search process based on their *posterior* probabilities estimated using neural network models. It is also different in that the pruning mechanism is embedded in a hybrid Viterbi-stack decoding algorithm. Finally, we use the phone prediction mechanism to activate new phones at a given point in the search, and not to deactivate already active ones, unlike in their case. We believe that this leads to a more robust pruning heuristic given the nature of our semi-continuous acoustic models.

In Section 4.5.2, we present the details of the heuristic and its incorporation as a fast match front end into the lexical tree search. This description is primarily based on equation 4.2, i.e., using *all* senones to compute the base phone scores. But most of it also applies to the alternative scheme of using only the context independent senones. We present details of the performance of both schemes on our benchmarks in Sections 4.5.3 and 4.5.4, respectively.

4.5.2 Details of Phonetic Fast Match

Phones Predicted by Best Scoring Senones

We first consider an example of the base phone ranking produced by equation 4.2. Figure 4.11 illustrates the heuristic with an example extracted from one of our benchmark tests. Each row represents one frame of speech. All senone output probabilities are computed in each frame to obtain a base phone score. The base phones are ranked accordingly and pruned with a certain threshold. We list the remaining active phones in each frame in descending order of their scores. (See Appendix A for a complete list of the 50 context independent phones used in Sphinx-II.) The figure underscores several points:

- The candidate basephone list in each frame, even though quite short, *appears* to contain the correct base phone, and quite often at the head of the list. We emphasize *appears* because, *a priori*, it is by no means clear which base phone is the "correct" one in any given frame. At this point we can only visually discern a pattern in the candidate lists that seems to match the expected basephone sequence fairly well.
- It is obvious that the best phone in a frame is certainly not always the correct one, whatever that may be, since we sometimes observe a best phone that is not *any* of the correct ones. Hence, it *is* necessary to look further down the list for the correct basephone.
- The choice of the pruning threshold is crucial. Too tight a threshold causes the correct phone to be pruned entirely from the list. On the other hand, if it is too wide, we end up with too many unnecessary candidates.
- The length of the list varies from frame to frame, indicating the acoustic confusability within each frame. The confusion is higher around phone boundaries.

Quality of Phone Prediction

We can estimate the quality of this heuristic by measuring the position of the correct base phone in the candidate list in each frame. But we first need to know what the correct phone is in a given frame. For that we use the *Viterbi alignment* [52] of the correct sequence of phones to the input speech[10]. Specifically, the experiment consists of the following steps:

1. Obtain the Viterbi alignment for an entire test set. This gives us a correct basephone mapping for each frame in the test set.

[10]The choice of Viterbi alignment as the reference is debatable. But we believe that any other alignment process will not make a significant difference to the resulting phone segmentations.

```
Frame#   Base phones active in each frame, ranked by score
------ ----------------------------------------------------
[ 331] dh  b  p th  d ax ix  k td
[ 332] dh  b th  p ix ax ih eh  k ae ey  d  g td
[ 333] dh  b th  p ax ix ey ih eh
[ 334] dh th ey ih ax eh  b ix  p ae  t  d
[ 335] ih ax ey  t dh ix eh  d ae th  b  p
[ 336] ih ix ey ax ae eh  t  d
[ 337] ih ix ax ey ae eh er
[ 338] ix ih ax er ey eh
[ 339] ih ix ax ey eh er uw ah
[ 340] ih ix ax ey uw
[ 341] ix ih ax  z
[ 342]  z ix  s ax
[ 343]  z  s
[ 344]  z  s ts
[ 345]  s  z ts
[ 346]  s  z ts
[ 347]  s  z ts td
[ 348]  s td  z ts  t dd
[ 349] td  t  s  z
[ 350]  t td ch
[ 351]  t td ch jh
[ 352]  t td dd  d ch
[ 353]  t td  k dd  p  d
[ 354]  t td  k dd
[ 355]  t  f ch hh td sh  k jh
[ 356] ch sh  t  y  f jh hh  s  k
[ 357]  f ch  t sh  y hh  p  s th  k
[ 358]  r ch  t  y ae hh  f  p eh  k th ax sh er ix  d  s
[ 359]  r ae eh ax
[ 360] eh ae  r ey aw ah aa ih ax ow ix
[ 361] ae eh aw ey ax ah ih ix  r ow ay
[ 362] ae eh ax ih ey ix aw ah
[ 363] ax ae eh ih ix aw ey ah iy
[ 364] eh ax ae ih ix aw
[ 365] eh ax ae ix ih ah
[ 366] eh ax ah ix ih ae  n ay
[ 367]  n eh ix ah ng ax ih ae ay
[ 368]  n ng ix
[ 369]  n ng
[ 370]  n dd  m
[ 371]  n dd  m ng
[ 372]  n dd  m ng  d
[ 373]  n dd  d  m ng dh td
[ 374] dd  d  n dh ng  m  y td  g  v  l
[ 375] dd  d  n dh ng td  y ix  g  m
```

Figure 4.11: Base Phones Predicted by Top Scoring Senones in Each Frame; Speech Fragment for Phrase *THIS TREND*, Pronounced *DH-IX-S T-R-EH-N-DD*.

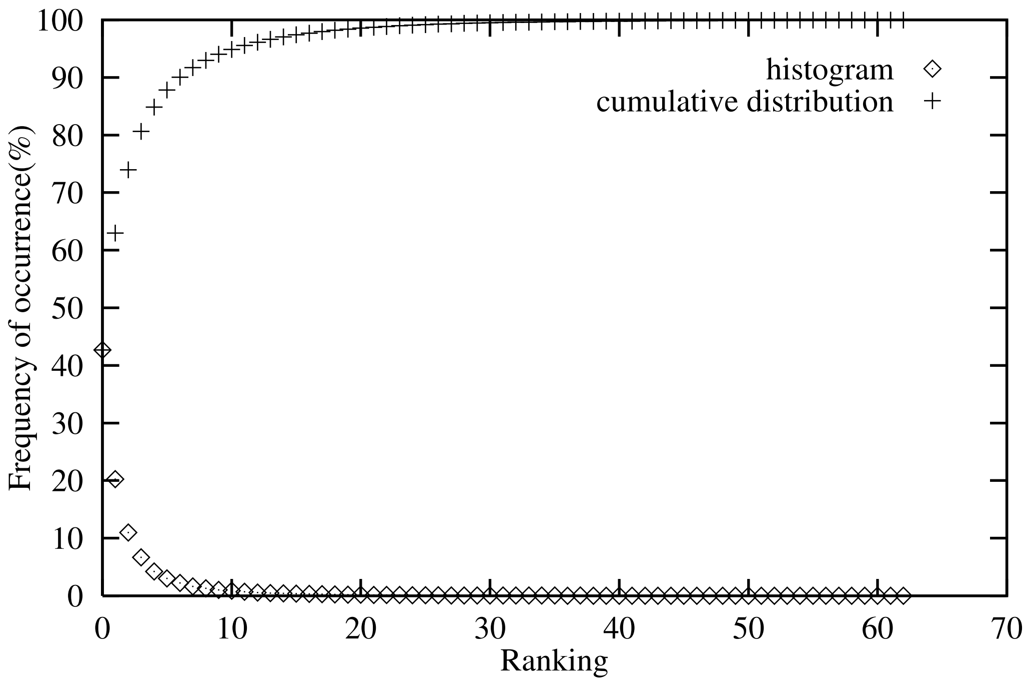


Figure 4.12: Position of Correct Phone in Ranking Created by Phonetic Fast Match.

2. Use the heuristic of equation 4.2 to obtain an ordering of all basephones in each frame (without any pruning).

3. In each frame, identify the position of the correct base phone in the ranked list.

This test was run on a random sampling of sentences chosen from our benchmark test sets described in Section 3.4.1. Figure 4.12 shows the resulting histogram of the position of the correct base phone[11] in the ordered list created by equation 4.2. For example, we note that the correct base phone occurs at the very head of the ordered list created by the phonetic fast match over 40% of the time. It is just one away from the head of the list 20% of the time.

The figure also includes the cumulative distribution corresponding to the histogram. It shows that the correct phone in a given frame is missing from the top 10 phones only about 5% of the time, and from the top 20 phones only 1% of the time. The number of pruning errors is proportional to the frequency of misses.

Clearly, we should use as tight a pruning threshold as possible without incurring an excessive number of pruning errors. Such errors occur if the correct base phone is not within the pruning threshold in some frame. With a tight threshold, they can occur fairly often. For a given threshold, pruning errors are more likely around phone boundaries, where the degree of confusability between phones is higher.

[11]The figure shows 63 basephones instead of the 50 listed in Appendix A. The remaining 13 correspond to various noise phones.

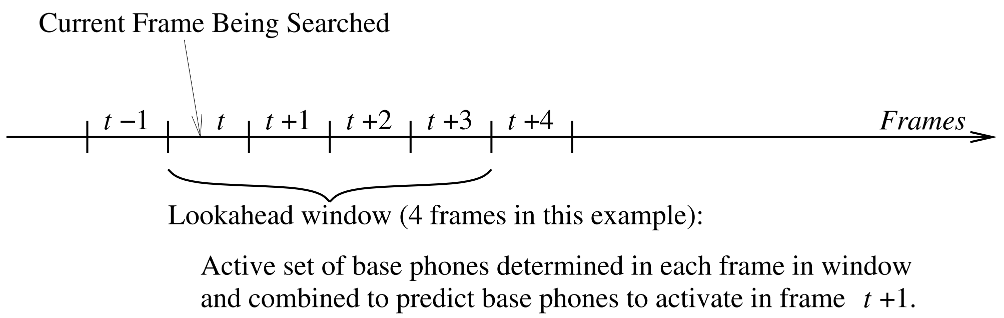


Figure 4.13: Lookahead Window for Smoothing the Active Phone List.

Reducing Pruning Errors Through Smoothing

Pruning errors can be reduced if we use the candidate list to determine when to activate new phones, but not to *de*-activate them. That is, once activated, a phone is searched as usual by the Viterbi beam search algorithm.

Secondly, the candidate list can also be made more robust by *smoothing* it by considering all candidate phones from a *window* of neighbouring frames. If the candidate list is used only to activate new phones to be searched in the future, the window should be positioned to look ahead into the future as well. Hence, we call it a *lookahead window*. The two strategies together overcome the problem of sporadic holes in the active phone list, especially around phone boundaries.

Figure 4.13 summarizes the lookahead window scheme used in our experiments. A window of 2 or 3 frames is usually sufficient, as we shall see in the experimental results.

Algorithm Summary

The phonetic fast match heuristic has been incorporated into the lexical tree search pass of the decoder. The resulting modification to the original algorithm (see Figure 4.6) is straightforward:

1. We compute all senones scores in each frame instead of just the active ones, since the base phone scores are derived from all senones (equation 4.2).

2. Since the active phones are determined by looking ahead into future frames, senone output probability evaluation (and hence Gaussian density or VQ computation) has to lead the rest by the size of the lookahead window.

3. In each frame, before we handle any cross-HMM or cross-word transition, the list of new candidate phones that can be activated is obtained by the phonetic fast

match heuristic together with the lookahead window mechanism. If a basephone is active, all triphones derived from it are also considered to be active.

4. In each frame, cross-HMM and cross-word transitions are made only to the members in the active phone list determined above.

The main drawback of this heuristic is that one needs to compute the output probabilities of all senones in each frame in order to determine the phones to be activated in that frame. This overhead is partly offset by the fact that we no longer need to track down the active senones in each frame by scanning the list of active HMMs. The alternative heuristic proposed at the end of Section 4.5.1 determines the active phones from just the context-independent senone, and does not require all of the context-dependent senone scores.

