HIERARCHICAL SEARCH FOR LARGE VOCABULARY CONVERSATIONAL SPEECH RECOGNITION[1]

Neeraj Deshmukh, Aravind Ganapathiraju and Joseph Picone

Institute for Signal and Information Processing (ISIP)
Department of Electrical and Computer Engineering
Mississippi State University, Mississippi State, MS 39762
{deshmukh, ganapath, picone}@isip.msstate.edu

ABSTRACT[2]

Speaker-independent speech recognition technology has made significant progress from the days of isolated word recognition. Today, state-of-the-art systems are capable of performing large vocabulary continuous speech recognition (LVCSR) on audio streams derived from complex information sources such as broadcast news and two-way telephone dialogs. A significant contribution to this advancement in technology is the development of search techniques that find suboptimal but accurate solutions in problems involving large search spaces and extremely complex statistical models. Moreover, these search strategies are capable of dynamically integrating information from a number of diverse knowledge sources to determine the correct word hypothesis, and limit the scope of the search by using a hierarchical search strategy. We refer to this problem as the *decoding* or *search* problem.

This paper describes the complexity associated with decoding using hierarchical representations for linguistic and acoustic knowledge sources. An extensible object-oriented decoder available in the public domain, that leverages current state-of-the-art technology is described to illustrate these concepts. This decoder supports efficient handling of acoustic models for cross-word context-dependent phones, multiple pronunciations of words using lexical trees, and rescoring of word graphs based on N-gram language models in a single pass. It employs a state-of-the-art Viterbi-style dynamic programming algorithm, and is equipped with several heuristic pruning criteria to minimize the consumption of computational resources while maintaining good accuracy.

1. This work was supported in part by the National Science Foundation under Grant No. EIA9809300.9811040.
2. An on-line copy of this paper is available at:
http://www.isip.msstate.edu/publications/journals/ieee_spmag/1999/search/.

1. THE SPEECH RECOGNITION PROBLEM

Speech is one of the most natural means of exchanging information for humans. This has spawned a growing interest in developing machines that can accept human speech as input and act appropriately based on the information conveyed. Enlisting the possible applications of such a system capable of understanding natural human speech is a task limited only by human imagination. In fact, a new field of human language engineering is emerging that attempts to harness this computational power to augment human-to-human or human-to-machine interfaces. The aim of a continuous speech recognition system is, therefore, to provide an efficient and accurate mechanism to transcribe human speech into text. To make such a system ubiquitous, it is important that the system be able to handle a large vocabulary, and be independent of speaker and language characteristics such as accents, speaking styles, dysfluencies (particularly important in spontaneous speech), syntax, and grammar.

Even though human communication through speech appears to be extremely easy, mathematically modeling the underlying processes has proven to be one of the grand challenges of modern computing. Many of the fundamentals of the speech communication process are still not clearly understood and defy rigorous mathematical descriptions. Due to the wide variation in the characteristics of speech produced by humans, and our inability to model such variations compactly, the dimensionality of an expert system based on such a limited understanding of the problem is prohibitively high. A statistical approach to speech recognition circumvents the need for manual encoding of such extraordinary amounts of complex information in favor of self-organization of such knowledge, and therefore is the most popular and successful approach to this problem.

Speech recognition systems today accomplish this task through an excruciatingly detailed analysis of the speech signal. An example of this analysis is summarized in Figure 1. To recognize speech using computers, we must ultimately assign a symbol to every short segment of the speech signal, and then combine these symbols to form words, sentences, actions, intentions, etc. As will be seen shortly, often these symbols represent phones — the basic sound units of the language according to current linguistic theories — which are modeled using Hidden Markov Model (HMM) technology. Since we don’t really know where words or phones begin or end in the signal, or whether pauses occurred between the words, it is extremely productive to let the recognition system decide for us (in an optimal manner) where these units exist in the signal. This is referred to as the segmentation or time-alignment of the speech signal, and is one of the reasons why the decoding problem is so complicated.

To someone skilled in the art of optimization, an obvious solution to this problem is to make use of the principle of dynamic programming (DP) [43], as summarized in Figure 2. Dynamic programming allows different paths through a network to be merged if they converge at the same node. Only the path with the most desirable score at each node needs to be retained. All resources allocated for the discarded paths can be returned to the system for reuse by the software. The DP approach guarantees an optimal solution in terms of the best path at a computational cost that grows only linearly with time and approximately linearly with the number of nodes in the network, assuming reasonable constraints on the model topology. Its ability to produce an optimal solution yet minimize computational costs makes it an extremely attractive algorithm for speech recognition.

In hierarchically structured search problems, such as that shown in Figure 1, the process of path merging is not as simple. For instance, the correct sentence hypothesis for the example in Figure 1 is “hard rock.” However, the search process generates a number of possible alternative word sequences (e.g. “heart wrong”, “hard raw”, “card rock” etc.), all very close to the correct hypothesis or the true utterance in terms of acoustic similarity and segmentation. Moreover, a silence “word” needs to be hypothesized after each word to account for possible pauses between words. In spontaneous speech, such pauses do not necessarily appear at each word end, and therefore the silence hypotheses are optional. This significantly increases the number of possible paths and the bookkeeping required to keep track of theses paths. The latter we will later refer to as the path history. Managing these histories is a concept central to the speech recognition search problem.

To make matters worse, extra care needs to be taken when merging partial paths terminating in such silences. For example, silences following the words “hard” and “heart” in Figure 1, though terminating at the same instance in time, cannot be merged as they represent two paths with different word histories. Similarly, neither of these silences can be merged with the silence hypothesized at the start of the utterance. At the same time, even though the path “sil hard rock” appears different from the path “sil hard sil rock” (the former hypothesis is missing the second silence), they both need to be treated as the same word sequence as the silence does not carry any syntactic meaning.

In this paper, we first introduce the fundamental components of a typical speech recognition system. We formally define the search problem and present an overview of several ways this problem can be solved. For an excellent comprehensive discussion of this topic, see a companion paper appearing in this issue [43]. Next, we introduce a specific implementation of a search algorithm, discuss its performance on two drastically different tasks, and analyze its run-time characteristics. The intent here is to provide the reader with first-hand knowledge of many important heuristics that have become commonplace in LVCSR systems, and are crucial to their success in extremely complex applications. There are many systems available that handle vocabularies of several thousand words in somewhat limited domains such as dictation or telephone queries. However, there are few systems capable of processing spontaneous speech such as telephone conversations, or other such unrestricted domains with performance approaching state of the art.

2. STATISTICAL METHODS IN SPEECH RECOGNITION

The most popular framework for the speech recognition problem is a statistical formulation [29] in which we choose the most probable word sequence from all word sequences that could have possibly been generated. For a sequence of words $W = w_1, w_2, \ldots, w_N$, if A is the acoustic evidence that is provided to the system to identify this sequence, then the recognition system must choose a word string $\hat{W}$ that maximizes the probability that the word string W was spoken given that the acoustic data A was observed:

$$\hat{W} = \underset{W}{\operatorname{argmax}}\ p(W/A)\ . \tag{1}$$

$p(W/A)$ is known as the a posteriori probability since it represents the probability of occurrence of a sequence of words after observing the acoustic signal A.

2.1. The Bayesian Approach

It is obviously difficult to directly compute the maximization in Equation 1 since there are effectively an infinite number of such word sequences for a given language. This problem can be significantly simplified by applying a Bayesian approach to finding $\hat{W}$:

$$\hat{W} = \underset{W}{\operatorname{argmax}}\ p(A/W)p(W)\ . \tag{2}$$

The probability, $p(A/W)$, that the data A was observed if a word sequence W was spoken is typically provided by an *acoustic model*. The likelihood $p(W)$ that enumerates the a priori chances of the word sequence W being spoken is determined using a *language model*. Probabilities for word sequences, which we refer to as hypotheses, are generated as a product of the acoustic and language model probabilities. The process of combining these two probability scores and sorting through all plausible hypotheses to select the one with the maximum probability, or likelihood score, is called decoding or search. Figure 3 illustrates the basic schematic structure of a such a statistical approach to speech recognition.

What makes the speech recognition problem difficult is that we never know exactly what individual sounds or words were spoken until the entire word sequence has been identified. Therefore we must allow the recognizer to systematically search all possible word sequences, including all possible start and stop times for each word, all possible ways silence could have occurred between words, and all possible ways the word could have been pronounced, to maximize the overall probability of the word sequence. Contrast this with the problem of a spell-checking program that processes keyboard input. Here we know the identity of each letter (the letter produced when you strike a key is a deterministic process), and merely have to find the "closest" correctly spelled word — a fairly small combinatorial problem by speech recognition standards. Given that $p(A/W)$ can be nonzero for most words at any point in time, we must rank order hypotheses and pursue only the most promising alternatives to keep the search space manageable. Language modeling plays a crucial role in reducing the size of this search space.

2.2. Acoustic Models

A key assumption in stochastic speech processing is that the speech signal is stationary over short intervals of time. Signal processing plays an important, but often unappreciated role in a speech recognition system. The acoustic front-end converts the analog speech signal into a sequence of feature vectors. A comprehensive tutorial of this aspect of a speech recognition system can be found in [56] (the acoustic front-end is not the focus of this paper). The main goal of this subsystem is to generate a sequence of feature vectors representing the temporal and spectral behavior of the signal. In our statistical approach, each feature vector is considered to be statistically independent from other feature vectors. Though this is not the case in practice, it is an extremely convenient assumption for the statistical models we will develop.

Typically, the signal is divided into 10 msec frames using an overlapping window approach in which each window accounts for 25 msec of the signal. The spectral features can be extracted using a multitude of techniques. The most popular acoustic front-end in use today employs 39 parameters per frame of speech data, and consists of the signal energy and 12 mel-spaced cepstral coefficients, plus their first and second-order temporal derivatives [2, 9, 13, 19, 25, 52, 56].

After a sequence of acoustic feature vectors Y is obtained from the front-end, the acoustic models need to provide a probability, or score, for any such Y given a word sequence W. It is impractical to do this calculation for every possible word in a large vocabulary application since it would require too many models to be processed. Hence, the word sequences are decomposed into basic sound units called *phones*. Since acoustic modeling is not the major focus of this paper, the reader is referred to [14, 27, 42, 55, 57, 58, 59] for more details on this topic.

An HMM is used to model each phone. The HMM is a doubly stochastic state machine that has a Markov distribution associated with the transitions across various states, and a probability density function that models the output for every state. Depending on the complexity of the recognition problem, this distribution can be modeled as a discrete-valued or continuous-valued process. In speech recognition applications the choice of this output probability function is crucial as it must model all of the intrinsic spectral variability of real speech. Most current state of the art systems use a mixture of multivariate Gaussian distributions to model context-dependent sequences of three phones (triphone models).

If w_i are the weights for combining the scores of the m mixture components, and $\aleph(\underline{\mu}_i, \Sigma_i)$ are the m multivariate Gaussian distributions of dimension d that make up the output distribution of the state s, then the probability of the acoustic feature vector $\underline{x}$ given the state s can be calculated as

$$p(\underline{x}/s) = \sum_{i=1}^{m} w_i \frac{1}{(2\pi)^{d/2}|\Sigma_i|^{1/2}} \exp\left[-\frac{1}{2}(\underline{x}-\underline{\mu}_i)^T \Sigma_i^{-1} (\underline{x}-\underline{\mu}_i)\right]. \qquad (3)$$

Figure 4 shows some topologies of the HMMs typically used to model context-dependent phones in large vocabulary speech recognition systems. These models can be efficiently trained using the Baum-Welch forward-backward training algorithm [6] or the Viterbi algorithm [18].

Since a typical LVCSR system requires thousands of models to account for all possible sounds in all possible contexts, the total number of parameters in an LVCSR system is prohibitively large. It is common for a state-of-the-art system to have several million parameters that need to be trained simultaneously from data. Most LVCSR systems therefore use clustering approaches to reduce parameter count. State-tying [69] and mixture-tying [17] are popular solutions to this problem. A large database of labeled training data is also required to train such systems. It is not uncommon to use more than 200 hours of speech training data to develop a state-of-the-art system.

2.3. Language Models

A language model (LM) provides constraints on the sequences of words that are allowed to be recognized. In particular, it provides a mechanism to estimate the probability of some word w_k in a word sequence W given the surrounding words. Ideally, the LM integrates linguistic knowledge, domain knowledge, and any other pertinent information to reduce the size of the search space. The linguistic complexity of the search space is often measured in terms of perplexity [31], an information theoretic measure closely related to the average branching factor or number of words possible at each point in the dialog or transaction.

Since the probability of a word being spoken often depends on the words spoken previously, a simple but effective way of modeling language is to model a sequence of M words as an n^{th} order Markov chain:

$$p(W) = p(W_1^M) = \prod_{i=1}^{M} p(w_i | w_{i-1}, w_{i-2}, \ldots, w_{i-n}) . \tag{4}$$

This gives rise to the notion of N-grams [30] where the probability of the occurrence of a word depends only on its N predecessors:

$$p(w_k | W_1^{k-1}) = p(w_k | W_{k-N+1}^{k-1}) . \tag{5}$$

N-grams indirectly encode syntax, semantics and pragmatics by concentrating on the local dependencies between words. Also, N-gram probabilities can be directly computed from text data and therefore do not require explicit linguistic rules like a formal language grammar. N-grams are a good example of how deeply rooted statistical methods are in speech recognition. Most systems use a trigram back-off language model [44], though there are some systems that have ventured into higher-order N-grams [26], long-range dependencies [33], cache [32], link [34] and trigger models [35], class grammars [28], and decision-tree clustered language models [4].

The net result of all such techniques is to limit the number of alternatives that must be investigated to find the most probable sequences of words. As stated earlier, the spontaneous speech recognition problem is rather unique in that any word can theoretically occur at any time, with some nonzero probability. Hence the search space is large, and our search algorithm must be efficient. In the next section, we present an overview of the fundamental issues in search for speech recognition.

3. SEARCH ALGORITHMS

A search strategy is used to select a word sequence with the highest probability given the observed acoustic data. The number of possible hypotheses grows exponentially as a function of the number of models, vocabulary size and the form of linguistic constraints; and this imposes formidable requirements on the computation and storage capability of the system for the implementation of the search algorithm.

3.1. The Complexity of Search

Consider a simple application of recognizing spoken telephone numbers. The vocabulary size could reasonably be constrained to 11 words (the digits “zero”, “one”,..., “nine” and the word “oh” often used in place of “zero”). In this case, it is easy to build an acoustic model for each complete word. For an unconstrained grammar, where any word is equally likely to follow any other word, the search space complexity is an exponential function of the length of the digit string. In other words, for sequences six digits long, there are a total of 11^6 possible alternatives, and every extra digit in the string increases the number of alternatives by an order of magnitude. Obviously, an exhaustive search through all possible hypotheses to find the best one is highly impractical. Good search strategies attempt to save on this computation by modifying the search space via imposition of some constraints in terms of additional knowledge about the domain of the speech recognition application. While these constraints reduce the number of hypotheses to be enumerated, there are some costs associated with their implementation.

For instance, a language model can be derived based on the knowledge that telephone numbers are typically 4, 5, 7, 10, or 11 digits long, and only certain numbers are allowed for area codes. Applying this language model can limit the search space to a fairly large extent, because the number of possible next words is now dependent on the branching factor, or perplexity [31], of the language model instead of the vocabulary size. In turn, the system needs to load the language model probabilities in addition to the acoustic models. It also needs to modify the search control flow to look up the list of the most likely words to follow the current word, and this adds some bookkeeping and storage overhead to the system.