4.5.3 Performance of Fast Match Using All Senones

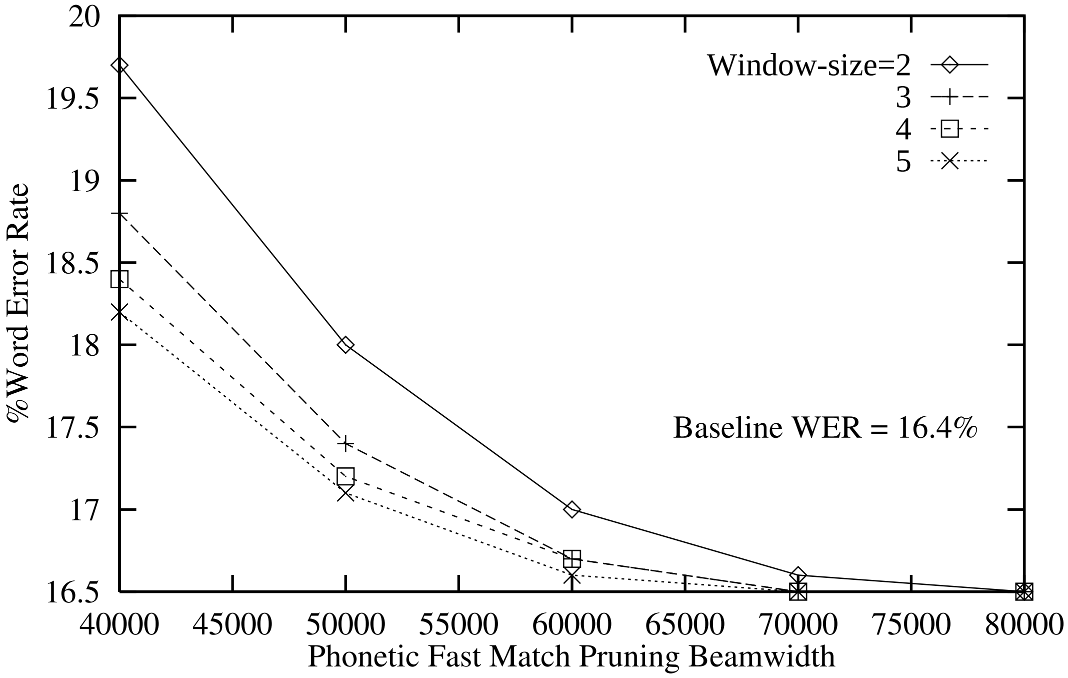
We measure the performance of the decoder in its full configuration—lexical tree search augmented with the phonetic fast match heuristic, followed by the rescoring pass (Section 4.4) and global best path search (Section 4.3). The benchmark test sets and experimental conditions have been defined in Section 3.4.1.

The two main parameters for this performance evaluation are the lookahead window width and the active phone pruning threshold. The benchmark test sets are decoded for a range of these parameters values. Since listing the performance on each test set individually for all parametric runs makes the presentation too cluttered, we only show the performance aggregated over all test sets.

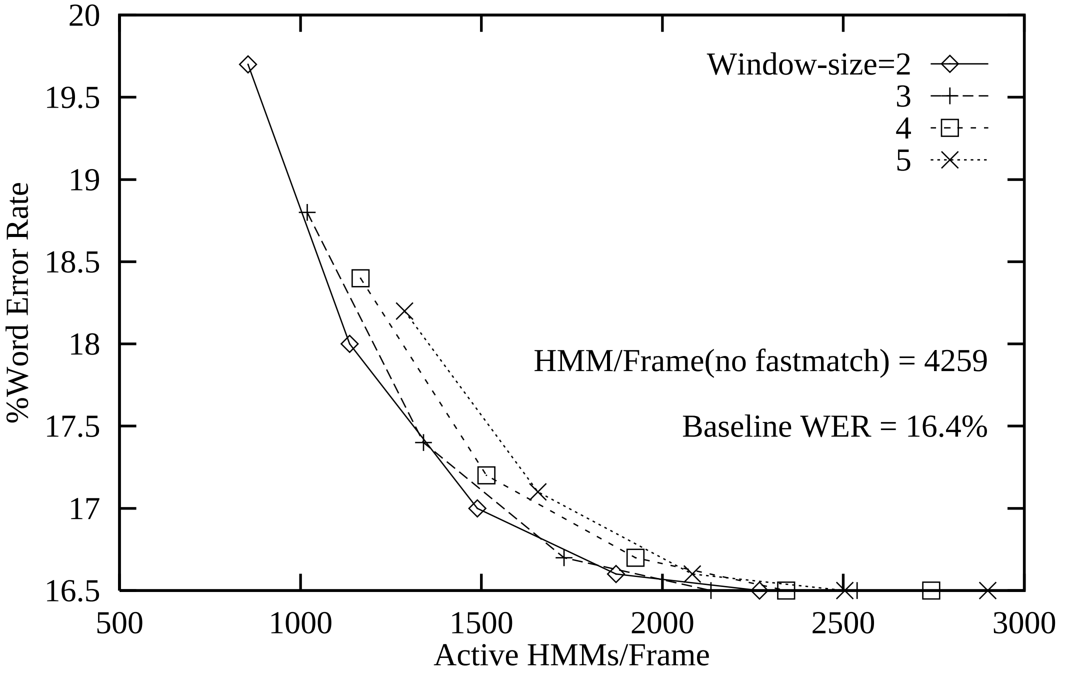
20K Task

Figure 4.14 shows two sets of performance curves for the *20K* task. In Figure 4.14(a), each curve shows the variation in word error rate with changing pruning threshold for a fixed lookahead window size. (The pruning threshold is in negative log-probability scale, using a log-base of 1.0001.) With the fast match heuristic we asymptotically reach the baseline word error rate of 16.4% as the pruning beamdwidth is widened. It is also clear that increasing the lookahead window size for a fixed beamwidth helps accuracy.

Figure 4.14(b) shows the same data points, but instead of the pruning beamdwidth, we plot the number of active HMMs per frame along the x-axis. Comparing this graph to the statistics for the lexical tree search without phonetic fast match (Section 4.2.4, Table 4.4), it is clear that we can essentially attain the baseline word error rate of 16.4% while reducing the number of active HMMs by about 50%.



(a) Word Error Rate *vs* Fast Match Pruning Beam Width.



(b) Word Error Rate *vs* Active HMMs/Frame in Tree Search.

Figure 4.14: Phonetic Fast Match Performance Using All Senones (*20K* Task).

Figure 4.14(*b*) also demonstrates that increasing the window size beyond 2 or 3 frames is not worthwhile. We conclude this from the fact that to achieve a given recognition accuracy, we do more work as the window size grows larger. It also makes intuitive sense that a smoothing window of 2-3 frames is all that is needed since the window is needed to cover "holes" in the active phones predicted by the heuristic. These holes occur at random and not in long bursts. In the following discussion, we use a window of size of 3 frames, which seems to be an optimal choice considering Figure 4.14.

Table 4.11 summarizes the performance results on the *20K* task from including the phonetic fast match heuristic. It corresponds to a fixed lookahead window size of 3 frames. With the incorporation of the phonetic fast match heuristic in the tree Viterbi search, there is a significant speedup in the total execution time, with a negligible increase in the word error rate. If we consider only the search time, excluding the acoustic output probability computation, the improvement in speed is between a factor of 2 to 3.

Pruning beamwidth	40K	50K	60K	70K	80K	No fastmatch
%Word Error Rate	18.8	17.4	16.7	16.5	16.5	16.4
No. HMMs/Frame	1019	1340	1728	2134	2538	4259
No. LM Ops/Frame	151	193	245	302	360	625
Total Time (xRealTime) (All Three Passes)	3.24	3.48	3.77	4.06	4.36	5.50
Speedup Over Baseline Forward Viterbi	6.86	6.39	5.90	5.48	5.10	4.04

Table 4.11: Fast Match Using All Senones; Lookahead Window=3 (*20K* Task).

58K Task

We show similar performance figures for the *58K* task in Table 4.12, with the lookahead window fixed once again at 3 frames. The conclusions are quite similar to those in the *20K* task. There is a factor of 2 reduction in the number of active HMMs per frame and in the search time, with no appreciable increase in recognition errors. The reduction in the overall execution time is more marked since the acoustic model evaluation is a relatively smaller fraction of the total computation.

Pruning beamwidth	40K	50K	60K	70K	80K	No fastmatch
%Word Error Rate	17.0	15.7	15.1	14.9	14.8	14.7
No. HMMs/Frame	1636	2195	2883	3604	4322	7369
No. LM Ops/Frame	457	559	685	820	958	1595
Total Time (xRealTime) (All Three Passes)	4.42	4.75	5.36	5.99	6.50	10.0
Speedup Over Baseline Forward Viterbi	9.60	8.93	7.92	7.08	6.53	4.24

Table 4.12: Fast Match Using All Senones; Lookahead Window=3 (*58K* Task).

4.5.4 Performance of Fast Match Using CI Senones

The main drawback of the fast match heuristic discussed in Section 4.5.2 is that all senones have to be computed in every frame in order to determine the active base phones. In this section we consider predicting the active basephones from just the context-independent senones. That is, equation 4.2 now becomes:

$$P_{p_i}(t) = \max_{s \,\epsilon\, p_i}(b_s(t)), i = 1, 2, \ldots \tag{4.3}$$

Thus, we still have to evaluate all senones of all basephones in every frame in order to determine the active phone list, but this is a much smaller number than the total number of senones. The context dependent senones can be evaluated on demand.

We expect the list of candidate phones indicated by this heuristic to be somewhat less reliable than the earlier one since context independent senones are cruder models of speech. In particular, coarticulation effects across phone boundaries in fluent speech are captured by triphone models but might be missed by the broader monophone models. On the other hand, we still use the lookahead window to smooth the candidate list from neighbouring frames. Overall, it is a tradeoff between a potential increase in the number of active HMMs in order to retain the original level of recognition accuracy, and a reduction in the number of senones evaluated per frame.

Table 4.13 shows a summary of the performance of this scheme on the *20K* and *58K* tasks. The lookahead window size is fixed at 3 frames. As we expected, the pruning threshold has had to be widened, compared to the previous scheme.

A Comparison with the results in Section 4.5.3 brings out the following observations:

- We are able to match the baseline recognition accuracy with a further reduction in computation. In both the *20K* and *58K* tasks, there is a reduction in total execution time of about 45% for a less than 2% relative increase in word error rate, compared to not using the fast match heuristic.

Pruning beamwidth	65K	75K	85K	95K	105K	No fastmatch
%Word Error Rate	18.1	17.1	16.7	16.6	16.5	16.4
No. HMMs/Frame	1402	1707	2016	2290	2580	4259
Total Time (xRealTime) (All Three Passes)	2.41	2.72	3.02	3.29	3.55	5.50
Speedup Over Baseline Forward Viterbi	9.23	8.18	7.36	6.76	6.26	4.04

(*a*) *20K* Task.

Pruning beamwidth	65K	75K	85K	95K	105K	No fastmatch
%Word Error Rate	16.6	15.6	15.1	14.9	14.8	14.7
No. HMMs/Frame	2334	2878	3430	3919	4438	7369
Total Time (xRealTime) (All Three Passes)	4.20	4.82	5.45	5.84	6.58	10.0
Speedup Over Baseline Forward Viterbi	10.1	8.80	7.79	7.27	6.45	4.24

(*b*) *58K* Task.

Table 4.13: Fast Match Using CI Senones; Lookahead Window=3.

- There is some increase in the number of active HMMs needed to achieve the same recognition accuracy, compared to the fast match based on all senones. But it is partially offset by the reduction in the number of senones evaluated. The balance, of course, depends on the relative costs of the two.

- Using context independent senones for the fast match is more beneficial to the smaller vocabulary system. As vocabulary size increases, the increase in the number of active HMMs affects larger vocabulary systems more. The reduction in the cost of computing senone output probabilities is offset by the need to scan the active HMMs to find the active senones in each frame.

4.5.5 Phonetic Fast Match Summary

The conclusion is that using the phonetic fast match heuristic, the speed of the lexical tree decoder can be improved by almost a factor of two, with negligible increase in recognition errors. The heuristic can be further tuned in ways that we have not explored. For example, one can use basephone-specific pruning beamwidths. The graph in Figure 4.12 is averaged over all 50 phones in the Sphinx-II system. However, certain base phones, such as fricatives and long vowels, are predicted much better than others. It is possible to tighten the pruning threshold for these phones.

The phonetic fast match also has several other potential uses. These need to be explored:

- The active base phones usually occur clustered in time. One can create a *phone lattice* in which each node is a phone that is active for some contiguous set of frames. Such a phoneme lattice could be searched very efficiently for producing a candidate list of active *words* to be searched.
- A phone lattice such as described above can be used for word spotting. We may consider the word to be present if we can trace a sequence of nodes through the phone lattice corresponding to the word's pronunciation.
- Obtaining confidence measures. Wherever the phone lattice is sparse, i.e., very few phones are identified as being active, we can conclude that we have a high degree of confidence about which phones are active. If a region of frames has many phones active in it, on the other hand, it indicates a high degree of acoustic confusion in that region. The acoustic modelling in that segment is less reliable.

The technique of identifying active phones has been discussed in [22], however, only in the context of applying it to a fast match. They have reported a reduction in fast-match computation of about 50% with a slightly under 10% increase in error rate. A similar technique using phone posterior probabilities has also been reported in [56]. It is also in the phase of the fastmatch step that generates word candidates and posterior probabilities to a stack-decoding algorithm. The phonetic HMM states are modelled by neural network that directly estimate phone posterior probabilities that are used to identify and deactivate inactive phones during the fast match step. They report an order of magnitude increase in search speed for a nearly real-time performance, while incurring a 7% relative increase in word error rate on a 20K task. Part of the large increase in speed in probably owing to the fact that the basic decoder is based on stack-decoding algorithm. They do not report frequency counts for the reduction in the number of active models per frame.