Things get significantly more complicated in a task where the knowledge used to constrain the search space is organized in a hierarchical fashion, since information from each of these sources is applied to the hypotheses at different levels in the hierarchy. For example, suppose instead of using word-level acoustic models, we use models that represent the basic sound units, or phones, in each word. Further, a word such as “zero” could be treated as having two pronunciations, “zero” and “oh.” We can add a silence model that, in addition to being a good model of the spectral characteristics of the background channel, also includes common mouth noises such as lip smacks, breath noises, etc. The net result is a system that better integrates linguistic and acoustic knowledge about the problem, but now consists of a network that describes digit strings in terms of words (a language model), a network that describes words in terms of phones (a pronunciation dictionary), and a network that describes phones in terms of sequences of spectral vectors derived from the speech signal (acoustic models).

In large vocabulary speech recognition the related complexity issues are even more pronounced. It is no longer practical to use word-level acoustic models, since sufficient training data for a large number of words is very expensive to generate. Sub-word units such as phones become essential. Words and sentences can be constructed by concatenating the corresponding phones that constitute their pronunciation. As a result, a new component is added to the recognition system. This is the pronunciation dictionary or lexicon, which consists of all the words in the vocabulary and their pronunciations in terms of the phones. Often, words have more than one possible pronunciation, and therefore multiple paths need to be explored for a single word.

Moreover, it has been found that simply using the models for individual phones does not yield good recognition performance, since the actual articulation of each phone is influenced strongly by its context — the preceding and following phones. Context-dependent phone models use either the left (previous) or right (next) phonetic context (such two-phone models are known as diphones), or both the left and right contexts (these are called triphone models). Such detailed acoustic models give measurable improvements in recognition performance, yet require an increase in the system size (e.g. the number of acoustic models rises from approximately 50 context-independent phones to over 80,000 triphones).

State-of-the-art LVCSR systems routinely use triphone acoustic models that also take into account the phonetic context across word boundaries (these models are referred to as cross-word triphone models). This adds one more degree of complexity, since the end of each word needs to be hypothesized multiple times, once each for a different phonetic context corresponding to the next possible word. Some of this complexity can be reduced using techniques such as phonetic lexical trees [5, 7], but overall the amount of information to be stored during the hierarchical search process is quite large.

Typically, the decoding strategy in a speech recognition system uses dynamic programming [45] to find the most likely word sequence given the acoustic models, language model constraints, and the input audio data. However, as described earlier in this section, the path calculations through the search space often involve a hierarchy of graphs (sentences, words, phones, and acoustic model states). Figure 5 displays the hierarchical structure of knowledge sources during search. The control structure required to perform this search is conceptually simple, but extremely hard to implement efficiently in software. It involves extensive data structure manipulations, particularly in the case of a trigram language model and cross-word context-dependent phone models. As a result, decoding is the most time-consuming component of the speech recognition process.

3.2. Typical Search Algorithms

The decoding process needs to be restructured to restrict the search space without compromising system performance. Most popular techniques for restructuring the search space [36] involve clever ways to share information amongst active hypotheses. Use of such approximations forces the decoder to make suboptimal choices. However, it has been observed that a suboptimal solution often does not impact performance (i.e. the number of words misrecognized by the system). In this section, we review some popular search techniques. Throughout this section it is important to note that as you decode an utterance, its cumulative path probability — which is a product of the probability computed at each frame of input speech data — monotonically decreases with the number of frames. This is the result of a fundamental concept in probability theory that probabilities are bounded by $[0, 1]$, and hence the product of these probabilities is a decreasing function of time. Therefore, comparison of two hypotheses that account for a different number of frames of data is difficult, since the hypothesis accounting for more frames of data will on average have a lower probability. During implementation of the search, underflow problems may arise for representing the scores of longer paths. Therefore, it is standard practice to use the logarithm of the probability (known as **likelihood**) to represent the path scores, and products of probabilities are handled as sums of likelihoods.

Viterbi Search

Viterbi search and its variant forms belong to a class of breadth-first search techniques. Here all hypotheses are pursued in parallel and gradually pruned away as the correct hypothesis emerges with the maximum score. In this case, the recognition system can be treated as a recursive transition network composed of the states of HMMs in which any state can be reached from any other state. The Viterbi search algorithm [67] builds a breadth-first search tree out of this network following the steps enumerated in Figure 6.

Viterbi search is time-synchronous, i.e. at any stage all partial hypotheses generated during the search terminate at the same point in time. Since these hypotheses correspond to the same portion of the utterance, they can be directly compared with each other. However, a complete Viterbi search is impractical for even moderate-sized tasks because of the large size of the state space. A Viterbi beam search is used to reduce the search space.

In Viterbi beam search only the hypotheses whose likelihood falls within a fixed radius, or beam, of the most likely hypothesis are considered for further growth [11, 37, 38, 43]. The best beam size can be determined empirically or adaptively. The advantage of the dynamic beam heuristics is that it allows the search to consider many good hypotheses in the absence of a clearly dominant solution. Conversely, in case of a clear best hypothesis few others need to be maintained. The main problem with the Viterbi beam search, as we will see in the next section, is that the state-level information cannot be merged readily to reduce the number of required computations.

Many variations of Viterbi beam search have been proposed to improve its performance. For instance, different beam widths can be applied at different levels in the search hierarchy [1], and each of these can be adjusted independently based on the number of active paths at that level. In another modification, a tighter pruning beam can be applied to the paths at initial frames of data to limit the extent of hypothesis generation [16]. In very large vocabulary tasks, a tree-structured network is used to represent the search space in which the states corresponding to phones that are common to different words are shared by different hypotheses [50]. This approach uses the fact that the uncertainty about the identity of the word is much higher at its beginning than at the end. Therefore, more computations are required at the beginning of a word than towards its end.

Stack Decoders

The stack decoding algorithm [3] is similar to the A* search popularly used in artificial intelligence [49]. It is a depth-first technique in which the most promising hypothesis is pursued until the end of the speech data is reached. The basic stack decoder paradigm [53, 54] can be summarized as described in Figure 7. Stack decoding algorithm requires an evaluation function to compare hypotheses of different lengths. Since the score of a path progressively decreases with time (it is a product of probabilities), the search process is biased to always prefer shorter hypotheses. This problem is overcome by normalizing the score of a path based on the number of frames of data it spans. However, the A* stack decoder suffers from problems of speed, size, accuracy and robustness for large vocabulary spontaneous speech applications.

Multi-Pass Search

A multi-pass search algorithm [8, 10, 40, 47, 48, 61, 62, 65] employs a coarse-to-fine strategy to decoding. In this approach, computationally inexpensive acoustic models are initially used to produce a list of likely word hypotheses. These hypotheses are later refined using more detailed and computationally demanding models. The first search pass (often called a fast match) produces either an N-best list of possible word sequences or a word graph (or lattice) as its output. These entities are illustrated in Figure 8.

For example, an initial search pass can be performed using word-internal context-dependent phones with a bigram language model to generate a list of candidate hypotheses. Then, in a second pass of decoding, a trigram language model, which treats common three-word sequences, can be used with cross-word context dependent phones. The resulting two-pass search will give performance comparable to a single-pass Viterbi search, but often require less computational resources. An important emerging variant of the stack decoding technique is envelope search [23].

Forward-Backward Search

Forward-backward search algorithms use an approximate time-synchronous search in the forward direction to facilitate a more complex and expensive search in the backward direction [8, 47, 48]. This generally results in speeding up the search process on the backward pass as the number of hypotheses to be explored is greatly reduced by the forward search.

A simplified acoustic or language model is used to perform a fast and efficient forward-pass search in which the scores of all partial hypotheses that fall above a pruning beam width are stored at every state. Then a normal within-word beam search is performed in the backward direction to generate a list of the N-best hypotheses. The backward search yields a high score on a hypothesis only if there also exists a good forward path leading to a word-ending at that instant of time. Figure 9 describes the concept of the forward-backward search.

Similar to the Baum-Welch algorithm used for training acoustic models [6], the total path score at each state s of the HMM at time t is obtained by combining the scores on the forward and backward passes. The forward pass score $\alpha_t(s)$ of a partial path represents the joint probability of observing the input feature sequence over time instants 1 through t, and being in a state s at the time t. Similarly, the backward pass score $\beta_t(s)$ denotes the joint probability of a path that accounts for the observed features from time $t + 1$ until the final frame T of input data, emerging from a state s at time t. Thus the total path score $\gamma_t(s)$ is given by combining the scores of the forward and backward paths that meet in the state s at time t.

$$\gamma_t(s) = \frac{\alpha_t(s)\beta_t(s)}{\alpha_T} \tag{6}$$

Here α_T is the score for the best complete forward path, and is used for normalization of path scores so that different paths can be compared. The N-best word sequences obtained using this procedure are rescored using more sophisticated acoustic and language models to obtain the best sentence hypothesis.

Forward-backward search algorithms have greatly facilitated real-time handling of large-scale speech recognition tasks. The backward pass search is fast enough to be performed without any perceptible delay after the forward search. The forward pass can be made extremely suboptimal and efficient, as the forward path scores do not need to be very accurate relative to the path scores obtained in the backward pass.

So far, we have provided a general overview of different search strategies prevalent in large vocabulary speech recognition. It is admittedly difficult to implement these algorithms from this high-level discussion. However, the main goal of the preceding sections was to provide some insight into how important organization of the search space is to the decoding process. Now we move to the main goal of this paper — a discussion of a specific implementation of the Viterbi search. We describe this particular algorithm in detail, and analyze its performance on two applications very different in terms of the search complexity.

4. A TIME-SYNCHRONOUS VITERBI-BASED DECODER

As we have established, the software complexity of a search algorithm is considerable, and the effort required to build an efficient decoder is quite large. For this reason, we describe below an algorithm that is available in the public domain [15]. This speech recognition system is designed to efficiently and transparently handle tasks of varied complexity, from connected digits to conversational speech. It is somewhat typical in its architecture, and similar to a number of systems available from a variety of sources [26, 60, 66, 68]. This system implements a time-synchronous Viterbi decoder, and includes the following modes of search:

- *Recognition Supervision:* forced alignment using the reference transcription and arbitrary pronunciation models; flexible alignments for dealing with noisy reference transcriptions
- *Decoding:* arbitrary hierarchical finite-state language models; N-grams with arbitrary acoustic model topologies; one-best or N-best decoding; generation of alternate word choices at each point in time
- *Word Graph Postprocessing*: word graph error rate calculations; word graph rescoring with new language models, acoustic models, etc.

The system is designed in an object-oriented fashion and written entirely in C++. It is designed to facilitate introduction of new research — something not easily done in most recognition systems, and to support a wide range of algorithm choices for each component of the system. The core of this system is a single-pass, lexical-tree based decoder that implements a hierarchical variation of the Viterbi time-synchronous search paradigm. We now present a detailed account of the evolution of the search algorithm implemented within this system.

4.1. Complexity of Search

As described earlier, the primary inputs to a decoder, beyond the speech data, are:

- *lexicon*: contains all the words in the system's vocabulary along with their pronunciations (often there are multiple pronunciations per word)
- *acoustic models*: HMMs that represent the basic sound units the system is capable of recognizing
- *language model*: determines the possible word sequences allowed by the system (encodes knowledge of the syntax and semantics of the language)

Not surprisingly, the complexity of the search space is strongly dependent on the representations used for these knowledge sources. Though computing is vastly more powerful today than ever before, it is still not possible to run an arbitrary combination of the above items for most LVCSR applications. Careful design of these components is crucial to the development of a successful system.

One obvious simplification that can be made to a system is to allow each acoustic unit to represent a phoneme, one of approximately 50 basic sound units used to represent a language (or the symbol set used to describe pronunciations in a dictionary). We refer to these as context-independent acoustic models since the choice of any symbol does not depend on its adjacent sounds. These same symbols can be used to represent pronunciations in the lexicon, thereby giving a one-to-one correspondence between the lexicon and the acoustic model set. Another simplification that can be made is to define a language model using a finite state machine, or network, that describes each possible sentence as a unique path through this network. This is an approach often used in small to medium-sized vocabulary applications. Systems based on these two principles are popular because of their conceptual simplicity, but do not deliver good performance in practice.

An important step beyond such a simple system is to allow the language model to represent n-tuples of words (such as “see Jane run” or “the red hat”). We refer to this as an N-gram language model. N-gram orders of two (bigrams) and three (trigrams) are popular models. Such language models, though also conceptually attractive, significantly increase the number of possible next words for each word-ending hypothesis during the search process. This makes the search problem significantly harder compared to a network language model.

The search complexity increases further if context-dependent phones are used to model the acoustics, since the number of models required to represent all the combinations of phones occurring in the data is extremely large (e.g. from approximately 50 context-independent phones to close to 80,000 context-dependent phones). While such phonetic context can be limited within each word (word-internal models), the search space explosion is even more severe when context across word boundaries (cross-word models) is also taken into account. Here, each word-end hypothesis needs to be replicated to account for all the phonetic contexts derived from the possible next words. Figure 10 describes this transition in the scale of the search problem. We now describe the steps required to go from a simple search engine to the complex engines employed by state-of-the-art systems.

Network Decoding

If the linguistic constraints on the search space are described in terms of a network of words, then the decoder can expand this network in terms of the phones constituting the pronunciation of each of the words. The word network can be a grammar that defines the structure of the language used in the recognition task, or a word graph generated by a previous recognition process. The decoder evaluates the states of each phone model in the network active at the time and propagates paths through the network. The corresponding language model score is added to the acoustic path score as soon as the path lands on a word-level node in the network.

Figure 11 illustrates the process of network decoding using a word-level network along with a corresponding network of word-internal context-dependent phones. Note that during the search process, there may exist two or more paths which reach the triphone *hh-aa+r* at the same time. However, they cannot be merged into a single path if such instances of the same triphone correspond to different nodes in the network at the word level. Thus an instance of an acoustic model is represented in terms of the identity of the associated triphone and the word-level node in the network. The two instances of the phone *hh-aa+r* are circled in Figure 11.

Network decoding can be performed efficiently only for moderately sized vocabularies using word-internal acoustic models. Since the decoder needs to expand the whole network in terms of the constituent phones before processing any data, the complexity of the search and memory requirements are directly proportional to the size of the expanded network. Often, we seek ways to reduce the size of the search network to increase efficiency. Dynamic expansion [46] of the network is an important approach to avoid a large initial memory allocation to hold the entire structure.

N-Gram Decoding

For larger vocabularies, the N-gram language model provides a relatively compact representation of the linguistically probable word sequences since it provides estimates of the likelihood of the occurrence of a word based on the previously observed $N-1$ words. If the vocabulary size is M words, then to provide complete coverage of all possible word sequences the language model needs to consist of M^N N-grams (i.e. sequences of N words). This is prohibitively expensive (e.g. a bigram language model for a 40,000 words vocabulary will require 1.6×10^9 bigram pairs), and many of such sequences have negligible probabilities. Therefore, the language model typically consists of only a subset of the possible N-grams, and the likelihood of the other word sequences can be estimated using a back-off model [44]. For instance, in a bigram language model the probability of a word sequence (w_i, w_j) is given by

$$p(w_i, w_j) = \begin{cases} p(w_j | w_i) & \ldots (w_i, w_j) \text{ exists in LM} \\ b(w_j) p(w_i) & \ldots \text{ otherwise} \end{cases} \quad (7)$$

where $b(w_j)$ is the back-off weight for the word w_j, and $p(w_i)$ is the unigram probability (the probability of any occurrence) of the word w_i. The score is added to the path at the instant where the evaluation of the word w_i has just ended, and the path is about to be propagated into the word w_j, i.e. at the start of the new word.

In N-gram decoding, different paths at the same instant of time can be differentiated only based on the phone model and the $N-1$ word history of each path. Thus paths with very different origins can be merged later in time if they have the same current instance, which is now defined by the phone model and the N-gram history word sequence. Figure 12 illustrates this approach for a bigram language model.