4.6 Exploiting Concurrency

Our main purpose in this section is to explore the potential for speeding up the recognition architecture via concurrency. It is relevant since modern commercial processor architectures are increasingly capable of multiprocessor operation and commercial operating systems support concurrency and multithreading within single applications. It is possible to take good advantage of this facility.

One of the early attempts at speeding up the Sphinx-II baseline system exploited the large degree of concurrency within its algorithmic steps [54]. In a parallel implementation on the PLUS multiprocessor designed at CMU [13], a speed up of 3.9 was obtained on a 5 node configuration. The parallelization involved static partitioning of data and computation among multiple *threads* using the Mach *C-threads* facility [16]. Though the lexical tree decoder has significant structural differences compared to the baseline system, some of the parallelization techniques can still be applied to it. We explore this question in some detail in this section.

In parallelizing any application, there are two broad options: dynamic load balancing using a central task queue model, or static load balancing based on some suitable static data and computation partitioning. The former presumably achieves the best (most even) load balance among the active threads. But it is more complex to implement, and harder to maintain and modify for experimentation purposes. We consider static partitioning schemes which are easier to manipulate in a coarse-grained fashion on small-scale multiprocessors.

In the following discussion we assume a shared-memory multiprocessor configuration with an application running in a single address space shared by multiple concurrent threads. The discussion is hypothetical. We have not actually implemented a parallel version of the tree search decoder. But we do address the relevant issues.

4.6.1 Multiple Levels of Concurrency

There are several levels of concurrency in the decoder as discussed so far, which can be exploited individually and in combination. We review the main ones briefly.

Pipelining Between Search Passes

The lexical tree search and rescoring passes can be pipelined and executed concurrently. If the latter is time synchronous, the only constraint is that it cannot proceed beyond time t until the lexical tree search has completed emitting all its word lattice entries that correspond to a begin time of t or earlier. This is easily established by checking the history information H_j^m in each active HMM (see Section 3.2.4). If all of them point to word lattice entries with end times greater than t, the rescoring pass can proceed beyond t[12].

The potential speedup obviously depends on the relative costs of the two passes. In our benchmark system, the lexical tree search is the dominant pass. However, it can be parallelized internally, so pipelining the later pass becomes useful.

The communication bandwidth between the two passes is minimal. It only involves the exchange of word lattice information, which consists of a few tens of words every frame, on average. The two passes must also synchronize to ensure that the rescoring pass does not overtake the other.

[12] Actually, the rescoring pass treats the word segmentations in the tree search word lattice *fuzzily*, using a window, as described in Section 4.4.1. We must also allow for the window.

Pipelining Between Acoustic and HMM Evaluation in Tree Search

We can pipeline the VQ and senone output probability computation with the HMM search if there is no feedback from the latter to the former. That is the case if we decide to evaluate all senones every frame, instead of computing just the active ones. The latter requires feedback from the active HMMs. The phonetic fast match heuristic presented in Section 4.5.2 requires the evaluation of all senone output probabilities every frame anyway. Hence this pipelining is suitable in that context.

Again, the potential speedup depends on the costs of the individual steps. In our lexical tree search using semi-continuous acoustic models, the execution times for senone output probability computation and searching the active HMMs are fairly even for the *20K* task. This is seen from Tables 4.3(a) and 4.11. Even otherwise, each component has a fair amount of internal parallelism which can be exploited to balance the computation.

We consider the communication bandwidth between the two components. The data exchanged between the pipeline components consists of the senone output probabilities in each frame. In our system, it amounts to 10K 4-byte values in each frame, or about 4MB/sec, and it is not likely to be significantly different for most systems. This volume is well within the capabilities of modern memory-bus systems.

The two pipeline components have a producer-consumer relationship. The producer, senone evaluation, must stay ahead of the consumer, the search component. The execution time for the former is fairly constant from frame to frame, but the search cost can vary by an order of magnitude or more. Hence, for a proper balance in computation, the two cannot be simply run in lockstep. We need a queue of frames between the two to smooth out variations in execution times. However, the queue cannot be arbitrarily large since it has to contain 40KB of senone output probability data per frame. We surmise that a queue depth of about 10 frames is sufficient to keep the pipeline flowing relatively smoothly without hiccups.

Partitioning Acoustic Output Probability Computation

The computation of senone output probabilities can be trivially partitioned in most instances. Basically, there is no dependency between the output probabilities of different senones within a given frame. Every senone can be evaluated concurrently with the others.

Partitioning the senone output probability computation creates a multiple-producer and consumer relationship with the search module. Hence the latter must synchronize with all of the producer partitions to ensure integrity.

Concurrency in Searching the Lexical Tree HMM Network

Parallelism inside the Viterbi tree search algorithm is the hardest to control and exploit effectively for several reasons:

- The beam search heuristic restricts the active HMMs to a small fraction of the total number. The actual identity of the active HMMs is time varying. It is necessary to balance the computation among parallel components in spite of this variation.
- The amount of search computation varies by orders of magnitude from frame to frame.
- There is global dependency from one frame to the next. This follows from the time-synchronous nature of the Viterbi algorithm. For example, the word lattice must be completely updated with new word entries in a given frame before moving on to attempt cross-word transitions in that frame.
- The search component is the most memory intensive. The memory access pattern is highly unstructured since the set of active HMMs varies over time. The memory bottleneck is probably the largest impediment to the effective parallelization of this component.

The most natural way to parallelize the lexical tree search is by partitioning the collection of trees among concurrent threads. For example, the two trees in Figure 4.2 can be assigned to separate threads. This approach has the following advantages:

- The computation within each thread is largely the same as in the sequential algorithm. It is important to retain the simplicity of structure offered by sequential implementations for ease of modification and experimentation. The main difference is that threads need to synchronize with each other once in each frame to exchange updates to the word lattice.
- Since the HMM data structures to be searched are partitioned, there is no conflict during access to them, except for cross-word transition updates. These are handled by exchanging updates to the word lattice as mentioned above.
- The large number of trees offer sufficient parallelism and scope for effective load balancing in spite of the level of activity within individual trees varying substantially with time. In the *20K* task, there are about 650 trees (see Table 4.1).

It is important to distribute the trees with some care, in order to maintain a proper load balance among threads. The phonetic fast match heuristic restricts the activity at the root nodes by allowing only the predicted phones to become active at a given point in time. Hence, it is advantageous to scatter trees with root nodes that have the same parent basephone among different threads. In general, it is desirable to spread trees with acoustically similar initial prefixes among threads, ensuring a more even distribution of the computation load.

4.6.2 Parallelization Summary

We have reviewed the potential for speedup through parallelism at several levels. Even with a static partitioning of the task, the available concurrency can be effectively exploited on small-scale shared-memory multiprocessors.

It is possible to pipeline individual passes of the decoder with one another. Within the lexical tree search, we can exploit pipelining between the acoustic output probability evaluation for senones and the HMM search algorithm. This is especially advantageous since they are fairly evenly matched in our lexical tree decoder using semi-continuous models. Finally, both acoustic output probability computation and HMM search can be partitioned into concurrent threads. However, the latter requires a careful static assignment of the overall search space to threads in order to balance the computational load among them.

However, the effectiveness of a parallel implementation is limited by the available processor-memory bandwidth. Since certain portions of the original sequential algorithm, especially HMM evaluation, are heavily memory bound, the actual speedup possible through concurrency remains to be seen.

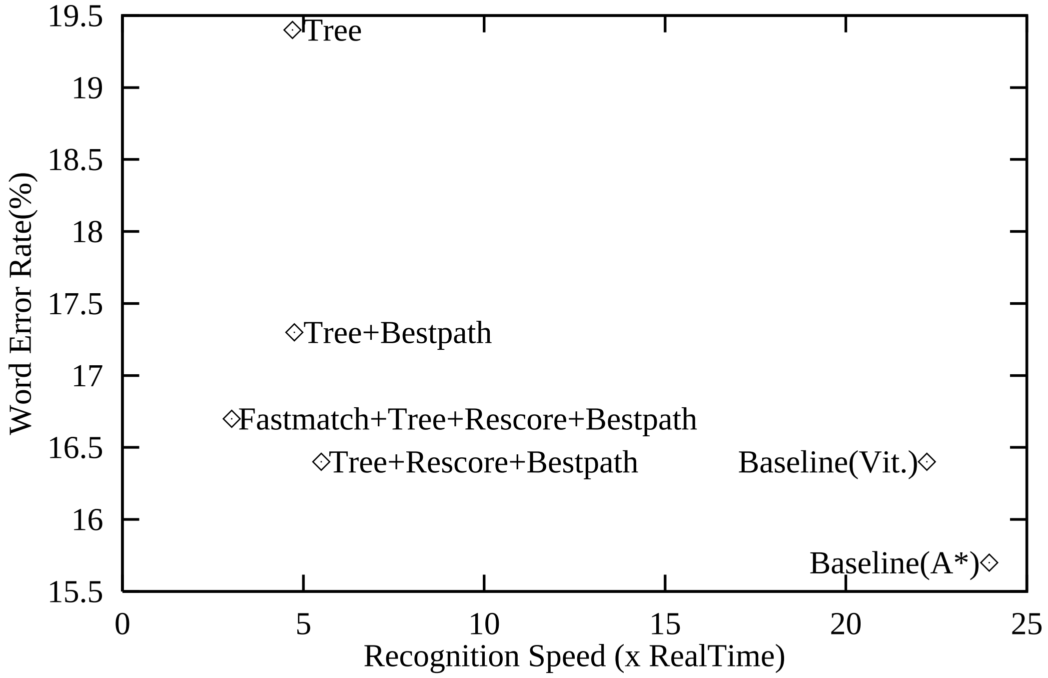
4.7 Summary of Search Speed Optimization

The contents of this chapter can be summarized by comparing the performances of various approaches along two axes: word error rate *vs* recognition speed. Figure 4.15 captures this information succinctly.

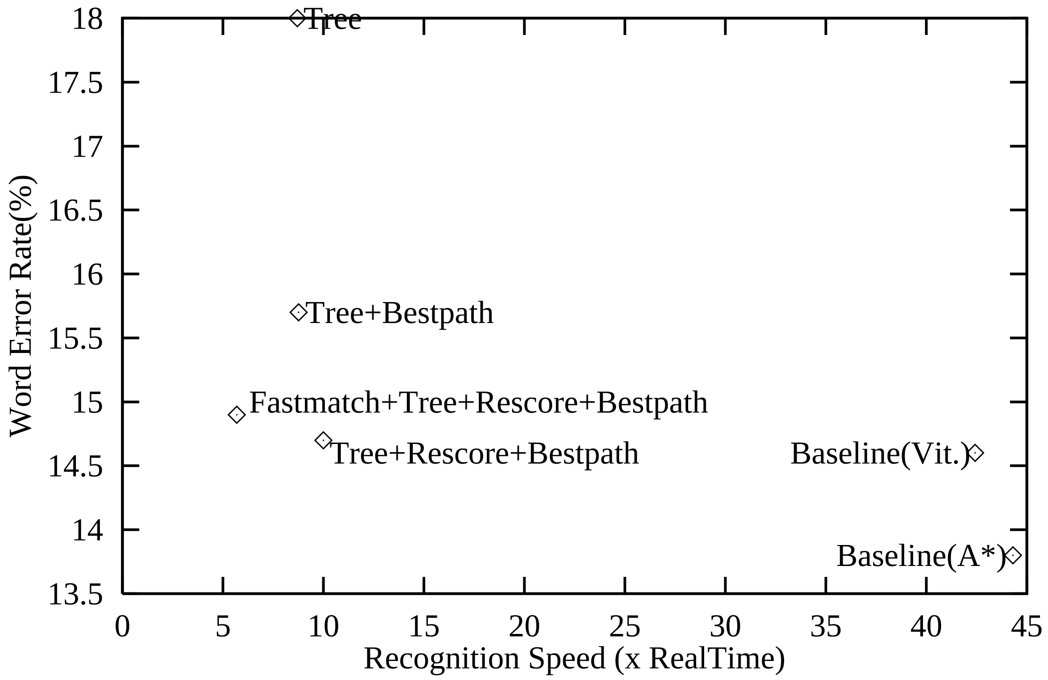
The figure shows that we can very nearly attain the recognition accuracy of the baseline system while speeding up the system by a factor of about 8. It also brings out the relative contributions and costs of each technique. Appendix B contains a summary of the results of significance tests on the differences between the recognition accuracies of the various systems.

We briefly review the results from this chapter:

- The lexical tree search algorithm is about 5 times faster than the baseline Sphinx-II system. The search component alone, excluding the computation of acoustic output probabilities of senones, is over 7 times faster than the baseline system.



(*a*) *20K* Task.



(*b*) *58K* Task.

Figure 4.15: Word Error Rate *vs* Recognition Speed of Various Systems.