Even though N-gram language models store only a small subset of all the possible sequences of N words, they are significantly large for large vocabulary applications. For instance, a bigram language model for a 20,000 word application involving telephone conversations may have about 300,000 bigrams along with the 20,000 unigrams; and a trigram language model for this task might hold an additional 200,000 trigrams. Therefore, loading the entire language model often poses severe demands on the system memory, and also makes the language model score lookup cumbersome. Therefore, an alternative approach to implement the language model is to cache the N-gram scores of all the active words (since these are likely to remain active for the next few frames) in memory, and leave the rest of the language model on disk.

Cross-Word Acoustic Models

Our prior discussions have been based on word-internal, context-dependent acoustic models. These are found to yield satisfactory performance for clearly articulated speech, such as read speech [51]. Word-internal triphones are unable to model pronunciation effects that occur across word boundaries, often referred to as a form of coarticulation, since they do not account for the full acoustic context for the first and last phone of a word. For instance, in the example in Figure 11, the word sequence “hard rock” is translated to word-internal triphones as *hh+aa hh-aa+r aa-r+d r-d r+ao r-ao+k ao-k*. The phone *r-d* at the end of “hard”, as well as the phone *r+ao* at the start of “rock”, do not contain knowledge of the phonetic context of the adjacent word.

In spontaneous or conversational speech, the pronunciation of a word depends largely on the coarticulation effects of the surrounding words (e.g. “did you” becomes “didja”, “three eight” is articulated as one word). To reduce the acoustic mismatch at word boundaries and model the effect of phonetic context across word boundaries, cross-word context dependent models are used in conversational speech recognition. This greatly complicates the search problem since the last phone in the current word becomes dependent on the next word, which is not known until later in time. Hence, we must change our search strategy so that we can handle such deferred decisions.

4.2. Search Space Organization

Figure 13 illustrates the cross-word phonetic network generated for the word graph in Figure 11. As can be seen here, the use of cross-word context increases the size of the network considerably, since every word end needs to be hypothesized multiple times, once each for the possible next words. Thus there is a large fan-out in the number of possible paths through the network at the end of each word, and even for a moderately large vocabulary size the search space explodes with the number of potential hypotheses.

This problem is even more severe for N-gram decoding using back-off language models, since any word has a chance to follow any other word and all such paths need to be hypothesized at the end of each word. With cross-word context, the end of each word results in M possible paths where M is the total number of words, and the complexity of the search space becomes M^{2t} with respect to the time t.

Lexical Trees

The solution to this combinatorial explosion lies in exploiting the fact that even though the vocabulary size may be large, the number of phones (context-independent) used to represent their pronunciations in the lexicon is very small (about 50 for English, for instance). Thus the number of possible unique phonetic contexts is much smaller than the number of possible next words, and by sharing this phonetic context across all the words the number of paths to be grown at each word end can be reduced drastically.

This concept results in the implementation of a lexical tree-based search [41]. A lexical tree or pronunciation prefix tree is used to represent the pronunciations of all the words in the vocabulary. Each node in the lexical tree is associated with a monophone in the pronunciation of the words (see Figure 14) and can be shared by multiple words with the same partial pronunciation. By sharing phones across different words (as opposed to using a separate instance of every phone in the pronunciation of each word) the lexical tree provides a compact representation of the acoustic-phonetic search space, as well as a mechanism to efficiently handle multiple pronunciations of the same word. A terminal node of the lexical tree signifies a unique word.

While the lexical tree nodes are associated with monophones (i.e. context-independent phones) constituting the pronunciations of the words, the models used to represent the acoustics of the speech typically use context-dependent phones (such as triphones). Building a lexical tree out of context-dependent phones is not practical as it significantly increases the tree size (and therefore the memory requirements), particularly when cross-word triphone models are used for acoustic modeling.

To avoid this problem, we use a technique known as *dynamic generation of context-dependent phone models*. Context-dependent phones, or triphones in this case, are generated dynamically by traversing the lexical tree nodes at each step as illustrated in Figure 15. The instance structures contained in the path markers keep track of the current lexical node, and create the next triphone by creating the appropriate contexts from the predecessor phone and all the child lexical nodes respectively. Cross-word triphones are created as needed by spanning the terminal node representing the ending word in the current lexical tree and all the start nodes of the lexical tree corresponding to the next words.

A drawback of this approach is that for an N-gram language model with back-off probabilities, every word in the vocabulary can be instantiated at any word end with the appropriate likelihood as specified by the language model. Therefore a really large lexical tree that covers all the words in the lexicon is required at every word end during the search process. Since the language modeling scores applied for each word depend on the predecessor words on the path, the language model scores at the terminal nodes in the tree must depend on the predecessor word. As a result, for every word end the LM scores stored in the lexical tree representing that node are different, and a copy of the lexical tree has to be made with the corresponding LM scores. This approach is referred to as a *lexical tree-based search*.

For large vocabulary applications, even a few copies of the complete lexical tree quickly overshoot the available memory. The decoder avoids this explosion in memory requirements by dissociating the LM scores from the lexical tree and using only a single tree that is independent of the predecessor words. The language model score for a word is calculated on an as-needed basis and stored in the instance associated with the corresponding history word and lexical node. Since the instances are reused in the decoder, the score calculation needs to be done only once.

In word graph rescoring modes, each lexical tree is associated with a node in the word graph and covers only the child nodes representing the next possible words, and therefore is of a much smaller size. The tree for a node is created only when the word corresponding to this node has been evaluated in the decoding process. It is shared across multiple instances of the word node. The language modeling score corresponding to the arc connecting the parent word node to each child word node in the word graph is stored in the instance corresponding to the child word. A tree no longer actively used for decoding is pruned away to save memory.

Also, due to sharing of phones across different words it is no longer possible to define a unique path instance only in terms of the acoustic model and the word history (either as the N-gram or the word network node). Since different instances of a word correspond to different lexical trees (or virtual tree copies), the instance definition also requires the identity of the lexical tree. Moreover, it is possible to come across the same triphone model at more than one place in the same lexical tree (see Figure 13). Therefore, the unique path instance is defined in terms of the phone model identity, the appropriately defined word history and the identity of the current lexical tree node.

As you might imagine, decoding the language model now has become a problem of immense proportions. We cannot simply entertain all possible paths in this network. Reduction of the search space by discarding unlikely paths is extremely important, and something that humans do extremely well by anticipating the next set of possible words. We now describe the process by which we implement lookahead in a speech recognition system. This plays a crucial role in limiting the size of the search space in the lexical tree approach.

Language Model Lookahead

Since the language model provides additional constraints on the search space by assigning different likelihoods to the possible words, it is beneficial to apply the language model scores to the hypotheses as early in the search as possible. Without linguistic knowledge, a large number of competing paths have fairly similar acoustic scores and it is difficult to discriminate between the more probable paths and the unlikely ones. As a result, a much larger than necessary number of competing hypotheses needs to be propagated forward, increasing the computational and memory requirements on the system.

In a decoder that does not use lexical trees, the identity of each next word is uniquely known at the end of the predecessor node and therefore the correct LM score can be applied at the instantiation of the very first phone in the pronunciation of the word. On the other hand, due to the phone sharing that occurs in lexical trees, the identity of a word is uniquely known only at the terminal node for that word. Therefore, the correct value of the language model score for the word for a given history is also known only at the end of the evaluation of that word. Often, the language model score for the terminating word is stored in this node as well. The delay in the application of the LM score at the word end as opposed to the start of the word allows for undesirable growth in the complexity of the search, and therefore must be avoided.

State-of-the-art decoders use a technique called language model lookahead [46] to overcome this problem. Here, the path markers corresponding to the models internal to a word (i.e. covering non-terminal lexical tree nodes) store in their instance the maximum LM score of all the words covered by that lexical node. This score is appended to the path score temporarily for the sake of pruning comparisons, and removed immediately thereafter. Once a terminal node is reached in the lexical tree, the identity of the word is uniquely known and the actual word LM score is added to the path score.

There is one more important issue related to the organization of the search space. We have seen that at any given instance, many different hypotheses require the same acoustic model to be evaluated. Evaluation of acoustic models, particularly the evaluation of the Gaussian model embedded at each state in an HMM, often comprises about 50% of the total computation time in a system. Constantly reevaluating these states can result in a significant amount of inefficiency in the system. We briefly describe next the process by which this is avoided.

Acoustic Evaluation

At each frame, the decoder reads in a new feature vector of speech data and evaluates its probability for all the active states of the acoustic models. Typically, this involves computation of the Mahalanobis distance of the feature vector from a weighted mixture of multivariate Gaussian distributions as described in Equation 3. As described earlier, the decoder deals with likelihood scores instead of probabilities to avoid underflow problems, and therefore only the logarithm of the Mahalanobis distance needs to be calculated. The resulting likelihood score is given by

$$l(\underline{x}/s) = \sum_{i=1}^{m} [(\underline{x} - \underline{\mu}_i)^T \Sigma_i^{-1} (\underline{x} - \underline{\mu}_i) + K_i] \quad (8)$$

where K_i is a constant term dependent on the particular distribution and w_i.

This is typically the most expensive computation in search, since it needs to be conducted for every active state for each frame. Since the same state can be accessed by different models (due to state-tying), or by different instances of the same model corresponding to different paths, this likelihood score needs to be calculated multiple times in each frame. For efficiency reasons, the decoder performs this calculation only once per frame, when this state is accessed for the first time by a path. The likelihood score evaluated is stored locally with the state information and reused whenever that state is revisited in that frame.

4.3. Search Space Reduction

Thus far, we have discussed the complexity of the search problem and the organization of the resulting search space. We have noted that discarding improbably or unlikely paths is an important way to maintain computational efficiency. Modern speech recognition systems use some extremely clever approaches to achieving this goal. These techniques belong to a family of algorithms known as *beam search* in the artificial intelligence literature, and are referred to as pruning algorithms.

Pruning

In order to conserve the computing and memory resources, it is imperative to identify low-scoring partial paths that have a very low probability of getting any better, and stop propagating them further. The process of removing such paths from the search space is known as pruning. A number of heuristic criteria are applied to identify such paths and to set the appropriate thresholds on path scores which allow only qualified paths to be grown. Some commonly used heuristics are:

- setting pruning beams based on the hypothesis score
- limiting the total number of model instances active at a given time
- setting an upper bound on the number of words allowed to end at a given frame

Most pruning techniques add to the bookkeeping overhead of the system (such as sorting of paths based on scores), but this is more than amply compensated by the ensuing reduction in the search complexity.

Our decoder allows the user to set a separate threshold, or beam, at each level in the search hierarchy (typically words, phones, and states). A constant likelihood value, known as the beam width, is added to the maximum path score at each level at that frame of time, and all paths with a score difference larger than the beam width compared to the maximum score are removed from further consideration. This is referred to as *beam pruning*. The beam width at each level is determined empirically, and the beam threshold is computed with respect to the best scoring path marker at that level. For instance, if at a frame t the best path scores at the state, phone and word levels are respectively given by

$$q_{max}(s, t) = \max_{s} \{q(s, t)\}$$

$$q_{max}(p, t) = \max_{p} \{q(p, t)\} , \qquad (9)$$

$$q_{max}(w, t) = \max_{w} \{q(w, t)\}$$

then for beam widths of $b(s)$, $b(p)$ and $b(w)$, the decoder prunes all hypotheses which satisfy

$$q(s, t) < q_{max}(s, t) + b(s)$$

$$q(p, t) < q_{max}(p, t) + b(p) \ . \qquad (10)$$

$$q(w, t) < q_{max}(w, t) + b(w)$$

State-level pruning is also referred to as the global beam pruning if applied across the search hierarchy (e.g. phone and word levels), in conjunction with the level-specific pruning criteria. In the decoder control flow, the state-level beam is applied right after all states have been evaluated and propagated to the transition states. Phone-level beam pruning takes place just before transcending to the word level by creating word instances out of end-of-word phones. Word-level pruning is conducted when the instantiating model enters into new word hypotheses.

Since the identity of a word is known with a much higher likelihood at the end of the word compared to its beginning (since by the end of the word, we have a much better idea what word was spoken), stricter pruning can be applied at word ends. Also, for large vocabulary applications it is beneficial to curb the fan-out caused by the language model list of possible next words. Therefore, the word-level threshold is usually tighter compared to the state and phone-level beams.

In a large vocabulary speech recognition application, the search space expansion is maximum at word ends, where a single path is propagated into multiple next words. If an N-gram back-off language model is used, the fan-out for each word end (i.e. the number of possible next words) is extremely large, since all the words in the vocabulary are possible candidates and need to be instantiated. Thus the search space expansion is even more severe. However, eventually very few of these generated paths survive the acoustic evaluation and subsequent pruning. The search space can be controlled by forbidding some of the word end paths to grow further. For accurate recognition, it is usually sufficient to propagate only a few word ends that are associated with the highest likelihood path scores. We refer to this as *maximum active word-end pruning*. The decoder can limit the number of words extended by keeping a sorted list of word-end paths at each frame and selecting only the top few for propagation. The rest of the paths can be pruned away.

The total memory requirements, as well as the amount of computation involved at each frame of the decoding process are directly related to the number of paths active at that frame. All variants of a path (i.e. paths that have the same word and model sequence, but possibly different time alignments) are associated with a single instantiation of the currently active model. Such an instance of a phone model can be defined in terms of its position in the search space. Each partial path or active hypothesis in the search space is identified in terms of the current node in the lexical tree it is associated with, the identity of the phone model being evaluated for that path and the last completely evaluated word in the word sequence defined by that path. Therefore all paths that have a common set of these coordinates are said to belong to the same instance. We can limit the number of these instances active at any time using an approach referred to as *maximum active phone model instance (MAPMI) pruning*.

The number of such instances active at a frame displays sizeable surges at word boundaries since a large number of new models are activated at a word boundary to accommodate the generation of new words. This can pose very severe requirements on the system memory. By setting an upper limit on the number of active phone model instances per frame, the memory usage (and the time required for the corresponding computations) can be effectively regulated [50]. At each frame the instances are sorted in order of the best score associated with each of them, and the instances that fall below the threshold score indicated by the upper bound on the number of instances are removed (i.e. all the paths associated with that instance are pruned off). All paths having an instance with the best score better than the threshold score are allowed to propagate.

Path Merging

If all hypotheses are allowed to grow independently, the search space expands exponentially as we step through the network or N-gram language model. In turn, the computational load on the decoder also increases exponentially with time. By sharing the evaluation of similar parts of different hypotheses the decoder can prevent this computational overload. Hypotheses with the same acoustic and linguistic context (as determined by the path history, the position in the lexical tree and the acoustic model index) have identical futures, and therefore can be merged into a single path with the information of the highest-scoring of all such merged paths propagated forward. This is an application of the principle of dynamic programming. Path merging takes place at all levels of the search hierarchy. The information that must be maintained for each path marker to support this step is known as an *instance* of the path marker.

At the word level, only the highest scoring of all the word ends corresponding to a particular instance is preserved. More specifically, at any frame, if more than one active path leads to the end of a word (as represented in the grammar or the word graph), such that they have the same acoustic context (same triphone), then only the best path among them is propagated further and the rest are deleted, returning valuable memory to the system. The lexical tree structure of the decoder framework automatically ensures that all the partial hypotheses represented here have identical linguistic context.

During word graph generation, the word-level markers keep track of multiple histories leading to the current word-end. Therefore when merging word-level paths, their histories are sorted by path score and assigned to the higher-scoring path. Thus even though only one path is propagated, multiple path histories are preserved through this sorted backpointer list. For phone and state levels, path merging takes place in a classical dynamic programming manner. At the state level, paths entering a state for a given instance are compared and only the best path is allowed to proceed further. Phone-level paths are can be merged when a path reaches the exit state of an acoustic models.

Word Graph Compaction

A word graph generated by N-gram decoding of an utterance often contains multiple instances of the same word sequences, each with a different alignment with respect to time. In other words, two arcs emerging from a node in the word graph may have the same word identity, but a different word-end time stamp. As a result, the total number of unique word sequences (based only on the word identity and ignoring the timing information) that can be derived from a word graph is only a fraction of the total number of derivable word sequences when the timing information is taken into account.

During acoustic or linguistic rescoring of the word graph, the graph is used only as a means to limit the number of possible word sequences. The timing information is rarely used for rescoring experiments. When this information is removed, many arcs of the word graph indicate essentially the same word sequence and need not be decoded individually [47]. Therefore, a word graph that provides only the unique word sequences is needed and it is fairly standard practice to ignore the timing information associated with the nodes. Word graph compaction is illustrated in Figure 16. The decoder uses a word graph compaction algorithm that reduces the original word graph into a word graph that preserves all the word hypotheses, yet merges all the duplicate arcs. Similar work has been described in [39, 63] in a more generalized framework of finite state machines (FSMs). This causes a significant reduction in the search space complexity (usually the word graph size drops by a factor of 2 to 5 in terms of the number of nodes and arcs) at minimal computational overhead.

4.4. System Architecture

Figure 17 shows a schematic representation of the control flow involved in the search process for the single-pass lexical-tree based decoder described in this paper. The decoder processes the input feature data frame by frame, evaluating the states of the active acoustic models and growing paths at the state level via transitions within the models, as well as at the phone level by transitioning from one phone to the next one in the pronunciation of the word. Various pruning criteria applied at different points in the search space attempt to constrain the search within a reasonable complexity.

Since bookkeeping is a large part of a search engine, we briefly describe the manner in which paths and their histories are represented within the system. The decoder uses a scoring information data structure, referred to as a marker, that maintains all information about the current path. The specific data structure used is shown in Figure 18. The current location in the search space is stored in this structure in terms of an instance definition, which is based on the identity of the most recently completed word, the current node in the lexical tree and the index of the acoustic model being evaluated. In addition, the marker contains other information such as the state index in the model, and pointers to the previous nodes on the current path.

At the state level in the search space, the markers corresponding to the same instance are compared and only the best-scoring of these is allowed to propagate. At the model and word levels in the search hierarchy, such comparisons require additional information. In the interim, the path markers at these levels are stored in various linked lists — each indexed by the corresponding model or word identity. The decoder loops over each linked list to process the paths stored in these, and passes markers back and forth among various lists as well as up and down the hierarchy. The newly created markers are stored in the appropriate linked lists.

The instance information, as well as the lexicon and the N-gram language model require frequent access and are reused extensively. Therefore, these are typically stored as hash table data structures for efficient lookup. Even then, language model lookup is an expensive process as the size of the LM increases tremendously with the N-gram order. Therefore, an alternative is to cache the language model scores of all the active words (since these are likely to remain active for the next few frames) in memory, and leave the rest of the language model on disk.

At each instantiation of an acoustic model (such as a triphone), a state-level path marker is projected from the previous phone or word-level marker and added to a state-level list of path markers. For each frame, the active states are evaluated only once. The state-level markers are compared and the best marker for each different instance of the state is projected to the next states as governed by the state transition probabilities (Viterbi decoding). The score for each state is stored locally and added to the projected path marker score. A marker exiting the model is added to the phone-level marker list and used to project the next triphone markers. Similarly, phone-level path markers at end of words are promoted to the word level and used to project paths into the subsequent words. An appropriate language model score is added for the path representing a word hypothesis once the identity of the current word on that path is known.

5. PERFORMANCE ANALYSIS

The recognition accuracy of a decoder is measured in terms of its word error rate (WER), which is the percentage of words recognized incorrectly. The WER is calculated by aligning the best hypothesis generated by the system with the reference word sequence, and then counting the number of misalignments. This quantity is often distributed into three classes of errors that provide some insight into the nature of complexity of the application:

- a *substitution* error refers to the case where the decoder misrecognizes a word in the reference sequence as another in the hypothesis.
- a *deletion* error occurs when the there is no word recognized corresponding to a word in the reference transcription.
- an *insertion* error corresponds to the case where the hypothesis contains an extra word that has no counterpart in the reference.

For example, if the original word sequence is “I like hard rock music”, and the best output of the decoder is “I uh like heart rock”, then the word “uh” is an insertion error, the word “heart” is a substitution in place of the word “hard”, and the word “music” from the reference transcription has been deleted.

While recognition accuracy is often the primary focus in the evaluation of a speech recognition system, there are many other factors that influence the performance of a system:

- *scalability*: Can the algorithm scale gracefully from small constrained tasks to large unconstrained tasks?
- *recognition accuracy*: How accurate is the best word sequence found by the system?
- *word graph accuracy*: Can the system generate alternate choices that contain the correct word sequence? How large must this list of choices be?
- *memory*: What memory is required to achieve optimal performance? How does performance vary with the amount of memory required?
- *run-time*: How many seconds of CPU time per second of speech are required (xRT) to achieve optimal performance? How does run-time vary with performance (run-time should decrease significantly as error rates increase)?

Of course, many of these factors are interdependent on a number of things, including the application. Hence, we present several experiments on two different corpora: the OGI Alphadigits [12] and SWITCHBOARD [22] tasks, using both word-internal as well as cross-word triphone models. The models were trained and evaluated using feature vectors that are comprised of 12 mel-frequency cepstral coefficients (mfcc) and energy, as well as their first and second-order temporal derivatives (delta and delta-delta coefficients respectively) as described in Section 2.2. All evaluations were conducted using a 333 MHz Pentium II processor with 512 MB of memory, the Sun Solaris x86 v2.6, and GNU's *gcc* compiler (v2.8.1).

Results are presented for both the network decoding and word graph rescoring modes. The relationship between various pruning and recognition performance is also discussed. Finally, an analysis of the search complexity is presented.

5.1. Alphadigits

The OGI Alphadigits Corpus (OGI-AD) is a database of telephone speech collected from approximately 3,000 subjects. The vocabulary consisted of the letters of the alphabet as well as the digits 0 through 9, a total of approximately 40 words. Each subject spoke a list of either 19 or 29 alphanumeric strings, each six words long. Each list was set up to balance phonetic context between all letter and digit pairs. In all, there were 1102 separate prompting strings which gave a balanced coverage of vocabulary and acoustic contexts. The language model for this application consisted of a fully connected graph (where any word can follow any other word) as shown in Figure 19. Thus the recognition system needs to use the network decoding mode to conduct a performance evaluation.

The OGI-AD task represents a small vocabulary application that involves making extremely precise acoustic distinctions. For example, the E-set words, which consist of all letters in the alphabet that contain the vowel “ih,” differ only by one phoneme (we refer to such words as minimal pairs). This is an application that is relatively easy to handle with a basic search algorithm, and one in which we can employ extremely complex acoustic models.

In Table 1, we present results for a system that uses a variety of acoustic models and a network grammar language model. It can be observed that as the complexity of the acoustic models increases (from context-independent to word-internal context-dependent to cross-word), the memory requirements and real-time rates increase as well. However, the quality of the acoustic models also improves, resulting in better recognition performance. Pruning thresholds typically need to adjusted with each type of acoustic model to achieve optimum performance and minimize run-time. Hence, different combinations of thresholds are used for each model set in Table 1.

The maximum memory requirement in Table 1 refers to the total amount of memory used by the recognition system — including the program space, lexicon, acoustic and language models, and decoding work space. The memory required to decode each test utterance varies with the length of the utterance. Longer utterances require a larger number of alternate paths, and hence an increased work space. Memory typically varies linearly with the length of the utterance. This is summarized in Figure 20, for both word-internal and cross-word context-dependent acoustic models and for word graph generation as well as rescoring modes of search.

5.2. Switchboard

The Switchboard (SWB) task consists of recognition of spontaneous conversational speech collected over standard telephone lines. It is currently one of the most challenging benchmarks for LVCSR systems, and an extremely hard task to decode. Some reasons for this are:

- *Acoustics*: telephone bandwidth speech with a variety of transducers and noisy channels;
- *Language Model*: an extremely large vocabulary (tens of thousands of words) with very few constraints on word sequences and frequent amounts of sentence restarts, dysfluencies, non-speech mouth noises and expressions, laughter, etc.
- *Pronunciation Variation*: the speaking style is spontaneous, so pronunciations vary widely from the expected forms contains in the lexicon (baseforms); many words are poorly articulated and smeared together (coarticulation and reduction)

As a result, word error rates on this task are typically in the mid-30% range with sophisticated systems, and in the mid-40% range for simple baseline systems. Often, it is prudent to generate word graphs with a simpler language model (e.g. a bigram) to conserve computing resources, and then run another pass of decoding to rescore these word graphs with more advanced acoustic and language models. Decoding with cross-word acoustic models is a challenge on this task, and a large amount of pruning is required to limit the search space from expanding indefinitely due to the large vocabulary size.

The decoder efficiency and accuracy is highly dependent on proper settings for all pruning thresholds. The optimal values of the various beam widths as well as the MAPMI and word-end limits need to be derived in an empirical fashion after careful experimentation. If pruning thresholds are set to allow only a few hypotheses to grow, the correct hypothesis may get pruned away early on in the decoding process and the word error rate degrades significantly. On the other hand, if the pruning is too relaxed, then a large number of competing paths are allowed to propagate.

The results of an evaluation that involves rescoring of word graphs derived from the development test set of the WS97 subset [20, 21] of the SWB corpus are presented in Table 2. Performance for word graph generation using a bigram language model and several types of context-dependent acoustic models is tabulated in Table 3. The maximum memory requirements and average computation time per second of speech data (xRT) are also presented for each case.

Since the search space is relatively simple for context-independent and word-internal context dependent models, the pruning in these cases can be tighter without any degradation in recognition performance. On the other hand, the search space complexity is very large when cross-word context-dependent acoustic models are used in conjunction with an N-gram language model. Correspondingly, pruning thresholds need to be modified to exercise a tight control on the search space expansion without significantly affecting the performance. Moreover, different types of pruning heuristics affect the search space in different fashions. In the next sections, we explore some of these relationships.

5.3. Beam Pruning

Since the state-level or global pruning beam affects the hypotheses at all levels, this threshold keeps the overall search space manageable by removing lower-ranked hypotheses (often, these are trivial variants of other hypotheses that remain within the beam). In the absence of a global pruning beam the number of new hypotheses generated per frame grows very rapidly and keeps monotonically increasing. The global beam is set to allow for the initial expansion where the decoder is uncertain about the initial words in a phrase, and thereafter limits the growth of these hypotheses once the search space is filled with active hypotheses.

The phone-level beam ensures that only those partial paths that have a good acoustic match with the input speech data are allowed to culminate in a word. Its effect is usually secondary to that of the global beam, and therefore it can be set to a tighter value compared to the state-level beam width. Only the word-end paths that survive the phone-level beam are updated with the language model score and projected to the word-level in the search space hierarchy.

In the absence of a word-level pruning beam, all word ends that survive the global pruning beam generate the corresponding next words. In a large vocabulary application this causes an explosion in the number of new paths created since the number of possible next words is very large. Word-level pruning prevents a huge number of follow-on words to be hypothesized by removing word-end paths that fall below the pruning threshold. A very narrow word-level beam, however, also destroys valid hypotheses and introduces recognition errors. Figure 21 shows the effect of beam pruning on recognition accuracy and computation time. The memory required by the system is directly proportional to computation time — the more memory the system needs, the more computation time is required to search through the hypotheses occupying this memory.

5.4. MAPMI Pruning

The maximum active phone model instance pruning also has a critical impact on the decoder performance. It has direct bearing on the memory usage of the decoder, since the number of number of active paths at each frame is directly proportional to the number of acoustic models active at that frame. If this number is allowed to grow unbounded, then at word boundaries the surge in the number of newly created paths can impose severe constraints on the memory requirements of the system.

This is especially critical in word graph generation and grammar decoding, as illustrated in Figure 22. For this example, even for utterances with a short duration, the required amount of memory approached 350 MB without MAPMI pruning. With MAPMI pruning, the number of active instances generated at each frame is reduced significantly, and an even smaller number of these is allowed to propagate. As a result, the memory required for the example utterance dropped to 135 MB. The decoding time, too, improved from 650 xRT to 330 xRT.

Figure 23 shows the overall effect of MAPMI pruning on decoder performance and efficiency during rescoring of word graphs. A very tight upper bound on the number of active instances can drastically affect the recognition accuracy, but beyond a certain range an increase in the MAPMI threshold only increases the memory and computation requirements by allowing a larger number of instances to exist. The effect of the MAPMI limit is considerably more pronounced for word graph generation.

6. SUMMARY

LVCSR systems have advanced significantly in recent years due largely to our ability to handle extremely large problem spaces in fairly small amounts of memory. The goal of this paper was to introduce readers to the problem of search, discuss in detail a typical implementation of a search engine, and demonstrate the efficacy of this approach on a range of problems. The approach presented here is nicely scalable across a wide range of applications. It is designed to address research needs, where a premium is placed on the flexibility of the system architecture, and the needs of application prototypers, who require decoding speeds approaching real-time without a great sacrifice in WER.

Future directions in search can be summarized in one word: real-time. Since the market for speech recognition technology has been exploding recently, one major area of focus for researchers is the development of real-time systems. With only minor degradations in performance (typically, no more than a 25% increase in WER), the systems described in this paper can be transformed into systems that operate at 10xRT or less. There are four active areas of research related to this problem. First, more intelligent pruning algorithms that prune the search space more heavily are required. Lookahead and N-best strategies at all levels of the system are key to achieving such large reductions in the search space. Second, multi-pass systems that perform a quick search using a simple system, and then rescore only the N-best resulting hypotheses using better models are very popular for real-time implementation. Third, since much of the computation in these systems is devoted to acoustic model processing, fast-matching strategies within the acoustic model are important. Finally, since Gaussian evaluation at each state in the system is a major issue consumer of CPU time, vector quantization-like approaches that enable one to compute only a small number of Gaussians per frame are proven to be successful.

In some sense, the Viterbi-based system presented here represents only one path through this continuum of recognition search strategies. One goal of the public domain system described in this paper is to accommodate as many alternate search approaches as possible, and to allow users to investigate these strategies from within a common recognition platform. To learn more about search, and speech recognition in general, we recommend several textbooks on speech recognition [29, 14] and some nice web sites devoted to these problems [24, 64].

REFERENCES

[1] F. Alleva, H. Hon, X. Huang, M. Hwang, R. Rosenfeld and R. Weide, “Applying SPHINX-II to the DARPA Wall Street Journal CSR Task,” *Proceedings of DARPA Speech and Natural Language Processing Workshop*, pp. 393-398, Harriman, New York, USA, February 1992.

[2] B.S. Atal, “Effectiveness of Linear Prediction Characteristics of the Speech Wave for automatic Speaker Identification and Verification,” *Journal of the Acoustical Society of America*, Vol. 55, No. 6, pp. 1304-1312, June 1974.

[3] R.L. Bahl et al, “Large Vocabulary Natural Language Continuous Speech Recognition,” *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, pp. 465-467, Glasgow, Scotland, May 1989.

[4] L. Bahl, P.F. Brown, P.V. de Souza and R.L. Mercer, “A Tree-Based Statistical Language Model for Natural Language Speech Recognition,” *IEEE Transactions on Acoustics, Speech and Signal Processing*, Vol. 37, No. 7, pp. 1001-1008, July 1989.

[5] L.R. Bahl, P.V. de Souza, P.S. Gopalakrishnan, D. Nahamoo and M.A. Pichney, “Context Dependent Modeling of Phones in Continuous Speech Using Decision Trees,” *Proceedings of DARPA Speech and Natural Language Processing Workshop*, pp. 264-270, Pacific Grove, California, USA, February 1991.

[6] L.E. Baum, “An Inequality and Associated Maximization Technique in Statistical Estimation for Probabilistic Functions of Markov Processes,” *Inequalities*, Vol. 1, pp. 1-8, 1972.