- During the lexical tree search, deferring the computation of language model probabilities until the leaves of the lexical reduces the cost of such accesses by about an order of magnitude compared to the baseline system. It also leads to an optimization whereby about half of these computations can be eliminated by reusing results from an earlier time.

- The lattice error rate of the word lattice produced by the lexical tree search is about 2%, and is highly compact, making it ideal for postprocessing steps using more sophisticated searches and models. The number of entries in the word lattice for a 10sec sentence is typically about 1000. The word lattice size and error rate is comparable to that of the baseline system.

- Even though the lexical tree search suffers an increase in word error rate of about 20% relative to the baseline system, the loss can be largely recovered by searching the word lattice for a globally optimum path. The resulting word error rate is within 7% relative to the baseline system. The computational cost of the best path search is negligible.

- The compact word lattice produced by the tree search can be efficiently postprocessed using detailed acoustic models and/or search algorithms. By applying the forward Viterbi search algorithm of the *baseline system* to the word lattice, followed by the best path search, we equal the recognition accuracy of the baseline system. The overall increase in computation with the addition of the postprocessing step is about 15%.

- The phonetic fast match heuristic improves recognition speed by identifying a limited set of candidate phones to be activated at each frame. Incorporating this step into the lexical tree search leads to a further speedup in overall execution time by a factor of about 1.8 with less than 2% relative increase in word error rate.

Based on our experiences reported in this chapter, we believe that a practical organization for the decoder is as shown in Figure 4.16.

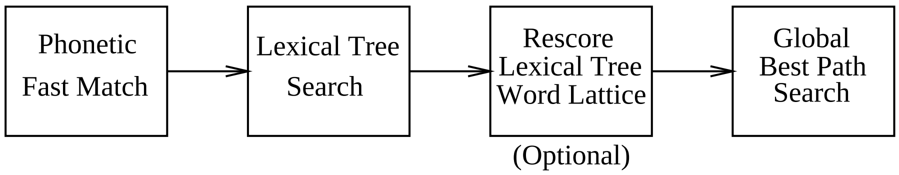


Figure 4.16: Configuration of a Practical Speech Recognition System.

Each of the blocks in the linear pipeline operates in the forward direction in time, and hence the entire configuration can be overlapped to a large extent, avoiding delays that would be inherent if any of the blocks involved a backward pass, for instance.

The lexical tree search module, using fairly detailed acoustic and grammar models but simple enough to allow near real-time operation, produces a compact word lattice with a low lattice error rate. It is the key to the overall organization of a practical speech recognition system.

Additional improvements in speed can be obtained by exploiting parallelism between and within the individual blocks in Figure 4.16. The pipelined organization lends itself naturally to a parallel implementation, operating asynchronously in a data-driven fashion. There is also a large amount of parallelism within some of the modules. For example, the evaluation of acoustic models and the HMM network search can be performed in parallel, with very simple communication between them. One of the early work in this area was in parallelizing the forward Viterbi pass of the baseline Sphinx-II system on a 5-node shared-memory multiprocessor [14, 13] on the 1000-word Resource Management task [51], which yielded a speedup of about 3.8 [54]. Parallelizing the lexical tree search is a little different, but the potential exists, nevertheless.

Chapter 5

Memory Size Reduction

The second important computational resource needed by modern speech recognition systems, after CPU power, is main memory size. Most research systems require hundreds of megabytes of main memory that are only found on high-end workstations. Clearly, practical applications of speech recognition must be able to run on much smaller memory configurations.

The work reported in this chapter is once again in the context of the baseline Sphinx-II system described in Chapter 3. It is typical of most other research systems in use in the speech community. The two main candidates for memory usage in the baseline Sphinx-II system are the acoustic and language models. The former is dominated by the senone mixture weights (see Sections 2.1.2, 3.1.1, and 3.4.4). The latter primarily consists of the bigram and trigram data structures described in Section 3.4.4, Figure 3.8.

In this chapter we describe the approaches taken to reduce their sizes in the lexical tree decoder. In Section 5.1 we discuss the reduction of acoustic model size and in Section 5.2 that of the language model. Some of the techniques presented here have also been incorporated into the baseline decoder with similar results.

5.1 Senone Mixture Weights Compression

The main hypothesis in designing a scheme for reducing the memory size requirement of acoustic models is that the exact state score (or senone score) in each frame is not as important to the detailed search as is the relative ranking of the models. Furthermore, short-term fine distinctions in the scores of states in a given frame are misleading, because of the inherent uncertainty in the statistical models underlying those states. These observations lead to the simple solution of *clustering* acoustic modelling values into a small number of bins which can be indirectly addressed using a small number of bits.

This approach is applied to the Sphinx-II semi-continuous acoustic models in the case of the senone weights. As we saw in Section 3.4.4, the senone mixture weights or coefficients account for about 40MB of memory usage, where each coefficient is a 4-byte value. These coefficients are reduced to 1 byte by simply truncating their least significant bits until the entire dynamic range can be fit into 8 bits. Thus, we obtain a 4-fold reduction in their memory requirement.

Most benchmark evaluations described in this thesis have been carried out with this memory organization. The impact of this truncation on recognition accuracy is virtually non-existent. More important, only about 1% of the *individual sentences* have different recognition results in the two cases. In other words, the reduction in precision makes little difference not just on average, but even in detail. In fact, the overall recognition accuracy is actually slightly better after the truncation.

There is also an execution speed benefit to this compaction. Equation 2.1 in Section 2.1.2 defines the expression for the output probability of a senone in a given frame. In Sphinx-II, as in many other speech recognition systems, all probability calculations are carried out in log-space, so that multiplication operations can be reduced to additions in log-space. But the *summation* term in equation 2.1 poses a difficulty. By truncating all log-probability values to 8-bits, however, the addition operation can be achieved in log-space by simply implementing it as a table-lookup operation. This optimization reduces the execution time for senone output probability computation by a factor of 2.

5.2 Disk-Based Language Models

In the case of the language model, a totally different approach is necessary. It is indeed feasible to reduce the probability values and backoff weights to 8-bit values without any effect on recognition accuracy. But in this case, the probability values have already been compacted to 16-bits as mentioned in Section 3.4.4. Hence, the payoff in reducing them to 8 bits is less. Furthermore, there are other fields in each bigram and trigram, such as the *word-id*, which are much harder to compress further.

The observation in this case is that in decoding any given utterance, only a very small portion of the language model is actually used. Hence, we can consider maintaining the language model entirely on disk, and retrieving only the necessary pieces, on demand. One would expect the virtual memory system to effectively accomplish this for free. But, as we noted in Section 3.4.4, the granularity of access to the bigram and trigram data structures is much smaller than a physical page size on most modern workstations. For example, the average number of bigrams per word in the case of the *58K* vocabulary is about 105. The average number of trigrams per word pair is at least an order of magnitude smaller. Hence, the virtual memory system is relatively ineffective in managing the working set for these data structures.

It is possible to maintain the language model on disk and explicitly load the necessary bigrams and trigrams on demand. However, to avoid excessive delays due to disk access, we must resort to some caching strategy. For this, we observe that if we are required to compute a bigram probability $P(w_j|w_i)$ during some frame, we are very likely to have to compute other bigram probabilities $P(w_k|w_i), k \neq j$, in the same frame. We can make a similar case for trigram probabilities. The reason for this phenomenon is that typically several words arrive at their ends in a given frame, and we need to compute each of their language model probabilities with respect to each of some set of predecessor words.

The caching policy implemented in our system is quite straightforward:

- Since unigrams are a small portion of the large vocabulary language models, they are always kept in memory. Only bigrams and trigrams are read from disk, and they are cached in logical chunks. That is, all bigram followers of a word or all trigram followers of a word pair are cached at a time.
- Whenever a bigram probability $P(w_j|w_i)$ is needed, and it is not in memory, all bigram followers of w_i are read from disk and cached into memory.
- Likewise, whenever a trigram probability $P(w_k|w_i, w_j)$ is needed, and it is not in memory, all trigram followers of w_i, w_j are read and cached in memory. Furthermore, all bigram followers of w_i are also read from disk and cached in memory, if not already there.
- To avoid a continual growth in the amount of cached language model data, it is necessary to *garbage collect* them periodically. Since the decoder processes input speech one sentence at a time (which are at the most a few tens of seconds long), the cached data are flushed as follows. At the end of each utterance, we simply free the memory space for those cached entries which were not accessed during the utterance. This ensures that we recycle the memory occupied by relatively rare words, but retain copies of frequently occurring words such as function words (`A`, `THE`, `IS`, etc.). The "cache lines" associated with the functions words are also relatively large, and hence more expensive to read from disk.

All the benchmark evaluations with the lexical tree decoder have been carried out using the disk-based language model with the above caching scheme. We have measured the number of bigrams and trigrams resident in memory during all of our benchmark evaluations. In both the *20K* and *58K* tasks, only about 15-25% of bigrams and about 2-5% of all trigrams are resident in memory on average, depending on the utterance.

The impact of the disk-based language model on elapsed time is minimal, implying that the caching scheme is highly effective in avoiding disk access latency. Measuring elapsed time is tricky because it is affected by factors beyond our control, such as other processes contending for the CPU, and network delays. However, on at least some experimental runs on entire benchmark test sets, the difference between the CPU usage and elapsed time is no more than if the entire language model is completely memory-resident. The elapsed time exceeds the CPU usage by less than 5%, and it is the existence proof for the effectiveness of the caching policy.

5.3 Summary of Experiments on Memory Size

We have presented two simple schemes for reducing the memory sizes of large acoustic and language models, respectively. The former is compressed by simply truncating the representation from 32 to 8 bits. This granularity appears to be quite sufficient in terms of retaining the original recognition accuracy, sentence for sentence.

The memory-resident size of the language model is reduced by maintaining it on disk, with an effective caching policy for eliminating disk access latency. The fraction of bigrams in memory is reduced by a factor of between 4 and 6, and that of trigrams by a factor of almost 20-50. Clearly, there is no question of loss of recognition accuracy, since there is no change in the data representation.

We believe that several other implementations use 8-bit representations for acoustic models, although the literature hasn't discussed their effect on recognition, to our knowledge. We do not know of any work dealing with the language model in the way described here. Other approaches for reducing the size of language models include various forms of clustering, for example into *class*-based language models, and eliminating potentially useless bigrams and trigrams from the model [58]. These approaches generally suffer from some loss of accuracy. The advantage of simple word bigram and trigram grammars is that they are easy to generate from large volumes of training data.

Chapter 6

Small Vocabulary Systems

6.1 General Issues

Although we have mainly concentrated on large vocabulary speech recognition in this thesis, it is interesting to consider how the techniques developed here extend to smaller vocabulary tasks. There are two cases: tiny vocabulary tasks of a few tens to a hundred words, and medium vocabulary of a few thousands of words. The former are typical of command and control type applications with highly constrained vocabularies. The latter are representative of applications in constrained domains, such as queries to a financial database system.

For a really small vocabulary of a few tens of words, search complexity is not a major issue. In a Viterbi beam search implementation, at most a few hundred HMMs may be active during each frame. Similarly there may be at most a few hundred cross-word transitions. (These are the two most prominent subcomponents of search in large vocabulary recognition, and our main focus.) The acoustic output probability computation, and questions of recognition accuracy are much more dominant issues in such small vocabulary tasks. The nature of the problem also allows the use of *ad hoc* techniques, such as word HMM models for improving recognition accuracy. The wide range of options in the extremely small vocabulary domain makes the efficiency measures for large vocabulary recognition less relevant.

Secondly, as the vocabulary size decreases, words tend to have fewer common pronunciation prefixes. For example, the triphone lexical tree for the digits lexicon in Section 3.1.2 is not a tree at all. It is completely flat. The tree structure is largely irrelevant in such cases. We do not consider extremely small vocabulary tasks any further.

Even when the vocabulary is increased to a few thousands of words, the cost of search does not dominate as in the case of large vocabulary tasks. The computation of HMM state output probabilities is about equally costly. Hence, even an infinite speedup of the search algorithm produces an overall speedup of only a small factor.

In this chapter we compare the baseline Sphinx-II system and the lexical tree search system on the speech recognition component of the ATIS (Airline Travel Information Service) task[1] [45, 46], which has a vocabulary of about 3K words.

Section 6.2 contains the details of the performance of the baseline Sphinx-II system and the lexical tree decoder on the ATIS task.

6.2 Performance on ATIS

6.2.1 Baseline System Performance

ATIS is a small-vocabulary task in which the utterances are mainly queries to an air travel database regarding flight and other travel-related information. The test conditions for the ATIS task are as follows:

- 3000 word vocabulary, including several *compound* words, such as `I_WANT`, `WHAT_IS`, etc.
- 10,000 senone semi-continuous acoustic models trained for ATIS from both the large vocabulary *Wall Street Journal* and ATIS data.
- Word bigram language model, with about 560,000 bigrams. It is a fairly constrained grammar with a much lower perplexity than the large vocabulary ones.
- Test set consisting of 1001 sentences, 9075 words.