[7] S. Browning, M. Russell and S. Downey, “Phoneme Decision Tree Construction for Automatic Speech Recognition,” DRA Memorandum No. 4666, Defence Research Agency, 1993.

[8] J.K. Chen and F.K. Soong, “An N-Best Candidates-Based Discriminative Training for Speech Recognition Applications,” *IEEE Transactions on Speech and Audio Processing*, Vol. 2, No. 1, Part II, pp. 206-216, January 1994.

[9] G.F. Chollet and C. Gagnoulet, “On the Evaluation of Speech Recognizers and Databases Using a Reference System,” *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, pp. 2026-2029, Paris, France, 1982.

[10] Y.L. Chow and R.M. Schwartz, “The N-Best Algorithm: An Efficient Procedure for Finding Top N Sentence Hypotheses,” *Proceedings of DARPA Speech and Natural Language Processing Workshop*, pp. 199-202, Cape Cod, Massachusetts, USA, October 1989.

[11] Y.L. Chow, M. Ostendorf-Dunham, O.A. Kimball, M.A. Krasner, G.F. Kubala, J. Makhoul, S. Roukos and R.M. Schwartz, “BYBLOS: The BBN Continuous Speech Recognition

System," *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, pp. 89-92, Dallas, Texas, USA, April 1987.

[12] R. Cole et al, "Alphadigits Corpus," *http://www.cse.ogi.edu/CSLU/corpora/alphadigit*, Center for Spoken Language Understanding, Oregon Graduate Institute, Portland, Oregon, USA, 1997.

[13] S.B. Davis and P. Mermelstein, "Comparison of Parametric Representations for Monosyllabic Word Recognition in Continuously Spoken Sentences," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol. 28, No. 4, pp. 357-366, August 1980.

[14] J.R. Deller, J.G. Proakis and J.H.L. Hansen, *Discrete-Time Processing of Speech Signals*, Macmillan Publishing, New York, USA, 1993.

[15] N. Deshmukh, A. Ganapathiraju, J. Hamaker and J. Picone, "Large Vocabulary Conversational Speech Recognition," *http://www.isip.msstate.edu/projects/speech/*, Mississippi State University, Mississippi State, Mississippi, USA, May 1999.

[16] N. Deshmukh, J. Picone and Y.H. Kao, "Efficient Search Strategies in Hierarchical Pattern Recognition Systems," *Proceedings of 27th IEEE Southeastern Symposium on System Theory*, pp. 88-91, Mississippi State, Mississippi, USA, 1995.

[17] V. Digalakis, P. Monaco and H. Murveit, "Genomes: Generalized Mixture Tying in Continuous Hidden Markov Model-Based Speech Recognizers," *IEEE Transactions on Speech and Audio Processing*, Vol. 4, No. 4, pp. 281-289, July 1996.

[18] V.V. Digalakis, M. Ostendorf and J.R. Rohlicek, "Fast Algorithms for Phone Classification and Recognition Using Segment-Based Models," *IEEE Transactions on Signal Processing*, Vol. 40, No. 12, pp. 2885-2896, December 1992.

[19] S. Furui, "Cepstral Analysis Technique for Automatic Speaker Verification," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol. 29, No. 2, pp. 254-272, April 1981.

[20] A. Ganapathiraju, V. Goel, J. Picone, A. Corrada, G. Doddington, K. Kirchoff, M. Ordowski and B. Wheatley, "Syllable — A Promising Recognition Unit for LVCSR," *Proceedings of the IEEE Automatic Speech Recognition and Understanding Workshop*, pp. 207-214, Santa Barbara, California, USA, December 1997.

[21] A. Ganapathiraju et al, "WS97 Syllable Team Final Report," *Proceedings of the 1997 LVCSR Summer Research Workshop*, Center for Language and Speech Processing, Johns Hopkins University, Baltimore, Maryland, USA, December 1997.

[22] J. Godfrey, E. Holliman and J. McDaniel, "SWITCHBOARD: Telephone Speech Corpus for Research and Development," *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, Vol. I, pp. 517-520, San Francisco, California,

USA, March 1992.

[23] P.S. Gopalakrishnan, L.R. Bahl and R.L. Mercer, “A Tree Search Strategy for Large-Vocabulary Continuous Speech Recognition,” *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, pp. 572-575, Detroit, Michigan, USA, May 1995.

[24] J. Hamaker and B. Brown, “Experiments,” *http://www.isip.msstate.edu/projects/speech/experiments/index.html*, Mississippi State University, Mississippi State, Mississippi, USA, May 1999.

[25] H. Hermansky, “Perceptual Linear Predictive (PLP) Analysis of Speech,” *Journal of the Acoustical Society of America*, Vol. 87, No. 4, pp. 1738-1752, April 1990.

[26] X.D. Huang, F. Alleva, H.W. Hon, M.Y. Hwang, K.F. Lee and R. Rosenfeld, “The SPHINX-II Speech Recognition System: An Overview,” *Computer, Speech and Language*, Vol. 7, No. 2, pp. 137-148, April 1993.

[27] X.D. Huang, H.W. Hon, M.Y. Hwang and K.F. Lee, “A Comparative Study of Discrete, Semi-Continuous and Continuous Hidden Markov Models,” *Computer Speech and Language*, Vol. 7, No. 4, pp. 359-368, October 1993.

[28] M. Jardino, “Multilingual Stochastic N-Gram Class Language Models,” *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, Vol. 1, pp. 161-163, Atlanta, Georgia, USA, May 1996.

[29] F. Jelinek, *Statistical Methods for Speech Recognition*, MIT Press, Cambridge, Massachusetts, USA, 1997.

[30] F. Jelinek, “Up From Trigrams! The Struggle for Improved Language Models,” *Proceedings of the 2nd European Conference on Speech Communication and Technology (Eurospeech)*, pp. 1037-1040, Genova, Italy, September 1991.

[31] F. Jelinek, R.L. Mercer and S. Roukos, “Principles of Lexical Modeling for Speech Recognition,” from S. Furui and M.M. Sondhi (Editors), *Advances in Speech Signal Processing*, pp. 651-699, Marcel Dakker Inc., New York, New York, USA, 1992.

[32] R. Kuhn and R. de Mori, “A Cache Based Natural Language Model for Speech Recognition,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 14, pp. 691-692, June 1992.

[33] J. Kupiec, “Probabilistic Models of Short and Long Distance Word Dependencies in Running Text,” *Proceedings of the ARPA Workshop on Speech and Natural Language*, pp. 290-295, Philadelphia, Pennsylvania, USA, February 1989.

[34] J. Lafferty, D. Sleator and D. Temperley, “Grammatical Trigrams: A Probabilistic Model of

Link Grammar," *Proceedings of the AAAI Fall Symposium on Probabilistic Approaches to Natural Language*, pp. 89-97, Cambridge, Massachusetts, USA, October 1992.

[35] R. Lau, R. Rosenfeld and S. Roukos, "Trigger-Based Language Models: A Maximum Entropy Approach," *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, Vol. 2, pp. 45-48, Minneapolis, Minnesota, USA, April 1993.

[36] K.F. Lee and F. Alleva, "Continuous Speech Recognition," *Advances in Speech Signal Processing*, edited by S. Furui and M.M. Sondhi, pp. 651-699, Marcel Dakker Inc., 1992.

[37] K.F. Lee and H.W. Hon, "Large-Vocabulary Speaker-Independent Continuous Speech Recognition," *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, pp. 123-126, Dallas, Texas, USA, April 1987.

[38] B.T. Lowerre, "The HARPY Speech Recognition System", Ph.D. Thesis, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA, 1976.

[39] M. Mohri, M. Riley, D. Hindle, A. Ljolje and F. Pereira, "Full Expansion of Context-Dependent Networks in Large Vocabulary Speech Recognition," *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 665-668, Seattle, Washington, USA, May 1998.

[40] H. Murveit, J. Butzberger, V. Digalakis and M. Weintraub, "Large Vocabulary Dictation Using SRI's Decipher Speech Recognition System: Progressive Search Techniques," *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, Vol. II, pp. 319-322, Minneapolis, Minnesota, USA, April 1993.

[41] H. Murveit, P. Monaco, V. Digalakis and J. Butzberger, "Techniques to Achieve an Accurate Real-Time Large-Vocabulary Speech Recognition System," *Proceedings of the ARPA Human Language Technology Workshop*, pp. 368-373, Austin, Texas, USA, March 1995.

[42] B.F. Necioglu, M. Ostendorf and J.R. Rohlicek, "A Bayesian Approach to Speaker Adaptation for the Stochastic Segment Model," *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, Vol. I, pp. 437-440, San Francisco, California, March 1992.

[43] H. Ney, "Dynamic Programming Parsing for Context Free Grammars in Continuous Speech Recognition," *IEEE Transactions on Signal Processing*, Vol. SP-39, No. 2, pp. 336-341, February 1991.

[44] H. Ney, U. Essen and R. Kneser, "On Structuring Probabilistic Dependencies in Stochastic Language Modeling," *Computer Speech and Language*, Vol. 8, No. 1, pp. 1-38, January 1994.

[45] H. Ney, D. Mergel, A. Noll and A. Paesler, “Data Driven Organization of the Dynamic Programming Beam Search for Continuous Speech Recognition,” *IEEE Transactions on Signal Processing*, Vol. SP-40, No. 2, pp. 272-281, February 1992.

[46] H. Ney and S. Ortmanns, “Dynamic Programming Search for Continuous Speech Recognition,” to appear in the *IEEE Signal Processing Magazine*, September 1999.

[47] L. Nguyen, R. Schwartz, F. Kubala and P. Placeway, “Search Algorithms for Software-Only Real-Time Recognition with Very Large Vocabularies,” *Proceedings of DARPA Human Language Technology Workshop,* pp. 91-95, Princeton, New Jersey, USA, March 1993.

[48] L. Nguyen, R. Schwartz, Y. Zhao and G. Zavaliagkos, “Is N-Best Dead?,” *Proceedings of DARPA Human Language Technology Workshop*, pp. 386-388, Plainsboro, New Jersey, USA, March 1994.

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

[50] J.J. Odell, V. Valtchev, P.C. Woodland and S.J. Young, “A One Pass Decoder Design for Large Vocabulary Recognition,” *Proceedings of ARPA Human Language Technology Workshop,* pp. 405-410, Princeton, New Jersey, USA, March 1994.

[51] D. Pallett et al, “1995 Benchmark Tests for the ARPA Spoken Language Program,” *Proceedings of the ARPA Spoken Language Systems Technology Workshop*, Harriman, New York, USA, February 1996.

[52] D.B. Paul, “Algorithms for an Optimal A* Search and Linearizing the Search in the Stack Decoder,” *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, pp. 693-696, Toronto, Canada, 1991.

[53] D.B. Paul, “An Efficient A* Stack Decoder Algorithm for Continuous Speech Recognition with a Stochastic Language Model,” *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing,* Vol. I, pp. 405-409, San Francisco, California, USA, March 1992.

[54] D.B. Paul, “The Lincoln Large-Vocabulary Stack Decoder Based HMM CSR,” *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, pp. 374-379, Minneapolis, Minnesota, USA, April 1993.

[55] J. Picone, “Continuous Speech Recognition Using Hidden Markov Models,” *IEEE Acoustics, Speech, and Signal Processing Magazine*, Vol. 7, no. 3, pp. 26-41, July 1990.

[56] J. Picone, “Signal Modeling Techniques in Speech Recognition,” *Proceedings of the IEEE*, Vol. 81, No. 9, pp. 1215-1247, September 1993.

[57] L.R. Rabiner and B.H. Juang, *Fundamentals of Speech Recognition*, Prentice Hall,

Englewood Cliffs, New Jersey, USA, 1993.

[58] A.J. Robinson, “An Application of Recurrent Nets to Phone Probability Estimation,”, *IEEE Transactions on Neural Networks*, Vol. 5, No. 2, pp. 298-305, March 1994.

[59] A.J. Robinson and F. Fallside, “A Recurrent Error Propagation Network Speech Recognition System,” *Computer Speech & Language*, Vol. 5, No. 3, pp. 259-274, July 1991.

[60] J. Schalkwyk, D. Colton, and M. Fanty, “The CSLUsh Toolkit for Automatic Speech Recognition,” *Technical Report CSLU-011-1995*, Center for Spoken Language Understanding, Oregon Graduate Institute of Science and Technology, Portland, Oregon, USA, December 1995.

[61] R. Schwartz and S. Austin, “A Comparison of Several Approximate Algorithms for Finding Multiple (N-Best) Sentence Hypotheses,” *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, pp. 701-704, Toronto, Canada, 1991.

[62] R.M. Schwartz and S. Austin, “Efficient, High-Performance Algorithms for N-Best Search,” *Proceedings of DARPA Speech and Natural Language Processing Workshop*, pp. 6-11, Hidden Valley, Pennsylvania, USA, June 1990.

[63] R.D. Sharp, E. Bocchieri, C. Castillo, S. Parthasarathy, C. Roth, M. Riley and J. Rowland, “The Watson Speech Recognition Engine,” *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 4065-4068, Munich, Germany, May 1997.

[64] K. Shobaki, “Learn About and Experience Spoken Language Technology,” *http://cslu.cse.ogi.edu/learn*, Oregon Graduate Institute of Science and Technology, Portland, Oregon, USA, May 1999.

[65] F.K. Soong and E.F. Huang, “A Tree-Trellis Based Fast Search for Finding the N Best Sentence Hypotheses in Continuous Speech Recognition,” *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, pp. 705-708, Toronto, Canada, 1991.

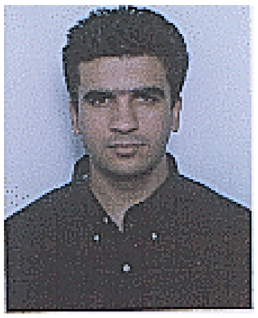
[66] T. Takezawa, T. Morimoto, Y. Sagisaka, N. Campbell, H. Iida, F. Sugaya, A. Yokoo and S. Yamamoto, “A Japanese-to-English Speech Translation System: ATR-MATRIX”, *Proceedings of the 5th International Conference on Spoken Language Processing*, Vol. 6, pp. 2779-2782, Sydney, Australia, November 1998.

[67] A.J. Viterbi, “Error Bounds for Convolutional Codes and an Asymptotically Optimal Decoding Algorithm,” *IEEE Transactions on Information Theory*, Vol. IT-13, pp. 260-269, April 1967.

[68] P. Woodland et al, *HTK Version 1.5: User, Reference and Programmer Manuals*, Entropic

Research Labs Inc., Cambridge, UK, 1995.

[69] S.J. Young and P.C. Woodland, "State Clustering in HMM-Based Continuous Speech Recognition," *Computer Speech and Language*, Vol. 8, No. 4, pp. 369-384, October 1993.



Neeraj Deshmukh is a Ph.D. student in the Department of Electrical and Computer Engineering at Mississippi State University. He received his B.Tech. from the Indian Institute of Technology, Mumbai, India in 1993, and his M.S. from Boston University in 1995, both in Electrical Engineering. He is currently involved in the development of a state-of-the-art public-domain large vocabulary speech recognition toolkit. Mr. Deshmukh's research interests span hierarchical search algorithms, statistical language modeling and pronunciation modeling techniques for speech recognition and synthesis. His previous experience includes object recognition in images and classification of letters to phones for automatic generation of proper noun pronunciations. Mr. Deshmukh is a member of Eta Kappa Nu and a student member of the IEEE, and was awarded the 1999 College of Engineering Graduate Student Research Award at Mississippi State University.

Aravind Ganapathiraju is a Ph.D. student in the Department of Electrical and Computer Engineering at Mississippi State University. He received his B.S. from REC, Trichy, India in Electronics and Communication Engineering in 1995 and his M.S. in Computer Engineering from Mississippi State University in 1997. He currently leads the Core Speech Technology team developing a state-of-the-art public-domain speech recognition system. Mr. Ganapathiraju's research interests lie in the development of discriminative algorithms for better acoustic modeling. He has previously worked on syllable-based speech recognition systems. Mr. Ganapathiraju is a member of Eta Kappa Nu and is a student member of the IEEE.