The style of speaking in ATIS is a little more conversational than in the large vocabulary test sets from the previous chapters.

The performance of the baseline Sphinx-II system on the ATIS task is summarized in Table 6.1. The noteworthy aspects of this test are the following (we focus mainly on the forward Viterbi pass):

- The cost of output probability computation (VQ and senone evaluation) is almost half of the total execution time of the forward Viterbi search. Hence speeding up the other half, i.e. search alone, by an order of magnitude has much less impact on the overall speed.
- The number of HMMs evaluated per frame is still sufficiently large that it is not worth while computing the senone output probabilities on demand. It is less expensive to compute all of them in each frame.

[1]The overall ATIS task has other components to it, such as natural language understanding of the spoken sentence. We ignore them in this discussion.

	Fwd. Vit.	Fwd/Bwd/A*
%Word Error Rate	5.11%	4.95 %
x RealTime	8.66	10.73

(a) Word Error Rates, Execution Times (x RealTime).

	VQ+Senone Evaluation	HMM Evaluation	Word Transition
x RealTime	4.06	2.88	1.54
%Forward Pass	46.9%	33.3%	17.8%

(b) Breakdown of Forward Pass Execution Times.

Table 6.1: Baseline System Performance on ATIS.

- Cross word transitions are only half as computationally costly as HMM evaluation, whereas in the large vocabulary tasks they are about evenly balanced (Table 3.4). Part of the reason is that the ATIS language model provides stronger constraints on word sequences, so that fewer cross-word transitions have to be attempted. The pruning behaviour of the language model is significant in this case.

6.2.2 Performance of Lexical Tree Based System

In this section we evaluate the lexical tree search and the associated postprocessing steps on the ATIS task. First of all, we compare the number of root nodes in the triphone lexical tree in the ATIS task with the other large vocabulary tasks. Table 6.2 shows these figures. (There are multiple alternative pronunciations for certain words, which increase the total number of lexical entries over the vocabulary size.) Clearly,

	ATIS	*20K* task	*58K* task
No. words	3200	21500	61000
No. root HMMs	450	650	850
Ratio(%) (root HMMs/words)	14.1	3.0	1.4

Table 6.2: Ratio of Number of Root HMMs in Lexical Tree and Words in Lexicon (approximate).

the degree of sharing at the root of the tree structure decreases as the vocabulary size decreases. Hence, we may expect that the lexical tree structure will give a smaller improvement in recognition speed compared to large vocabulary situations.

Recognition Speed

The recognition speed for various configurations of the experimental decoder is reported in Table 6.3. The phonetic fast match heuristic in this case is based on context-independent HMM state scores. (We have observed in Section 4.5.4 that it is advantageous to do so for smaller vocabulary systems.) As expected, the overall speedup is less compared to the large vocabulary case (Figure 4.15).

	T	TB	TRB	FTRB
x RealTime	2.50	2.55	2.89	1.57
Speedup Over Baseline Viterbi	3.46	3.40	3.00	5.52

(T=Tree Search, B=Bestpath, R=Rescoring, F=Phonetic Fastmatch)

Table 6.3: Execution Times on ATIS.

The main reason for the limited speedup over the baseline system is clearly the cost of acoustic probability computation, which is nearly 50% of the total in the baseline system. The tree search algorithm is primarily aimed at reducing the cost of the search component, and not at the acoustic model evaluation. Secondly, as noted earlier in Table 6.2, there is less sharing in the ATIS lexical tree than in the larger vocabulary tasks. Hence, even the potential speedup in search is limited.

Let us first consider the situation in further detail without the phonetic fast match heuristic. Table 6.4 summarizes the breakdown of the lexical tree search execution time on the ATIS task when the fast match step is not employed. Comparing these

	VQ+Senone Evaluation	HMM Evaluation	Leaf Node Transition	Word Transition
x RealTime	1.30	0.72	0.39	0.08
%Tree Search	52.0	28.8	15.6	3.2

Table 6.4: Breakdown of Tree Search Execution Times on ATIS (Without Phonetic Fast Match).

figures to Table 4.3 for the large vocabulary case, we see the relative dominance of the cost of acoustic model evaluation.

In spite of the large cost of acoustic model evaluation, we are still able to obtain a speedup of over a factor of 3 for two reasons. First, the fewer number of active HMMs in the tree search, compared to the baseline system, allows us to scan for and evaluate just the active senones, instead of all of them. Second, the reduction in the precision of senone mixture weights allows us to implement some of the output probability computation with simple table-lookup operations, as discussed in Section 5.1. Hence, the larger than expected gain in speedup is only partly owing to improvement in search.

The phonetic fast match heuristic, however, is as effective as before. When the heuristic is included in the lexical tree search, it reduces the number of HMMs evaluated. Since it is based only on context-independent state scores, the reduction in the number of active HMMs also translates to a reduction in the number of active context-dependent senones. In other words, this technique helps both the search and acoustic model evaluation components. As a result, we are able to further reduce the total execution time by about 45%.

Recognition Accuracy

Table 6.5 lists the recognition accuracy of various configurations of the experimental decoder. As noted earlier, the phonetic fast match heuristic is based on context-independent HMM state scores. As in the case of the large vocabulary tasks, the

	T	TB	TRB	FTRB
%Word Error Rate	6.31	5.70	5.27	5.34
%Degradation w.r.t. Baseline Viterbi	23.5	11.5	3.1	4.5
%Degradation w.r.t. Baseline A*	27.5	17.2	6.5	7.9

(T=Tree Search, B=Bestpath, R=Rescoring, F=Phonetic Fastmatch)

Table 6.5: Recognition Accuracy on ATIS.

rescoring and global best path searches are able to recover the loss in accuracy from the lexical tree search. There is still a resultant loss of about 3-5% relative to the forward search of the baseline system. It is probably due to a larger degree of pruning errors in the tree search. Since the lexical tree is searched without the benefit of prior language model probabilities, a word must survive acoustic mismatches until the leaf node. There is a greater likelihood of poor acoustic matches in the ATIS task because of its more conversational nature and associated fluent speech phenomena.

Memory Size

As in the case of large vocabulary systems, truncating senone mixture weights down to 8 bit representation has no effect on recognition accuracy. The language model size, while not a major issue at 560K bigrams, is also significantly reduced by the disk-based implementation outlined in Section 5.2. The average number of bigrams in memory is reduced to about 10% of the total. Once again, the caching strategy effectively eliminates the cost of disk access latency.

6.3 Small Vocabulary Systems Summary

Recognition speed and memory size are typically not a serious issue in the case of extremely small vocabulary tasks of a few tens or hundreds of words. They are dominated by concerns of modelling for high accuracy. The lexical tree structure is entirely irrelevant because of the limited amount of sharing in the pronunciation of individual words.

In the case of medium vocabulary tasks consisting of a few thousand words, the cost of search does become an issue, but the cost of acoustic model evaluation is an equally important concern.

The techniques described in this thesis are together able to improve the speed of recognition on the 3K word ATIS task by a factor of about 5.5, with a 4.5% relative increase in word error rate over the forward Viterbi result of the baseline Sphinx-II system. The speedup in search is limited by the relative dominance of the acoustic output probability computation in the small vocabulary environment. Furthermore, the lesser degree of sharing in the lexical tree structure reduces the effectiveness of the tree search algorithm. Hence, it is also necessary to speed up the acoustic model evaluation.

The smaller number of active HMMs per frame allows us to compute only the active senones per frame. This is not useful in the baseline system as the reduction in computing active senones is offset by the need to scan a large number of active HMMs to determine the set of active senones. Clearly, there is a tradeoff, depending on the relative costs of the two operations.

The phonetic fast match heuristic is able to provide a speedup comparable to that in the large vocabulary situation, demonstrating the effectiveness and power of the heuristic. When it is based on context-independent state scores, it is effective in reducing both the search and context-dependent acoustic model evaluation times.

Chapter 7

Conclusion

This thesis work has focussed on the problems relating to the computational efficiency of large vocabulary, continuous speech recognition. The foremost concern addressed in this thesis is that of dealing with the large search space associated with this task. This space is so large that brute force approaches can be several orders of magnitude slower than real-time. The beam search heuristic is able to narrow down the search space dramatically by searching only the most likely states at any time. However, searching even this reduced space requires several tens of times real time on current computers. A reduction in the computational load must come from algorithmic and heuristic improvements.

The second issue addressed in this thesis is efficiency of memory usage. In particular, the acoustic and language models are the largest contributors to memory size in modern speech recognition systems.

In order to translate the gains made by research systems in recognition accuracy into practical use, it is necessary to improve the computational and memory efficiency of speech recognition systems. It is relatively easy to improve recognition speed and reduce memory requirements while trading away some accuracy, for example by using tighter beamwidths for most drastic pruning, and by using simpler or more constrained acoustic and language models. But it is much harder to improve both the recognition speed and reduce main memory requirements while preserving the original accuracy.

The main contributions of this thesis are an 8-fold speedup and a 4-fold reduction in the memory size of the CMU Sphinx-II system. We have used the Sphinx-II system as a baseline for comparison purposes since it has been extensively used in the yearly ARPA evaluations. It is also one of the premier systems, in terms of recognition accuracy, of its kind. On large vocabulary tasks the system requires several tens of times real time and 100-200MB of main memory to perform its function. The experiments have been performend on several commonly used benchmark test sets and two different vocabulary sizes of about *20K* and *58K* words.

We first present a brief description of the lessons learnt from this work, followed by a summary of its contributions, concluding with some directions for future work based on this research.

7.1 Summary of Results

The results in this thesis are based on benchmark tests carried out on the *Wall Street Journal* and *North American Business News* development and test sets from the Nov. 1993 and Nov. 1994 ARPA evaluations. They consist of read speech in a clean environment using high quality, close-talking microphones, and are widely used in the ARPA speech community. The experiments are carried out using two different vocabulary sizes, of 20,000 words and 58,000 words, respectively. We now summarize the main results from this thesis below.

- The lexical tree search algorithm is about 5 times faster than the baseline Sphinx-II system. The search component alone, excluding the computation of acoustic output probabilities of senones, is over 7 times faster than the baseline system.

- During the lexical tree search, deferring the computation of language model probabilities until the leaves of the lexical reduces the cost of such accesses by about an order of magnitude, compared to the baseline system. It also leads to an optimization whereby about half of these computations in a given frame can be eliminated by reusing results from an earlier frame.

- The lattice error rate of the word lattice produced by the lexical tree search is about 2% (excluding out of vocabulary words), and is highly compact. This makes it an ideal input for postprocessing steps using more detailed models and search algorithms. The number of entries in the word lattice for a 10sec sentence is typically about 1000. The word lattice size and error rate are comparable to that of the baseline system.

- Even though the lexical tree search suffers an increase in word error rate of about 20% relative to the baseline system, the loss can be largely recovered from the word lattice. The best path algorithm presented in this thesis finds a globally optimum path through the word lattice and brings the word error rate down to within 7% relative to the baseline system. The computational cost of the best path search is negligible.

- The compact word lattice produced by the tree search can be efficiently postprocessed using detailed acoustic models and/or search algorithms. By applying the forward Viterbi search algorithm of the *baseline system* to the word lattice, followed by the best path search, we equal the recognition accuracy of the baseline system. The overall increase in computation with the addition of the postprocessing step is about 15%.

- The phonetic fast match heuristic improves recognition speed by identifying a limited set of candidate phones to be activated at each frame. Incorporating this step into the lexical tree search leads to a further speedup in overall execution time by a factor of about 1.8 with less than 2% relative increase in word error rate.

- It is possible to reduce the precision of representation of the statistical databases, and thus reduce their memory requirement, with no significant effect on recognition accuracy. Thus, a reduction in the precision of senone weight values from 32 bits to 8 bits reduces the acoustic model size from about 40MB to about 10MB. This result has also been observed by several other sites such as IBM, BBN and Microsoft.

- The compact representation of senone weights in 8 bits enables the computation of senone output probabilities to be implemented by simple table look up operations. This speeds up the computation by about a factor of 2.

- A disk-based language model, coupled with a simple software caching scheme to load bigrams and trigrams into memory on demand leads to a reduction in its memory requirements by over a factor of 5. The fraction of bigrams resident in memory is reduced to around 15-25% of the total, and that of trigrams to 2-5% of the total number, on average. The caching scheme is effective in neutralizing the cost of disk access latencies. Since there is no change in data representation, there is no loss of accuracy.

In summary, it is possible to reduce the computation time of the Sphinx-II recognizer by nearly an order of magnitude and the memory size requirements by a factor of about 4 for large vocabulary continuous speech recognition, with very little loss of accuracy. Appendix B contains a summary of the results of significance tests on the differences between the recognition accuracies of the various systems.