Joseph Picone received his Ph.D. in Electrical Engineering from Illinois Institute of Technology in 1983. He is currently an Associate Professor in the Department of Electrical and Computer Engineering at Mississippi State University, where he also directs the Institute for Signal and Information Processing. He has previously been employed by Texas Instruments and AT&T Bell Laboratories. His primary research interest currently is the development of public domain speech recognition technology. Dr. Picone is a Senior Member of the IEEE and a registered Professional Engineer. He has also served in several capacities with the IEEE, including an associate editor of this journal. Dr. Picone has published more than 90 papers in the area of speech processing and has been awarded 8 patents.

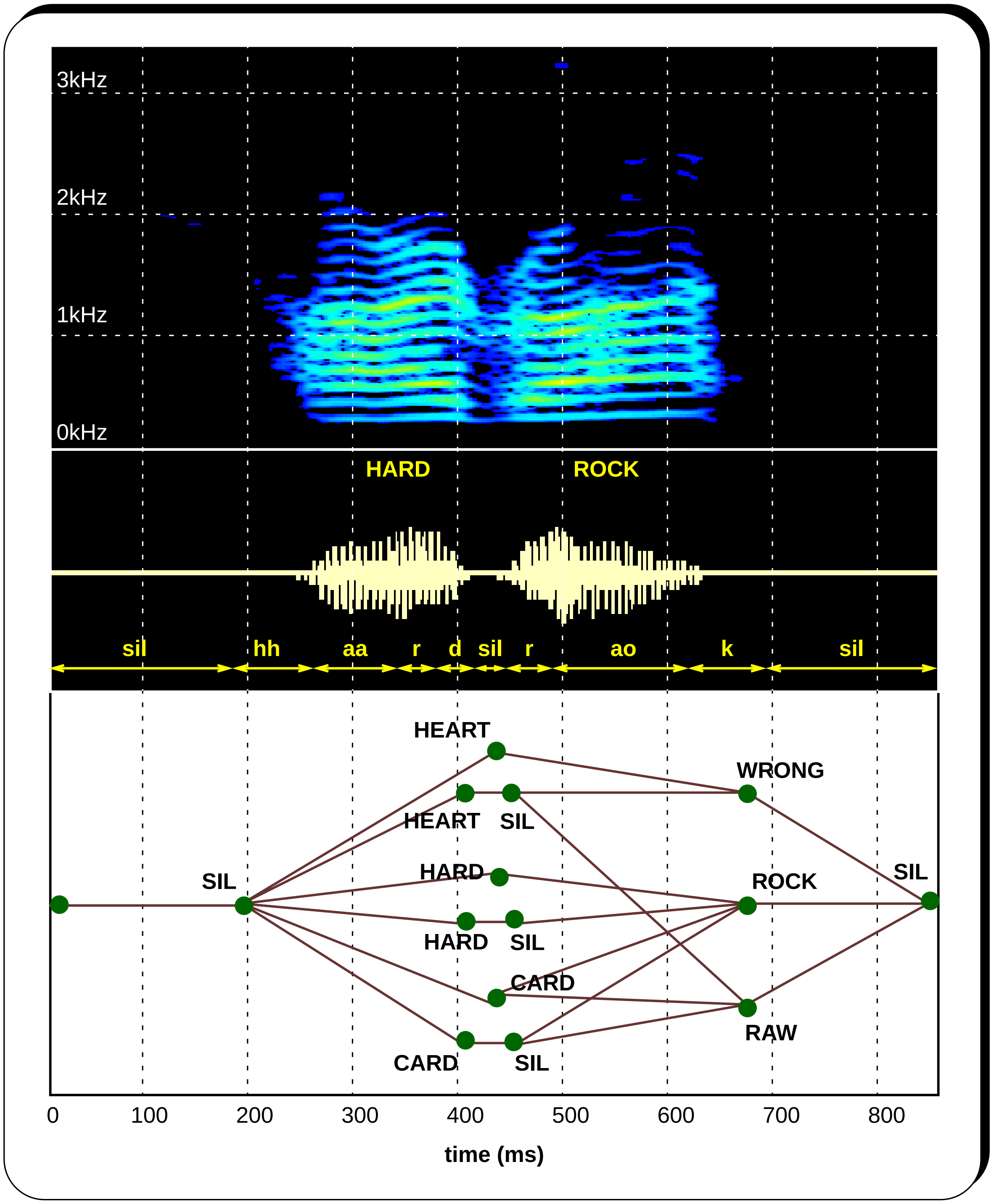


Figure 1. An example illustrating why the decoding problem is so difficult. The search algorithm produces a segmentation of the utterance as well as an estimate of the words that were spoken. Since pauses often do not occur between words in spontaneous speech, we must allow pauses to be optional at every point in the network describing allowable word sequences. This is one example of a constraint that makes the traditional dynamic programming solution much more difficult to implement.

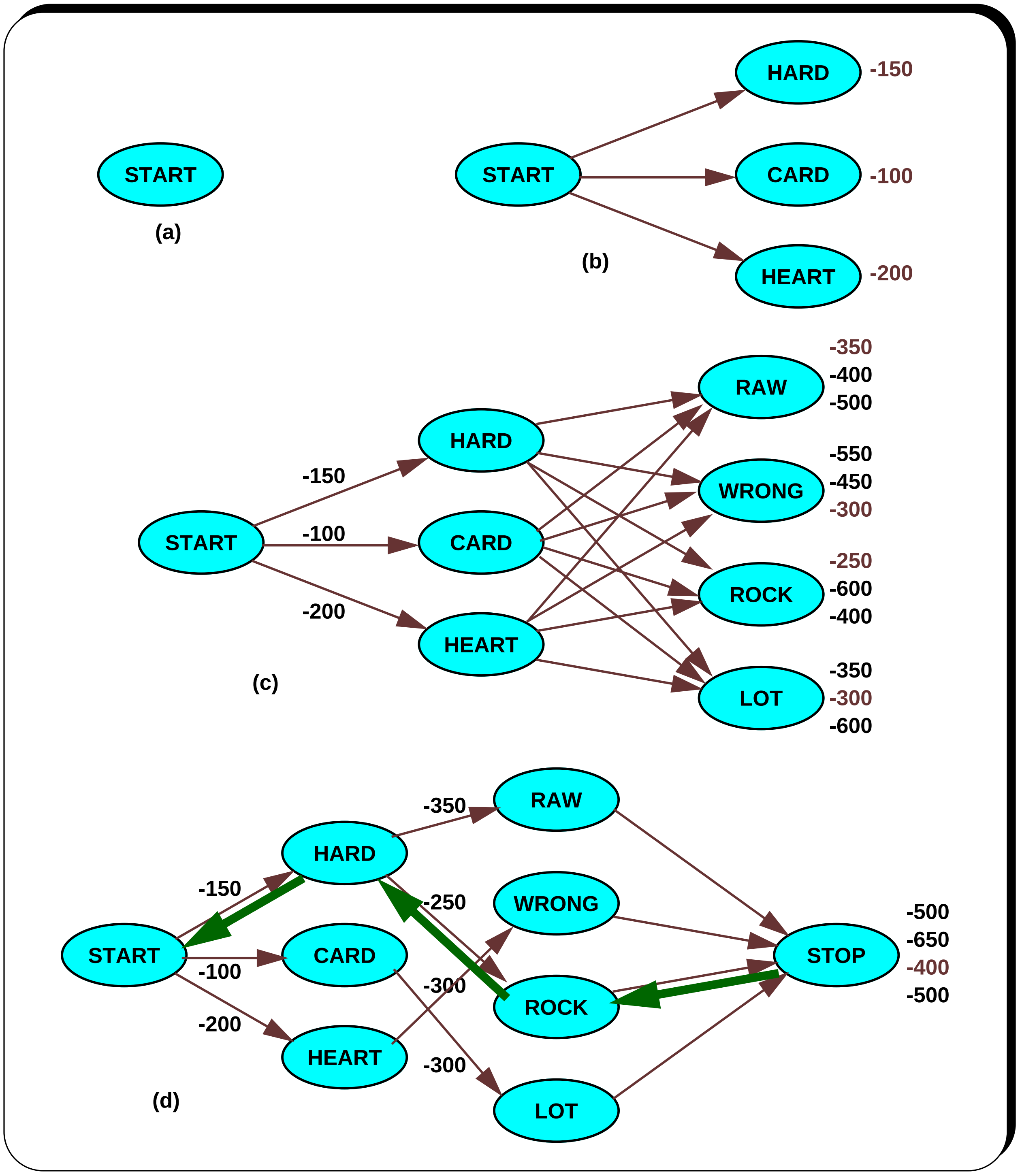


Figure 2. An overview of the principle of dynamic programming illustrated over a small word graph. (a) The start node. (b) For all current nodes, make transitions to all possible nodes keeping track of the history node. (c) At each node at the end of a path, sort all the incoming paths by score. Keep only the highest scoring path and delete the rest. (d) In the end, follow back the history of the stop node with the highest path score to generate the best hypothesis. Note that the final best path need not be the best at an earlier point in the network.

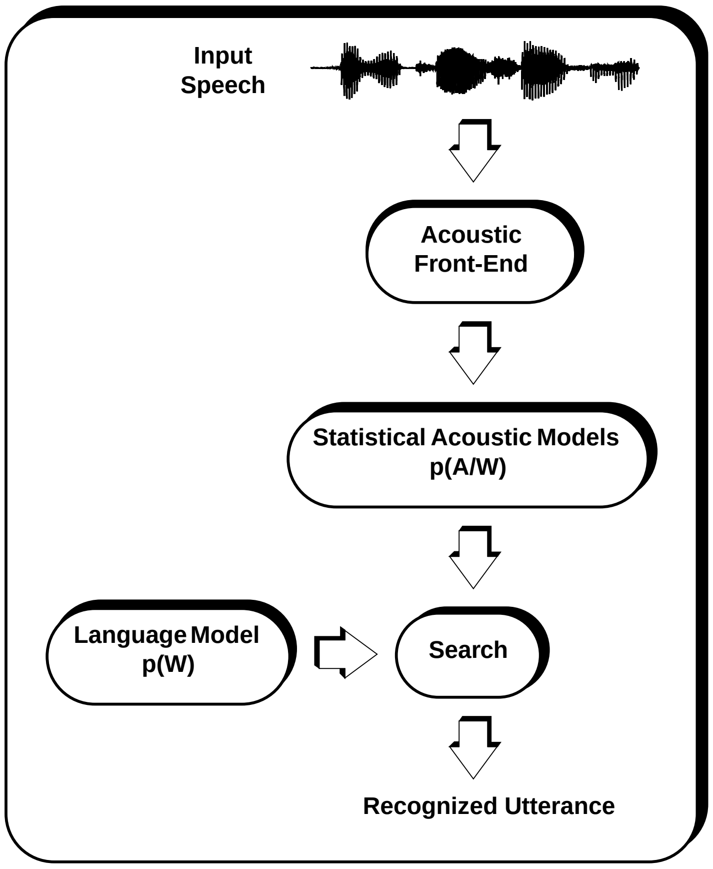


Figure 3. Schematic overview of a statistical speech recognition system

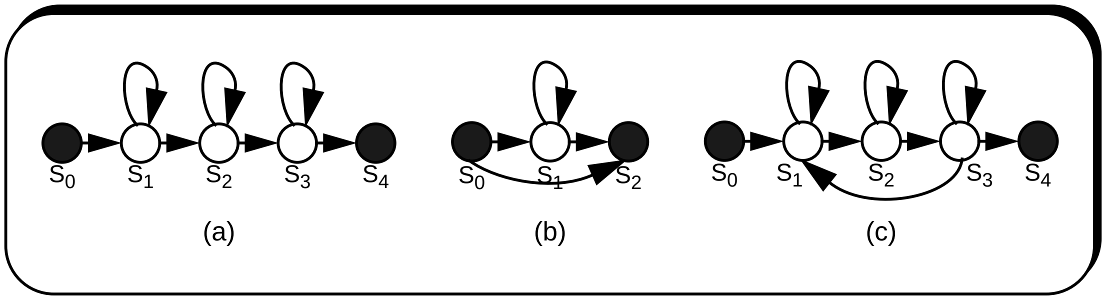


Figure 4. Some typical HMM topologies used for acoustic modeling in large vocabulary speech recognition: (a) typical triphone, (b) short pause, and (c) silence. The shaded states denote the start and stop states for each model.

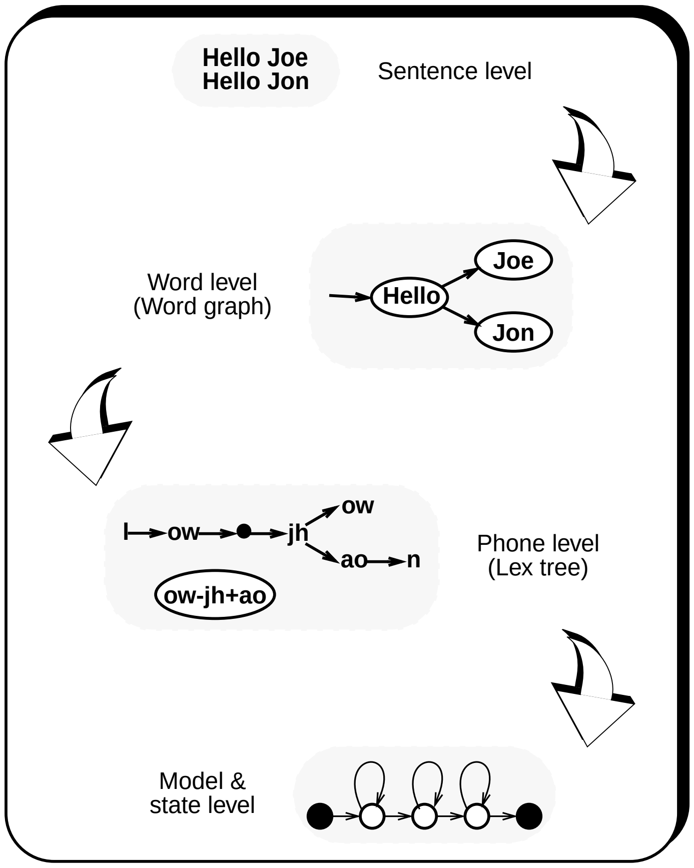


Figure 5. Hierarchical representation of the search space.

```
1. Generate a list of all states $S(t)$ for each frame $t$ of the utterance.
2. Initialize each list by setting the probability of the initial state to 1 and the rest to 0.
3. For each state $s \in S(t)$
       For each possible transition from $s$ to some state $s' \in S(t)$
           • If the list for $s'$ is uninitialized, initialize it with the transition score $p(s'/s)$ and a
             back-pointer to $s$.
           • Else update the score for $s'$ only if this transition gives a better path score.
4. Repeat step 3 with $t = t + 1$.
5. If $t = N$, the utterance duration, then trace back to get the best path.
```

Figure 6. An outline of the Viterbi search algorithm (also see Figure 2).

1. Pop the best partial hypothesis from the stack
2. Apply acoustic and language model fast matches (computationally cheap methods for reducing the number of word extensions) to shortlist the candidate next words.
3. Apply acoustic and language model detailed matches (more accurate but computationally expensive methods) to candidate words.
4. Choose the most likely next word and update all hypotheses.
5. Insert surviving new hypotheses into the stack and reorder stack.
6. Go to 1 if it is not the end of the utterance.
7. Output the hypothesis on the top of the stack.

Figure 7. Simple overview of the stack decoding algorithm.

Rank	Hypotheses	Likelihood
1	SILENCE HARD ROCK SILENCE	-5880.11
2	SILENCE HARD WRONG SILENCE	-5905.17
3	SILENCE HARD RAW SILENCE	-5906.32
4	SILENCE A HARD ROCK SILENCE	-5920.68
5	SILENCE HARD ROT SILENCE	-5922.05
6	SILENCE HARD RON SILENCE	-5923.69
7	SILENCE CARD WRONG SILENCE	-5924.51
8	SILENCE CARD RAW SILENCE	-5925.66
9	SILENCE YOU HARD ROCK SILENCE	-5928.95
10	SILENCE HART WRONG SILENCE	-5929.97
11	SILENCE HEART WRONG SILENCE	-5930.42
12	SILENCE ARE HARD ROCK SILENCE	-5936.11
13	SILENCE CARD ROCK SILENCE	-5936.86
14	SILENCE OF HARD ROCK SILENCE	-5937.56
15	SILENCE CARD ROT SILENCE	-5941.39
16	SILENCE CARD RON SILENCE	-5943.03
17	SILENCE A HARD WRONG SILENCE	-5945.74
18	SILENCE PART WRONG SILENCE	-5946.36
19	SILENCE HART ROT SILENCE	-5946.85
20	SILENCE A HARD RAW SILENCE	-5946.89

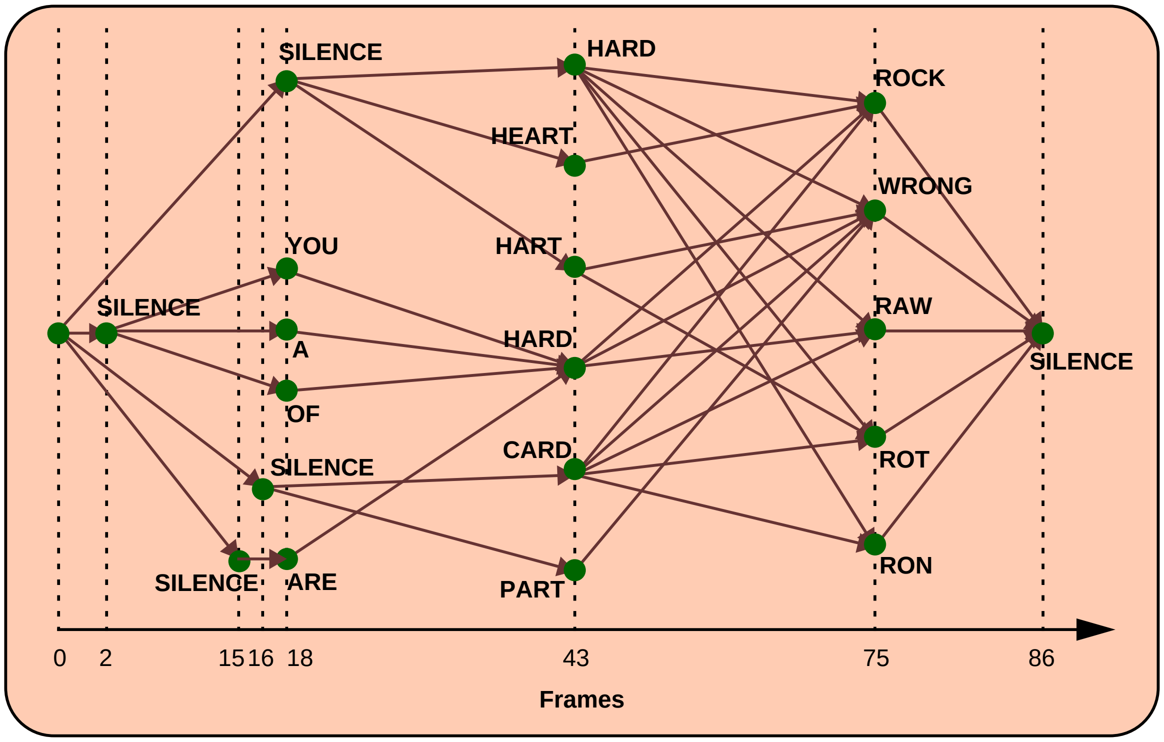


Figure 8. An example of the N-best list of hypotheses generated for a simple utterance, and the resulting word graph with N equal to 20. Note that most of the paths are almost equally probable, and only minor variants of each other in terms of segmentation. This indicates the severity of the acoustic confusability in spontaneous, conversational speech recognition.

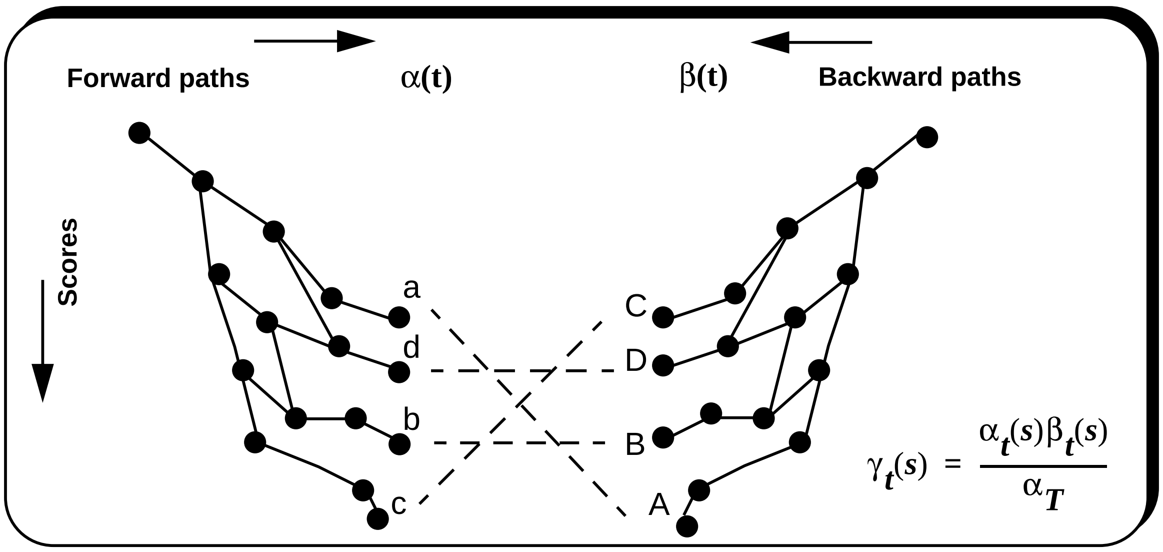


Figure 9. Forward-backward search — the combined score is the normalized product of the forward and backward path scores.

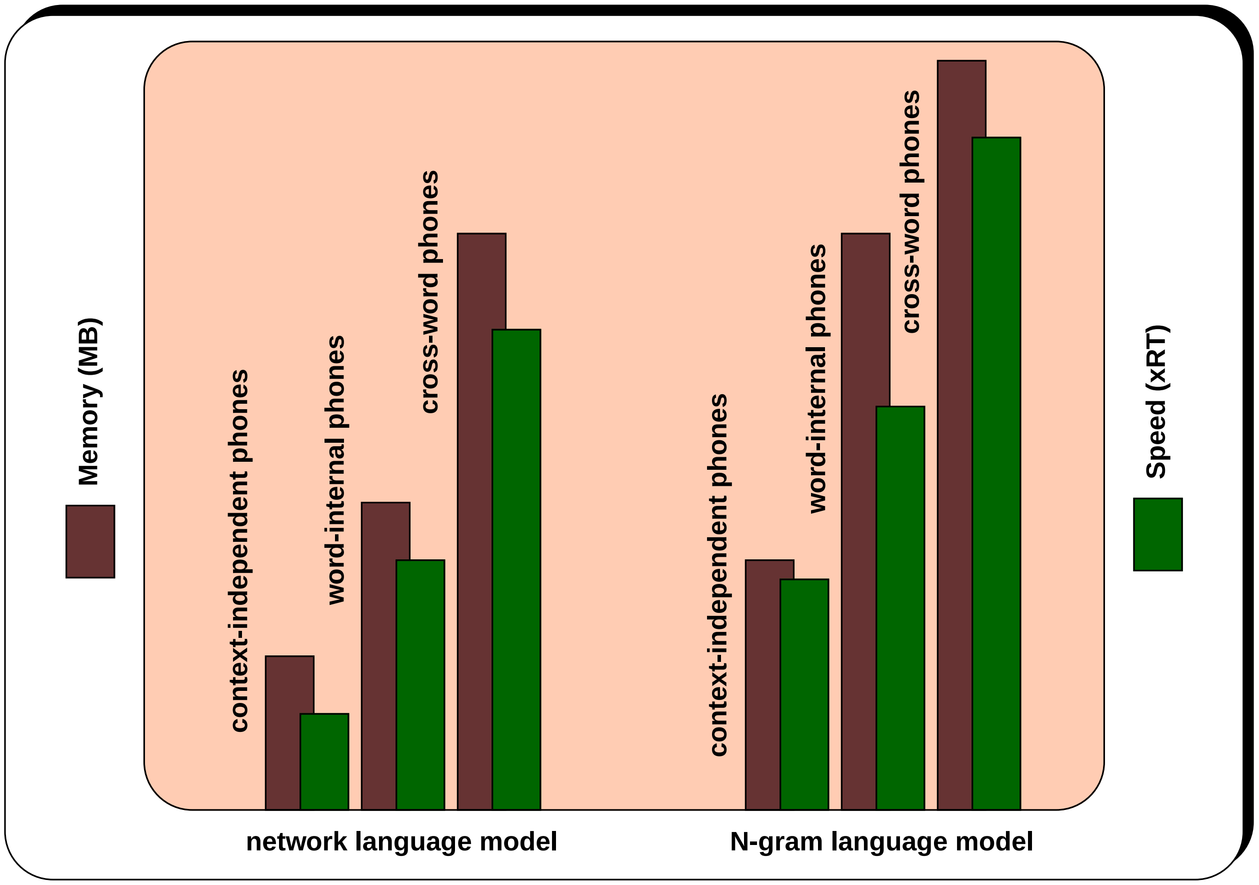
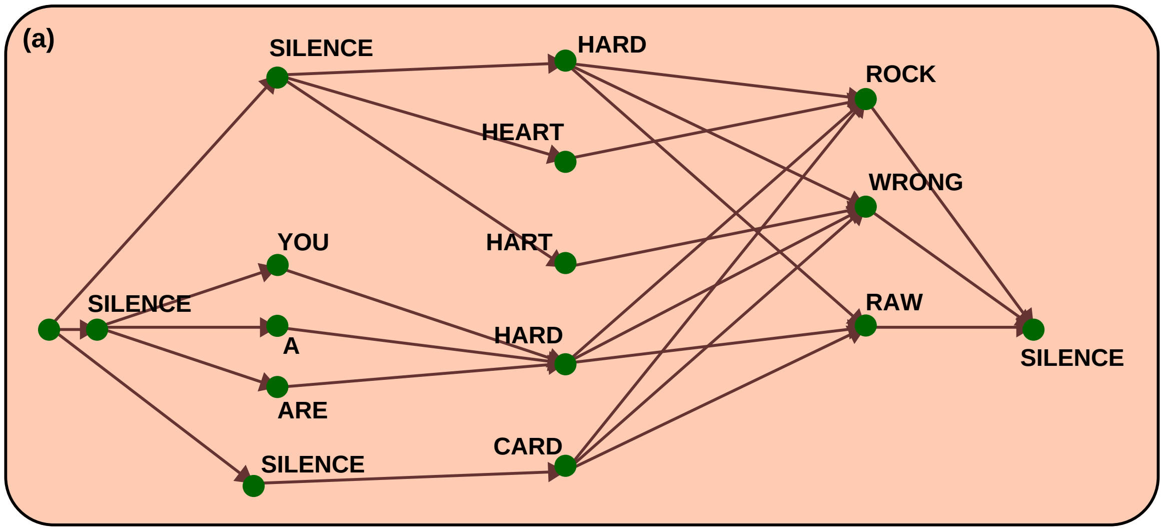


Figure 10. An overview of the relative complexity of the search problem for large vocabulary conversational speech recognition that shows the impact of various types of acoustic and language models.

(a)



(b)

SILENCE	sil	HART	h aa r t
A	ax	HEART	h aa r t
A	ey	RAW	r ao
ARE	aa r	ROCK	r ao k
CARD	k aa r d	WRONG	r ao ng
HARD	h aa r d	YOU	y uw

(c)

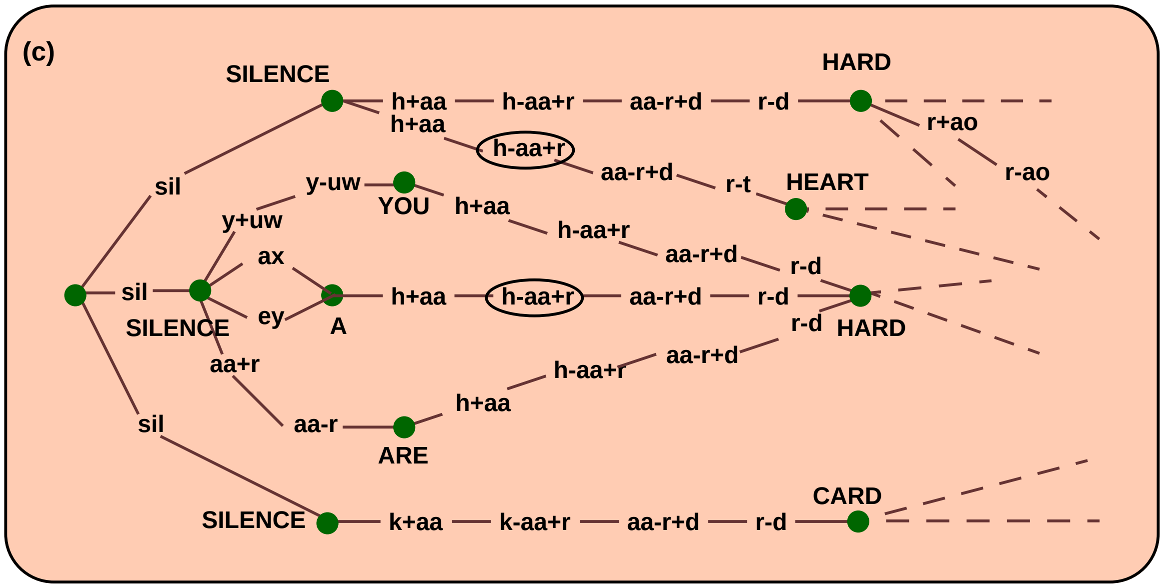


Figure 11. An example of network decoding using word-internal context-dependent models. (a) The word network providing linguistic constraints (b) The pronunciation lexicon for the words involved (c) The network expanded using the corresponding word-internal triphones derived from the pronunciations of the words. Note that every pronunciation of a word needs to be treated as a different word (e.g. “A”), and every instance of a word needs to be hypothesized separately. The two circled triphones represent different instances as they belong to two different words.

1. SILENCE YOU HARD ROCK...
2. SILENCE CARD ROCK...

Figure 12. While the two paths above have very different origins, they will be merged at the end of the word “rock” in case of a bigram language model being used for decoding. Since for the bigram the paths can be uniquely separated only based on the previous complete word, these paths can be merged at the word-level in the search hierarchy.