As an additional benchmark result, we note that the techniques described in this thesis are sufficiently robust that the lexical tree based recognizer was used by CMU during the Nov. 1995 ARPA evaluations.

7.2 Contributions

One of the main contributions of this thesis is in providing a comprehensive account of the design of a high-performance speech recognition system in its various aspects of accuracy, speed, and memory usage. One of the underlying questions concerning research systems that focus mainly on accuracy, and require large amounts of computational power is whether the approaches taken will ever be of practical use. This work suggests that concerns of accuracy and efficiency are indeed separable components of speech recognition technology, and lends validity to the ongoing effort in improving accuracy.

The second major contribution of the thesis is the presentation an overall organization of a speech recognition system for large vocabulary, continuous speech recognition. We contend that a fast but detailed search, such as that provided by the lexical tree search described in this thesis, is the key step in obtaining a highly compact and accurate word lattice. The lattice can be searched using more detailed and sophisticated models and search algorithms efficiently. The use of multi-pass systems is not new. Most current speech recognition systems are of that nature [41, 65, 5, 15, 19, 38]. Many reports cite the savings to be had by postprocessing a word lattice [65, 38] instead of the entire vocabulary. However, the task of actually producing such lattices *efficiently* has been relatively unexplored. This is necessary for the practical application of accurate, large vocabulary speech recognition and it is addressed in this thesis.

Technically, the thesis presents several other contributions:

- The design and implementation of search using lexical trees. It analyzes the technique of deferring the application of language model probabilities until the leaves of the lexical tree in an efficient way, and reduces the cost of computing these probabilities by an order of magnitude. Other tree-based searches [39, 43, 65, 66, 40] attempt to overcome the problem by creating bigram copies of the search tree. This has three problems: the search space is increased, the cost of language model access is still substantial, and the physical memory size requirements also increase. Though the lexical tree search presented in this thesis suffers some loss of accuracy, the loss is recovered by postprocessing the word lattice output, which can be done efficiently since the lattice is highly compact.

- A best path search for global optimization of word lattices. This technique searches the word graph formed from word segmentations and scores produced by an earlier pass. It finds a globally optimum path through the graph, which can be accomplished by any textbook shortest path algorithm. When applied to the word lattice produced by the lexical tree search, it brings the final accuracy much closer to that of the baseline system, largely overcoming the degradation in accuracy incurred by the lexical tree search. Furthermore, it operates in a small fraction of real time and its cost is negligible. Word lattice searches have been proposed in [65, 39], for example, but they are directed more towards using the lattice for driving later search passes with more detailed models.

- Use of HMM state output probabilities as a fast match to produce candidate list of active phones to be searched. This heuristic has the potential to reduce the active search space substantially, depending on the sharpness of the underlying acoustic models. We have tried two different approaches for determining the set of active phones: based on all HMM states, and based on only the context-independent states. The former leads to better phone prediction and more effective search pruning, but incurs a fixed cost of evaluating all HMM state probabilities. The latter does not have this limitation but is somewhat less accurate in predicting active phones. It is more appropriate for smaller vocabularies since senone evaluation does not become a bottleneck.

- Precision of representation of statistical models is largely irrelevant for recognition accuracy. Substantial savings in memory space can be obtained by quantizing, clustering, or truncating probability values into few bits. When probability values are represented in this compact fashion, it is sometimes possible to implement complex operations on them by means of simple table-lookup operations. The space reduction is not specific to the lexical tree search algorithm. It has been implemented in the baseline Sphinx-II system as well, with similar results.

- Use of disk-based language models in reducing memory requirements. Bigrams and trigrams are read into memory on demand, but a simple software caching policy effectively hides long disk access latencies. The technique is not specific to the lexical tree search algorithm. It has been implemented in the baseline Sphinx-II system as well, and has proven to be as effective.

7.3 Future Work on Efficient Speech Recognition

The efficiency of speech recognition is ultimately judged by the end application. Transforming today's laboratory versions of speech recognition systems into practical applications requires solutions to many other problems. The resource requirements of current systems—CPU power and memory—are still beyond the capabilities of commonly available platforms for medium and large vocabulary applications. Furthermore, the notion of "performance" extends beyond accuracy, speed and memory size. These factors include, for example, robustness in the presence of noise, adaptation to varying speaking styles and accents, design issues dealing with human-computer interfaces that are appropriate for speech-based interaction, speech understanding, etc. We consider how research on the following might be useful in this respect.

Combining Word-Level Fast Match With Lexical Tree Search

The speed of recognition systems could be improved by a combination of a word-level fast match algorithm and a lexical tree search. The former typically reduces the number of candidate words to be searched by at least an order of magnitude.

Furthermore, the candidates produced at a given instant are likely to be phonetically closely related, which can be exploited effectively by the lexical tree search algorithm. We do not know of any detailed report on a system that takes advantage of the two together. It is worthwhile studying this problem further. The results presented in this thesis provide a baseline for comparison purposes.

Robustness to Noise Using Phone Lattices

Normal speech is often interrupted or overshadowed by noise, and recognition during such periods is highly unreliable. The need for confidence measures attached to recognition results has often been felt. The phone lattices produced by the phonetic fast match heuristic described in Section 4.5 could be adapted for this purpose. During clean speech, there is a distinct separation between the leading active phones and the remaining inactive ones. Few phones fall within the beam and they are generally closely related or readily confusable. Also there is a good correlation between neighbouring frames. In the presence of noise, especially non-speech noise, the number of active phones within the beamwidth increases significantly. The reason is that the underlying acoustic models are unable to classify the noisy speech with any confidence. There is a much greater degree of confusability within a frame, and little correlation among active phones between neighbouring frames. One could use these measures to detect regions where the recognition is potentially unreliable. The advantage of this approach is that it is inexpensive to compute.

Postprocessing Word Lattices Using Multiple Language Models

This technique pertains to good user interface design. Several practical applications of speech recognition deal with the handling of a number of well-defined but distinct tasks. For example, in a dictation task text input through speech might be interspersed with spoken commands to manipulate the document being edited, such as "scroll-up", "previous paragraph", etc. However, it is cumbersome for the user to constantly have to indicate to the system the type of command or speech that is forthcoming. It is desirable for the system to make that decision after the fact. Such a task can be accomplished by searching the word lattice output of the lexical tree search using multiple language models. The dictation task would use two language models, a general purpose one, and a restricted one for editing commands. Initial speech recognition is always carried out using the general purpose model. Once a sentence is recognized and the word lattice is built, it can be searched again, this time using the constrained language model. If a path through the word lattice is found, the sentence can be interpreted as an editing command. The compact nature of the word lattice allows such searches to be carried out rapidly.

Rescoring Word Lattices Using Prosodic Models

Word lattices often contain alternative paths that are phonetically identical. Rescoring these using conventional acoustic models, however detailed, is unlikely to resolve them. However, word or context-dependent prosodic models applied to longer units such as syllables can help discriminate between the alternatives. Once again, the compact nature of the lattices enables such postprocessing to be performed efficiently.

Disk-Based Acoustic Models

With the emergence of continuous HMMs for acoustic modelling, demands on CPU power and memory capacities increase. The availability of compact lattices that can be rescored with detailed acoustic models alleviates the CPU power problem. It also implies that fewer HMM states are active at any given instant. Furthermore, once a state becomes active, it remains active for a few more frames. Thus, it may be possible to use disk-based acoustic models to reduce their memory requirements as well. One needs to explore caching mechanisms and their effectiveness in overcoming disk access latencies for this purpose.

Design of Scalable Systems

It is important to consider the problem of designing a speech application on a specific platform with given resource constraints. The main criteria are that the application should perform in real time within the CPU and memory capacities of the system. Therefore, it is generally necessary to make trade-offs regarding the level of sophistication of modelling and search that may be employed. The memory size problem can be attacked by using less detailed models during the initial search to create a word lattice, as well as using disk-based mechanisms. The cost of search can be reduced by increasing the degree of sharing in the lexical tree, for example, by using diphone or even monophone models instead of triphones. The word lattice output may have to be larger in order to provide an acceptably low lattice error rate, and hence the postprocessing costs also increase. It is worth investigating the tradeoffs that are possible in this respect.

Appendix A

The Sphinx-II Phone Set

Phone	*Example*	*Phone*	*Example*	*Phone*	*Example*
AA	*o*dd	EY	*a*te	P	*p*ee
AE	*a*t	F	*f*ee	PD	li*p*
AH	h*u*t	G	*g*reen	R	*r*ead
AO	*ou*ght	GD	ba*g*	S	*s*ea
AW	c*ow*	HH	*h*e	SH	*sh*e
AX	*a*bide	IH	*i*t	T	*t*ea
AXR	us*er*	IX	ac*i*d	TD	li*t*
AY	h*i*de	IY	*ea*t	TH	*th*eta
B	*b*e	JH	*g*ee	TS	bi*ts*
BD	du*b*	K	*k*ey	UH	h*oo*d
CH	*ch*eese	KD	li*ck*	UW	t*wo*
D	*d*ee	L	*l*ee	V	*v*ee
DD	du*d*	M	*m*e	W	*w*e
DH	*th*ee	N	*kn*ee	Y	*y*ield
DX	ma*tt*er	NG	pi*ng*	Z	*z*ee
EH	*E*d	OW	*oa*t	ZH	sei*z*ure
ER	h*ur*t	OY	t*oy*		

Table A.1: The Sphinx-II Phone Set.

Appendix B

Statistical Significance Tests

We have conducted statistical significance tests using the scoring and stats packages from NIST. They were run on recognition results from all test sets put together. We have reproduced the results from these significance tests on the *20K* and *58K* tasks below[1]. In these tables, five different systems are identified:

- `f6p1.m`: Baseline system; forward Viterbi pass.
- `f6p3.m`: Baseline system; all three passes.
- `f8p1.m`: Lexical tree search.
- `f8p2.m`: Lexical tree search and global best path search.
- `f8p3.m`: Lexical tree, rescoring, and global best path search.

The conclusion from these tests is that the new system with three passes (lexical tree search, rescoring, and best path search) is essentially identical to the baseline Viterbi search in recognition accuracy.

[1]The recognition accuracy results are a little bit better here from the figures reported in the main thesis sections because of small differences in the scoring packages used in the two cases.

B.1 20K Task

```
            COMPARISON MATRIX: FOR THE MATCHED PAIRS TEST
   PECENTAGES ARE MEAN PCT ERROR/SEGMENT.  FIRST # IS LEFT SYSTEM
                        STATS from std_stats
             Minimum Number of Correct Boundary words 2
       |-----------------------------------------------------|
       |        | f6p1.m | f6p3.m | f8p1.m | f8p2.m | f8p3.m |
       |--------+--------+--------+--------+--------+--------|
       | f6p1.m |        | f6p3.m | f6p1.m | f6p1.m |  same  |
       |--------+--------+--------+--------+--------+--------|
       | f6p3.m |        |        | f6p3.m | f6p3.m | f6p3.m |
       |--------+--------+--------+--------+--------+--------|
       | f8p1.m |        |        |        | f8p2.m | f8p3.m |
       |--------+--------+--------+--------+--------+--------|
       | f8p2.m |        |        |        |        | f8p3.m |
       |--------+--------+--------+--------+--------+--------|
```

COMPARISON MATRIX: McNEMAR'S TEST ON CORRECT SENTENCES FOR THE TEST:
STATS from std_stats
For all systems

	f6p1.m(209)	f6p3.m(228)	f8p1.m(157)	f8p2.m(202)	f8p3.m(215)
f6p1.m(209)		D=(19) f6p3.m	D=(52) f6p1.m	D=(7) same	D=(6) same
f6p3.m(228)			D=(71) f6p3.m	D=(26) f6p3.m	D=(13) same
f8p1.m(157)				D=(45) f8p2.m	D=(58) f8p3.m
f8p2.m(202)					D=(13) same

Comparison Matrix for the Sign Test
Using the Speaker Word Accuracy Rate (%) Percentage per Speaker
as the Comparison Metric

	f6p1.m	f6p3.m	f8p1.m	f8p2.m	f8p3.m
f6p1.m		f6p3.m	f6p1.m	f6p1.m	same
f6p3.m			f6p3.m	f6p3.m	f6p3.m
f8p1.m				f8p2.m	f8p3.m
f8p2.m					f8p3.m

Comparison Matrix for the Wilcoxon Test
Using the Speaker Word Accuracy Rate (%) Percentage per Speaker
as the Comparison Metric

	f6p1.m	f6p3.m	f8p1.m	f8p2.m	f8p3.m
f6p1.m		f6p3.m	f6p1.m	f6p1.m	same
f6p3.m			f6p3.m	f6p3.m	f6p3.m
f8p1.m				f8p2.m	f8p3.m
f8p2.m					f8p3.m

```
   RANGE ANALYSIS ACROSS RECOGNITION SYSTEMS FOR THE TEST:
                    STATS from std_stats
              by Speaker Word Accuracy Rate (%)
      |-----------------------------------------------|
      |   SYS  |  high  |   low  || std dev |   mean  |
      |--------+--------+--------++---------+---------|
      | f6p3.m |  95.1  |  50.3  ||     8.0 |    84.5 |
      | f6p1.m |  94.9  |  48.5  ||     8.3 |    83.8 |
      | f8p3.m |  95.4  |  46.6  ||     8.5 |    83.8 |
      | f8p2.m |  95.4  |  46.3  ||     8.8 |    82.9 |
      | f8p1.m |  94.1  |  45.1  ||     8.7 |    80.9 |
      |-----------------------------------------------|


|------------------------------------------------------------|
|        |                    PERCENTAGES                    |
|--------+---------------------------------------------------|
|        |0   10   20   30   40   50   60   70   80   90  100|
|   SYS  ||    |    |    |    |    |    |    |    |    |    ||
|--------+---------------------------------------------------|
| f6p3.m |                          ------------+---|---+-   |
| f6p1.m |                         -------------+---|---+-   |
| f8p3.m |                        --------------+---|---+-   |
| f8p2.m |                        -------------+---|----+-   |
| f8p1.m |                       -------------+---|----+--   |
|------------------------------------------------------------|
      | -> shows the mean
      + -> shows plus or minus one standard deviation
```

```
                Composite Report of All Significance Tests
                     For the STATS from std_stats Test

                          Test Name                                 Abbrev.
 ---------------------------------------------------------------    -------
        Matched Pair Sentence Segment (Word Error) Test               MP
 Signed Paired Comparison (Speaker Word Accuracy Rate (%)) Test       SI
   Wilcoxon Signed Rank (Speaker Word Accuracy Rate (%)) Test         WI
                McNemar (Sentence Error) Test                         MN
  |-----------------------------------------------------------------------|
  |           |  f6p1.m   |  f6p3.m   |  f8p1.m   |  f8p2.m   |  f8p3.m   |
  |-----------+-----------+-----------+-----------+-----------+-----------|
  |  f6p1.m   |           | MP f6p3.m | MP f6p1.m | MP f6p1.m | MP   same |
  |           |           | SI f6p3.m | SI f6p1.m | SI f6p1.m | SI   same |
  |           |           | WI f6p3.m | WI f6p1.m | WI f6p1.m | WI   same |
  |           |           | MN f6p3.m | MN f6p1.m | MN   same | MN   same |
  |-----------+-----------+-----------+-----------+-----------+-----------|
  |  f6p3.m   |           |           | MP f6p3.m | MP f6p3.m | MP f6p3.m |
  |           |           |           | SI f6p3.m | SI f6p3.m | SI f6p3.m |
  |           |           |           | WI f6p3.m | WI f6p3.m | WI f6p3.m |
  |           |           |           | MN f6p3.m | MN f6p3.m | MN   same |
  |-----------+-----------+-----------+-----------+-----------+-----------|
  |  f8p1.m   |           |           |           | MP f8p2.m | MP f8p3.m |
  |           |           |           |           | SI f8p2.m | SI f8p3.m |
  |           |           |           |           | WI f8p2.m | WI f8p3.m |
  |           |           |           |           | MN f8p2.m | MN f8p3.m |
  |-----------+-----------+-----------+-----------+-----------+-----------|
  |  f8p2.m   |           |           |           |           | MP f8p3.m |
  |           |           |           |           |           | SI f8p3.m |
  |           |           |           |           |           | WI f8p3.m |
  |           |           |           |           |           | MN   same |
  |-----------+-----------+-----------+-----------+-----------+-----------|
```

B.2 58K Task

```
             COMPARISON MATRIX: FOR THE MATCHED PAIRS TEST
    PECENTAGES ARE MEAN PCT ERROR/SEGMENT.  FIRST # IS LEFT SYSTEM
                          STATS from std_stats
              Minimum Number of Correct Boundary words 2
         |-----------------------------------------------------|
         |        | f6p1.m | f6p3.m | f8p1.m | f8p2.m | f8p3.m |
         |--------+--------+--------+--------+--------+--------|
         | f6p1.m |        | f6p3.m | f6p1.m | f6p1.m |  same  |
         |--------+--------+--------+--------+--------+--------|
         | f6p3.m |        |        | f6p3.m | f6p3.m | f6p3.m |
         |--------+--------+--------+--------+--------+--------|
         | f8p1.m |        |        |        | f8p2.m | f8p3.m |
         |--------+--------+--------+--------+--------+--------|
         | f8p2.m |        |        |        |        | f8p3.m |
         |--------+--------+--------+--------+--------+--------|
```

COMPARISON MATRIX: McNEMAR'S TEST ON CORRECT SENTENCES FOR THE TEST:
STATS from std_stats
For all systems

	f6p1.m(220)	f6p3.m(256)	f8p1.m(155)	f8p2.m(213)	f8p3.m(230)
f6p1.m(220)		D=(36) f6p3.m	D=(65) f6p1.m	D=(7) same	D=(10) same
f6p3.m(256)			D=(101) f6p3.m	D=(43) f6p3.m	D=(26) f6p3.m
f8p1.m(155)				D=(58) f8p2.m	D=(75) f8p3.m
f8p2.m(213)					D=(17) f8p3.m

Comparison Matrix for the Sign Test
Using the Speaker Word Accuracy Rate (%) Percentage per Speaker
as the Comparison Metric

	f6p1.m	f6p3.m	f8p1.m	f8p2.m	f8p3.m
f6p1.m		f6p3.m	f6p1.m	f6p1.m	same
f6p3.m			f6p3.m	f6p3.m	f6p3.m
f8p1.m				f8p2.m	f8p3.m
f8p2.m					f8p3.m

Comparison Matrix for the Wilcoxon Test
Using the Speaker Word Accuracy Rate (%) Percentage per Speaker
as the Comparison Metric

	f6p1.m	f6p3.m	f8p1.m	f8p2.m	f8p3.m
f6p1.m		f6p3.m	f6p1.m	f6p1.m	same
f6p3.m			f6p3.m	f6p3.m	f6p3.m
f8p1.m				f8p2.m	f8p3.m
f8p2.m					f8p3.m

```
RANGE ANALYSIS ACROSS RECOGNITION SYSTEMS FOR THE TEST:
                 STATS from std_stats
          by Speaker Word Accuracy Rate (%)
```

SYS	high	low	std dev	mean
f6p3.m	96.7	41.4	9.1	86.2
f6p1.m	96.2	47.2	8.2	85.5
f8p3.m	96.7	47.2	8.4	85.4
f8p2.m	95.9	43.3	8.9	84.4
f8p1.m	94.4	39.6	9.3	82.2

```
|-------------------------------------------------------------|
|        |                   PERCENTAGES                      |
|--------+----------------------------------------------------|
|        |0    10   20   30   40   50   60   70   80   90  100|
|   SYS  ||    |    |    |    |    |    |    |    |    |    ||
|--------+----------------------------------------------------|
| f6p3.m |                     ------------------+---|----+   |
| f6p1.m |                        ---------------+---|---+-   |
| f8p3.m |                        ---------------+---|---+-   |
| f8p2.m |                      ----------------+---|----+    |
| f8p1.m |                    ----------------+----|----+-    |
|-------------------------------------------------------------|

     | -> shows the mean
     + -> shows plus or minus one standard deviation
```

```
                 Composite Report of All Significance Tests
                      For the STATS from std_stats Test

                           Test Name                                Abbrev.
---------------------------------------------------------------     -------
       Matched Pair Sentence Segment (Word Error) Test                MP
Signed Paired Comparison (Speaker Word Accuracy Rate (%)) Test        SI
  Wilcoxon Signed Rank (Speaker Word Accuracy Rate (%)) Test          WI
                McNemar (Sentence Error) Test                         MN


 |-----------------------------------------------------------------------|
 |           |  f6p1.m   |  f6p3.m   |  f8p1.m   |  f8p2.m   |  f8p3.m   |
 |-----------+-----------+-----------+-----------+-----------+-----------|
 |  f6p1.m   |           | MP f6p3.m | MP f6p1.m | MP f6p1.m | MP   same |
 |           |           | SI f6p3.m | SI f6p1.m | SI f6p1.m | SI   same |
 |           |           | WI f6p3.m | WI f6p1.m | WI f6p1.m | WI   same |
 |           |           | MN f6p3.m | MN f6p1.m | MN   same | MN   same |
 |-----------+-----------+-----------+-----------+-----------+-----------|
 |  f6p3.m   |           |           | MP f6p3.m | MP f6p3.m | MP f6p3.m |
 |           |           |           | SI f6p3.m | SI f6p3.m | SI f6p3.m |
 |           |           |           | WI f6p3.m | WI f6p3.m | WI f6p3.m |
 |           |           |           | MN f6p3.m | MN f6p3.m | MN f6p3.m |
 |-----------+-----------+-----------+-----------+-----------+-----------|
 |  f8p1.m   |           |           |           | MP f8p2.m | MP f8p3.m |
 |           |           |           |           | SI f8p2.m | SI f8p3.m |
 |           |           |           |           | WI f8p2.m | WI f8p3.m |
 |           |           |           |           | MN f8p2.m | MN f8p3.m |
 |-----------+-----------+-----------+-----------+-----------+-----------|
 |  f8p2.m   |           |           |           |           | MP f8p3.m |
 |           |           |           |           |           | SI f8p3.m |
 |           |           |           |           |           | WI f8p3.m |
 |           |           |           |           |           | MN f8p3.m |
 |-----------+-----------+-----------+-----------+-----------+-----------|
```

Bibliography

[1] Alleva, F., Hon, H., Hwang, M., Rosenfeld, R. and Weide, R. *Applying SPHINX-II to the DARPA Wall Street Journal CSR Task.* In **Proceedings of Speech and Natural Language Workshop**, Feb. 1992, pp 393-398.

[2] Alleva, F., Huang, X., and Hwang, M. *An Improved Search Algorithm for Continuous Speech Recognition.* In **IEEE International Conference on Acoustics, Speech, and Signal Processing**, 1993.

[3] Antoniol, G., Brugnara, F., Cettolo, M. and Federico, M. *Language Model Representation for Beam-Search Decoding.* In **IEEE International Conference on Acoustics, Speech, and Signal Processing**, May 1995, pp 588-591.

[4] Bahl, L.R., Bakis, R., Cohen, P.S., Cole, A.G., Jelinek, F., Lewis, B.L. and Mercer, R.L. *Further Results on the Recognition of a Continuously Read Natural Corpus.* In **IEEE International Conference on Acoustics, Speech, and Signal Processing**, Apr. 1980, pp 872-876.

[5] Bahl, L.R., Balakrishnan-Aiyer, S., Franz, M., Gopalakrishnan, P.S., Gopinath, R., Novak, M., Padmanabhan, M. and Roukos, S. *The IBM Large Vocabulary Continuous Speech Recognition System for the ARPA NAB News Task.* In **Proceedings of ARPA Spoken Language System Technology Workshop**, Jan. 1995, pp 121-126.

[6] Bahl, L.R., Brown, P.F., DeSouza, P.V. and Mercer, R.L. *Obtaining Candidate Words by Polling in a Large Vocabulary Speech Recognition System.* In **IEEE International Conference on Acoustics, Speech, and Signal Processing**, 1988.

[7] Bahl, L.R., De Gennaro, V., Gopalakrishnan, P.S. and Mercer, R.L. *A Fast Approximate Acoustic Match for Large Vocabulary Speech Recognition.* **IEEE Transactions on Speech and Audio Processing**, Vol. 1, No. 1, Jan 1993, pp 59-67.

[8] Bahl, L.R., DeSouza, P.V., Gopalakrishnan, P.S., Nahamoo, D. and Picheney, M. *A Fast Match for Continuous Speech Recognition Using Allophonic Models.* In **IEEE International Conference on Acoustics, Speech, and Signal Processing**, 1992, vol.I, pp. I-17 – I-21.

[9] Bahl, L.R., Jelinek, F. and Mercer, R. *A Maximum Likelihood Approach to Continuous Speech Recognition.* In **IEEE Transactions on Pattern Analysis and Machine Intelligence**, Vol. PAMI-5, No. 2, Mar. 1983, pp. 179-190.

[10] Baker, J.K. *The DRAGON System–An Overview.* In **IEEE Transactions on Acoustics, Speech, and Signal Processing**, ASSP-23(1), Feb. 1975, pp. 24-29.

[11] Baum, L.E. *An Inequality and Associated Maximization Technique in Statistical Estimation of Probabilistic Functions of Markov Processes.* **Inequalities** 3:1-8, 1972.

[12] Bellegarda, J. and Nahamoo, D. *Tied Mixture Continuous Parameter Modeling for Speech Recognition.* In **IEEE Transactions on Acoustics, Speech, and Signal Processing**, Dec. 1990, pp 2033-2045.

[13] Bisiani, R. and Ravishankar, M *PLUS: A Distributed Shared-Memory System.* 17th International Symposium on Computer Architecture, May 1990, pp. 115-124.

[14] Bisiani, R. and Ravishankar, M *Design and Implementation of the PLUS Multiprocessor* Internal report, School of Computer Science, Carnegie Mellon University, Pittsburgh, Jan. 1991.

[15] Chase, L., Rosenfeld, R., Hauptmann, A., Ravishankar, M., Thayer, E., Placeway, P., Weide, R. and Lu, C. *Improvements in Language, Lexical, and Phonetic Modeling in Sphinx-II.* In **Proceedings of ARPA Spoken Language Systems Technology Workshop**, Jan. 1995, pp. 60-65.

[16] Cooper, E.C. and Draves, R.P. *C Threads.* Technical Report, School of Computer Science, Carnegie Mellon University, Nov. 1990.

[17] Coremen, T.H., Leiserson, C.E. and Rivest, R.L. **Introduction to Algorithms**, The MIT Press, Cambridge, Massachusetts, 1992.

[18] Gauvain, J.L., Lamel, L.F., Adda, G., and Adda-Decker, M. *The LIMSI Nov93 WSJ System.* In **Proceedings of ARPA Speech and Natural Language Workshop**, Mar. 1994, pp 125-128.

[19] Gauvain, J.L., Lamel, L. and Adda-Decker, M. *Developments in Large Vocabulary Dictation: The LIMSI Nov94 NAB System.* In **Proceedings of ARPA Spoken Language Systems Technology Workshop**, Jan. 1995, pp. 131-138.

[20] Gillick, L.S. and Roth, R. *A Rapid Match Algorithm for Continuous Speech Recognition.* In **Proceedings of DARPA Speech and Natural Language Workshop**, Jun. 1990, pp. 170-172.

[21] Gopalakrishnan, P.S., Bahl, L.R., and Mercer, R.L. *A Tree Search Strategy for Large-Vocabulary Continuous Speech Recognition.* In **IEEE International Conference on Acoustics, Speech, and Signal Processing**, May 1995, pp 572-575.

[22] Gopalakrishnan, P.S., Nahamoo, D., Padmanabhan, M. and Picheny, M.A. *A Channel-Bank-Based Phone Detection Strategy.* In **IEEE International Conference on Acoustics, Speech, and Signal Processing**, April 1994, Vol II, pp 161-164.

[23] Gopinath, R. *et al.* The IBM continuous speech recognition system on demonstration. **ARPA Spoken Language Systems Technology Workshop**, Jan. 1995.

[24] Hauptmann, A. *et al.* The *News-on-Demand* demonstration. In **ARPA Speech Recognition Workshop**, Feb. 1996.

[25] Huang, X., Acero, A., Alleva, F., Beeferman, D., Hwang, M. and Mahajan, M. *From CMU Sphinx-II to Microsoft Whisper–Making Speech Recognition Usable.* In **Automatic Speech and Speaker Recognition–Advanced Topics**, Lee, Paliwal, and Soong, editors, *Kluwer Publishers*, 1994.

[26] Huang, X., Acero, A., Alleva, F., Hwang, M., Jiang, L. and Mahajan, M. *Microsoft Windows Highly Intelligent Speech Recognizer: Whisper.* In **IEEE International Conference on Acoustics, Speech, and Signal Processing**, May 1995, Vol. 1, pp. 93-96.

[27] Hwang, Mei-Yuh. *Subphonetic Acoustic Modeling for Speaker-Independent Continuous Speech Recognition.* Ph.D. thesis, Tech Report No. CMU-CS-93-230, Computer Science Department, Carnegie Mellon University, Dec. 1993.

[28] Hwang, M. and Huang X. *Shared-Distribution Hidden Markov Models for Speech Recognition.* In **IEEE Transactions on Speech and Audio Processing**, Oct. 1993, pp 414-420.

[29] Jelinek, F. *Continuous Speech Recognition by Statistical Methods.* In **Proceedings of the IEEE**, Vol. 64, No. 4, Apr. 1976, pp. 532-556.

[30] Katz, S.M. *Estimation of Probabilities from Sparse Data for the Language Model Component of a Speech Recognizer.* In **IEEE Transactions on Acoustics, Speech and Signal Processing**, vol. ASSP-35, Mar. 87, pp. 400-401.

[31] Kershaw, D.J., Robinson, A.J., and Renals, S.J. *The 1995 ABBOT Hybrid Connectionist-HMM Large-Vocabulary Recognition System.* In **ARPA Speech Recognition Workshop**, Feb. 1996.

[32] Lee, K. *Large Vocabulary Speaker-Independent Continuous Speech Recognition: The SPHINX System.* Ph.D. thesis, Computer Science Department, Carnegie Mellon University, April 1988.

[33] Lee, K. *Context-Dependent Phonetic Hidden Markov Models for Continuous Speech Recognition.* In **IEEE Transactions on Acoustics, Speech, and Signal Processing**, Apr. 1990, pp 599-609.

[34] Lee, K. *Context-Dependent Phonetic Hidden Markov Models for Speaker-Independent Continuous Speech Recognition.* In **Readings in Speech Recognition**, *ed.* Waibel, A. and Lee, K. Morgan Kaufmann Publishers, San Mateo, CA, 1990, pp. 347-365.

[35] Lee, K., Hon, H., and Reddy, R. *An Overview of the SPHINX Speech Recognition System.* In **IEEE Transactions on Acoustics, Speech, and Signal Processing**, Jan 1990, pp 35-45.

[36] Ljolje, A., Riley, M., Hindle, D. and Pereira, F. *The AT&T 60,000 Word Speech-To-Text System.* In **Proceedings of ARPA Spoken Language System Technology Workshop**, Jan. 1995, pp. 162-165.

[37] Lowerre, B. *The Harpy Speech Understanding System.* Ph.D. thesis, Computer Science Department, Carnegie Mellon University, Apr 1976.

[38] Murveit, H., Butzberger, J., Digalakis, V. and Weintraub, M. *Large-Vocabulary Dictation Using SRI's Decipher Speech Recognition System: Progressive Search Techniques.* In **IEEE International Conference on Acoustics, Speech, and Signal Processing**, Apr. 1993, vol.II, pp. II-319 – II-322.

[39] Murveit, H., Monaco, P., Digalakis, V. and Butzberger, J. *Techniques to Achieve an Accurate Real-Time Large-Vocabulary Speech Recognition System.* In **Proceedings of ARPA Human Language Technology Workshop**, Mar. 1994, pp 368-373.

[40] Ney, H., Haeb-Umbach, R. and Tran, B.-H. *Improvements in Beam Search for 10000-Word Continuous Speech Recognition.* In **IEEE International Conference on Acoustics, Speech, and Signal Processing**, Mar. 1992, vol.I, pp. I-9 – I-12.

[41] Nguyen, L., Anastasakos, T., Kubala, F., LaPre, C., Makhoul, J., Schwartz, R., Yuan, N., Zavaliagkos, G. and Zhao, Y. *The 1994 BBN/BYBLOS Speech Recognition System.* In **Proceedings of ARPA Spoken Language Systems Technology Workshop**, Jan. 1995, pp. 77-81.

[42] Nilsson, N.J. *Problem Solving Methods in Artificial Intelligence.* McGraw-Hill, New York, 1971.

[43] Normandin, Y., Bowness, D., Cardin, R., Drouin, C., Lacouture, R. and Lazarides, A. *CRIM's November 94 Continuous Speech Recognition System.* In **Proceedings of ARPA Speech and Natural Language Workshop**, Jan. 1995, pp 153-155.

[44] Odell, J.J., Valtchev, V., Woodland, P.C. and Young, S.J. *A One Pass Decoder Design for Large Vocabulary Recognition.* In **Proceedings of ARPA Human Language Technology Workshop**, Princeton, 1994.

[45] Pallett, D.S., Fiscus, J.G., Fisher, W.M., Garofolo, J.S., Lund, B.A., and Przybocki, M.A. *1993 Benchmark Tests for the ARPA Spoken Language Program.* In **Proceedings of ARPA Speech and Natural Language Workshop**, Mar. 1994, pp 15-40.

[46] Pallett, D.S., Fiscus, J.G., Fisher, W.M., Garofolo, J.S., Lund, B.A., Martin, A. and Przybocki, M.A. *1994 Benchmark Tests for the ARPA Spoken Language Program.* In **Proceedings of ARPA Spoken Language Systems Technology Workshop**, Jan. 1995, pp 5-38.

[47] Pallett, D.S., Fiscus, J.G., Fisher, W.M., Garofolo, J.S., Martin, A. and Przybocki, M.A. *1995 HUB-3 NIST Multiple Microphone Corpus Benchmark Tests.* In **ARPA Speech Recognition Workshop**, Feb. 1996.

[48] Pallett, D.S., Fiscus, J.G., Garofolo, J.S., and Przybocki, M.A. *1995 Hub-4 "Dry Run" Broadcast Materials Benchmark Tests.* In **ARPA Speech Recognition Workshop**, Feb. 1996.

[49] Patel, S. *A Lower-Complexity Viterbi Algorithm.* In **IEEE International Conference on Acoustics, Speech, and Signal Processing**, May 1995, Vol. 1, pp. 592-595.

[50] Paul, Douglas B. *An Efficient A* Stack Decoder Algorithm for Continuous Speech Recognition with a Stochastic Language Model.* In **Proceedings of DARPA Speech and Natural Language Workshop**, Feb. 1992, pp 405-409.

[51] Price, P., Fisher, W.M., Bernstein, J. and Pallet, D.S. *The DARPA 1000-Word Resource Management Database for Continuous Speech Recognition.* In **IEEE International Conference on Acoustics, Speech, and Signal Processing**, 1988.

[52] Rabiner, L.R. *A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition.* In **Readings in Speech Recognition**, *ed.* Waibel, A. and Lee, K. Morgan Kaufmann Publishers, San Mateo, CA, 1990, pp. 267-296.

[53] Rabiner, L.R. *Applications of Voice Processing to Telecommunications.* In **Proceedings of the IEEE**, Vol. 82, No. 2, Feb. 1994, pp. 199-228.

[54] Ravishankar, M. *Parallel Implementation of Fast Beam Search for Speaker-Independent Continuous Speech Recognition.* Technical report submitted to Computer Science and Automation, Indian Institute of Science, Bangalore, India, Apr. 1993.

[55] Ravishankar, M. *et al.* The CMU continuous speech recognition system demonstration. **ARPA Spoken Language Technology Workshop**, Mar. 1994.

[56] Renals, S. and Hochberg, M. *Efficient Search Using Posterior Phone Probability Estimates.* In **IEEE International Conference on Acoustics, Speech, and Signal Processing**, May 1995, pp 596-599.

[57] Rosenfeld, R. *Adaptive Statistical Language Modeling: A Maximum Entropy Approach.* Ph.D. thesis, School of Computer Science, Carnegie Mellon University, 1994.

[58] Rosenfeld, R. and Seymore, K. *Personal communication.* School of Computer Science, Carnegie Mellon University, Mar. 1996.

[59] Schwartz, R. and Chow, Y.L. *The Optimal N-Best Algorithm: An Efficient Procedure for Finding Multiple Sentence Hypotheses.* In **IEEE International Conference on Acoustics, Speech, and Signal Processing**, Apr. 1990.

[60] Schwartz, R.M. *et al.* The BBN continuous speech recognition system demonstration. **ARPA Spoken Language Technology Workshop**, Mar. 1994.

[61] Sites, R.L., editor. *Alpha Architecture Reference Manual.* Digital Press, 1992.

[62] Viterbi, A.J. *Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm.* In **IEEE Transactions on Information Theory**, vol. IT-13, Apr. 1967, pp. 260-269.

[63] Weide, R. *Personal communication.* School of Computer Science, Carnegie Mellon University.

[64] Woodland, P.C., Gales, M.J.F., Pye, D., and Valtchev, V. *The HTK Large Vocabulary Recognition System for the 1995 ARPA H3 Task.* **ARPA Speech Recognition Workshop**, Feb. 1996.

[65] Woodland, P.C. Leggetter, C.J., Odell, J.J., Valtchev, V. and Young, S.J. *The Development of the 1994 HTK Large Vocabulary Speech Recognition System.* In **Proceedings of ARPA Spoken Language System Technology Workshop**, Jan. 1995, pp 104-109.

[66] Woodland, P.C., Odell, J.J., Valtchev, V. and Young, S.J. *The HTK Large Vocabulary Continuous Speech Recognition System: An Overview.* In **Proceedings of ARPA Speech and Natural Language Workshop**, Mar. 1994, pp 98-101.