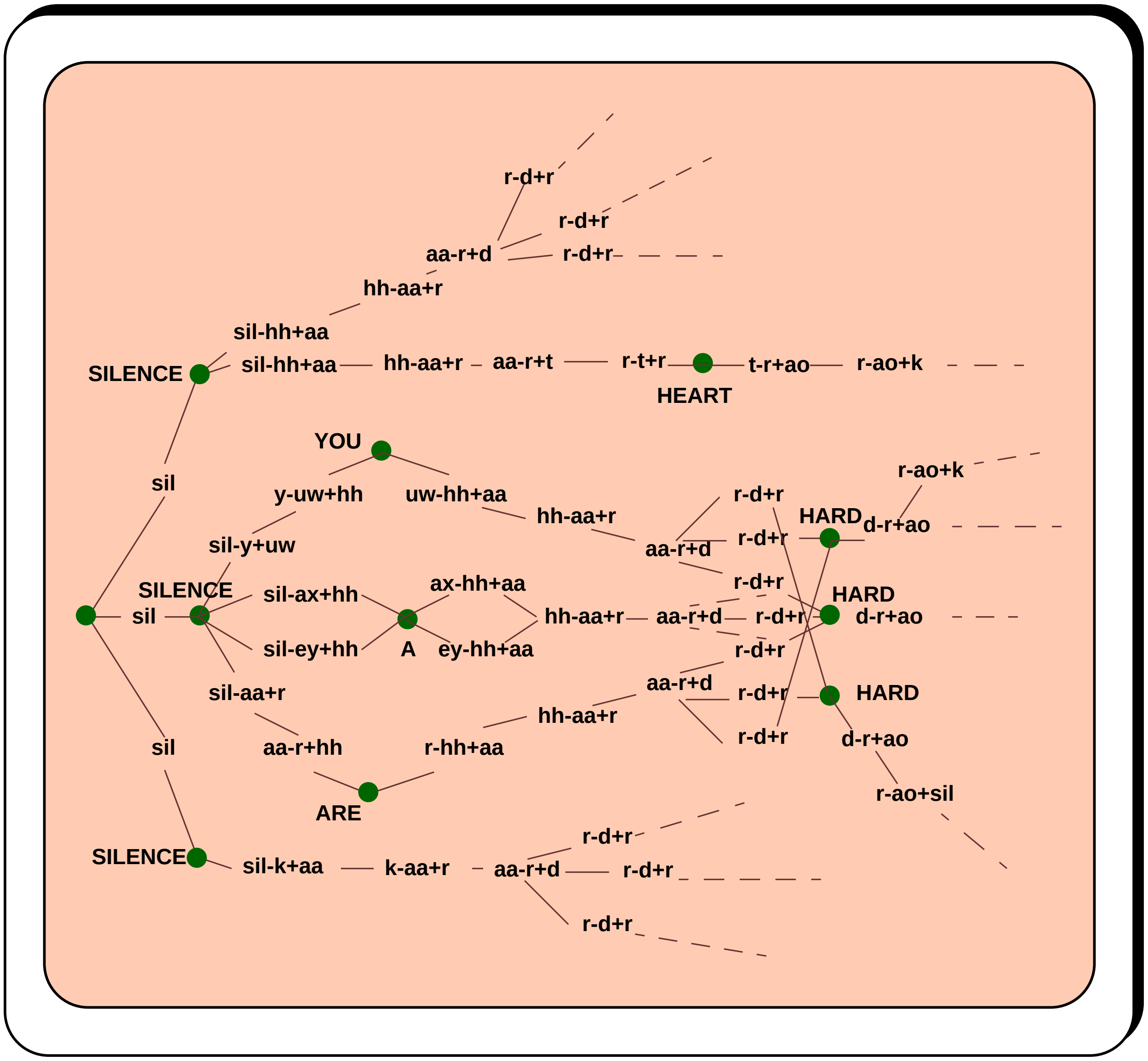


Figure 13. A small part of the expanded network from Figure 11 using cross-word triphones. Note the explosion in the number of paths at the end and start of each word.

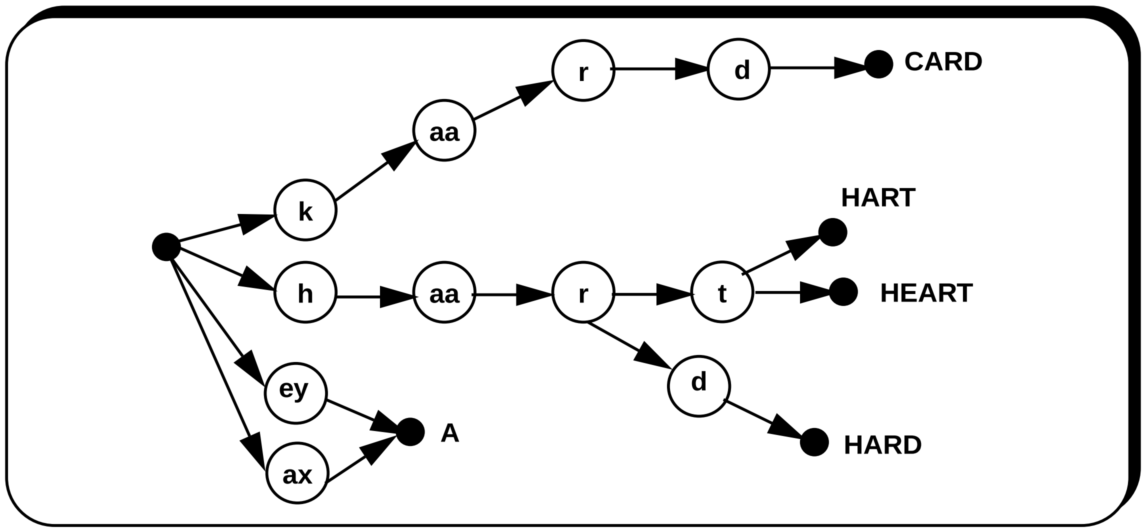


Figure 14. An example lexical tree used in the decoder. The triphones are generated dynamically, on the fly for each of the lexical tree nodes. Each lexical node contains a list of the words (or lattice nodes) on that path covered by the monophone held in the lexical node. The dark circles represent starts and ends of words, the word identity is unknown till a word-end lexical node is reached.

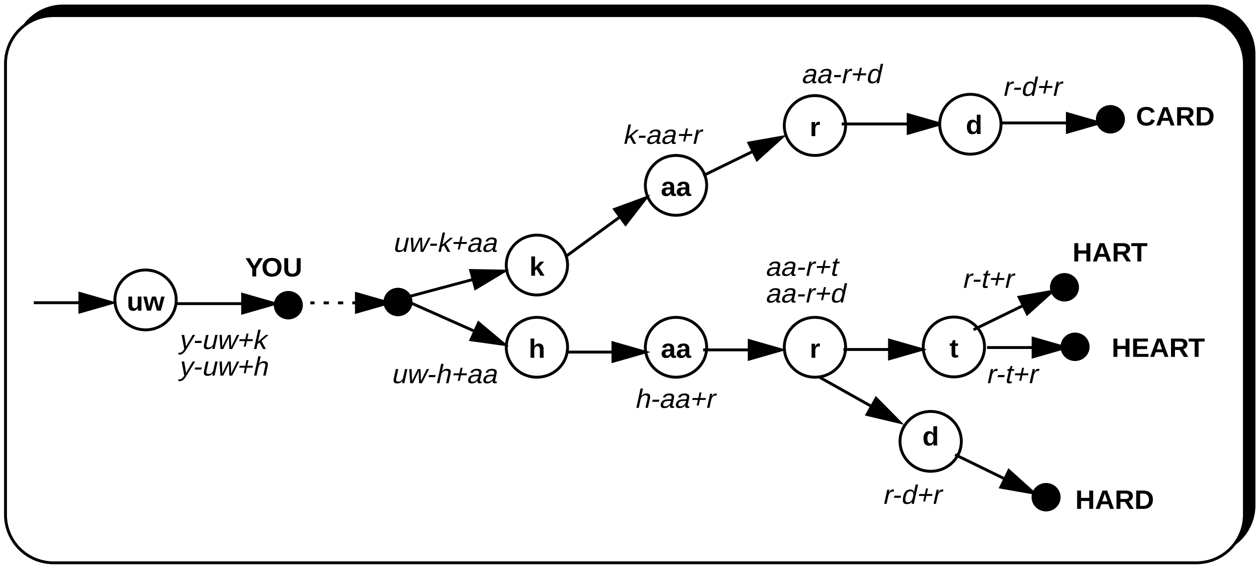


Figure 15. Generation of triphones from the lexical tree consisting of monophone lexical nodes. Note the increase in the number of triphones at word boundaries due to cross-word context.

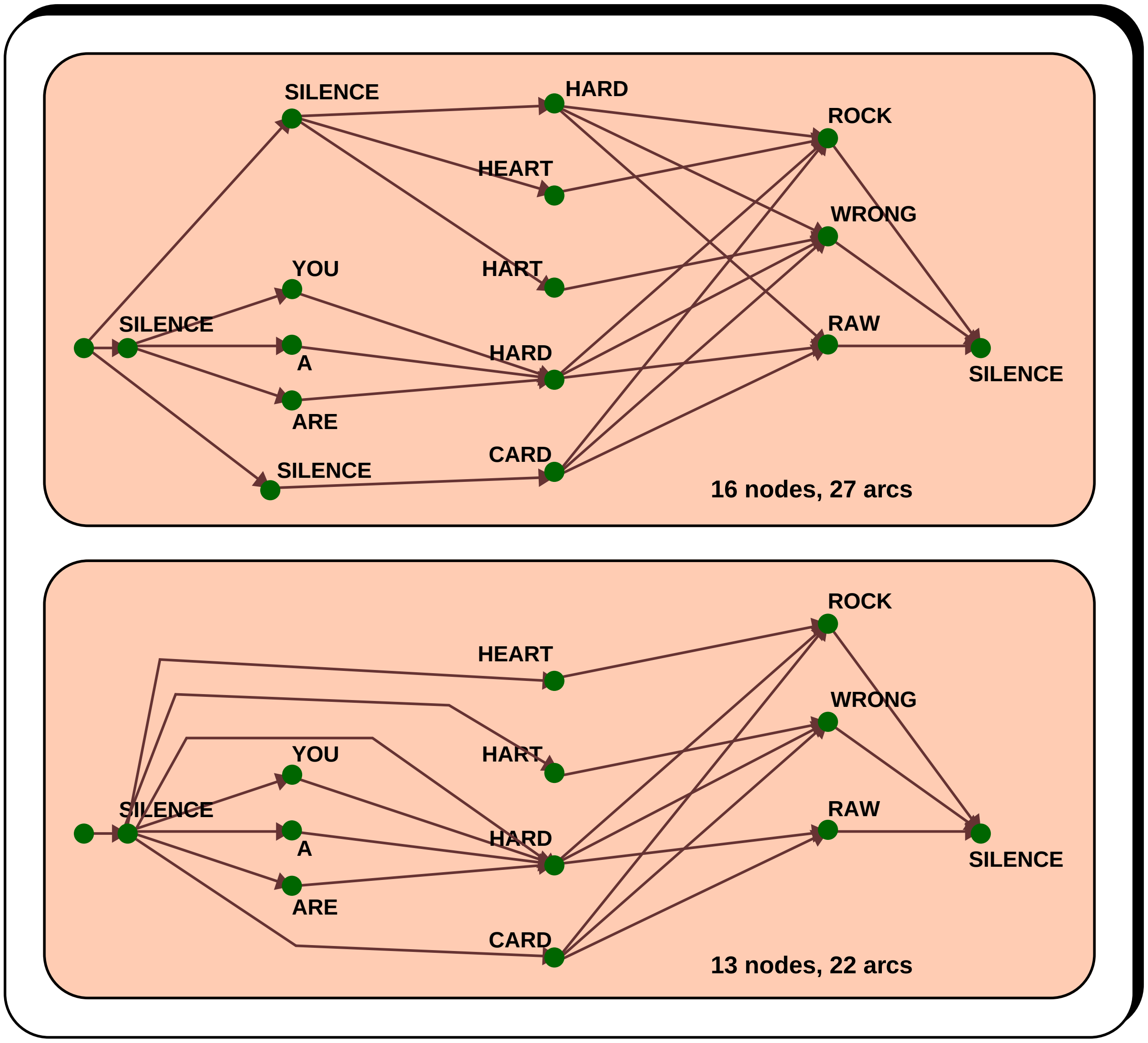


Figure 16. An illustration of the word graph compaction in the decoder. The reduced word graph yields the same unique word sequences as the original, but its size is significantly smaller. On large word graphs, the compaction results in a 2 to 5 times drop in the word graph size.

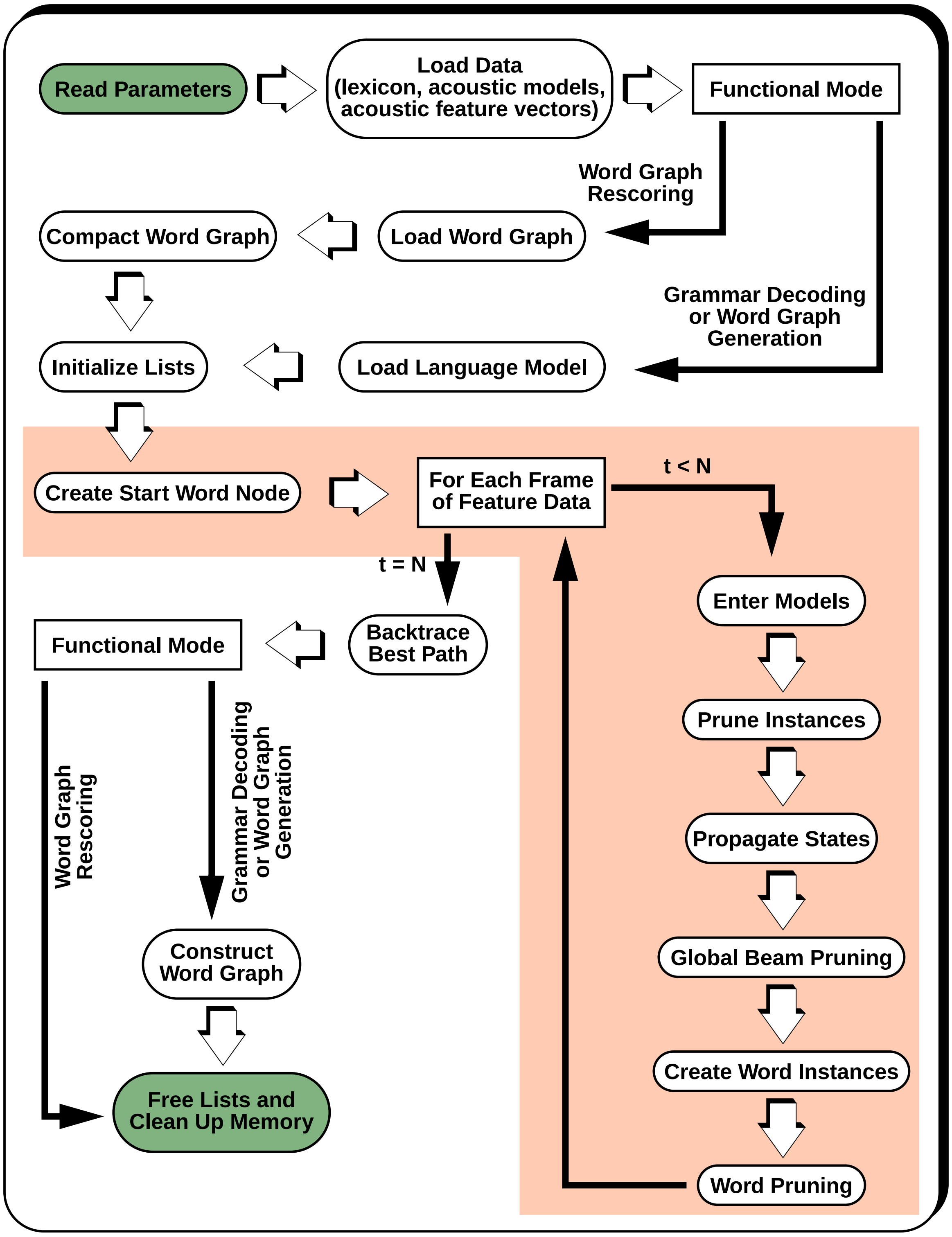
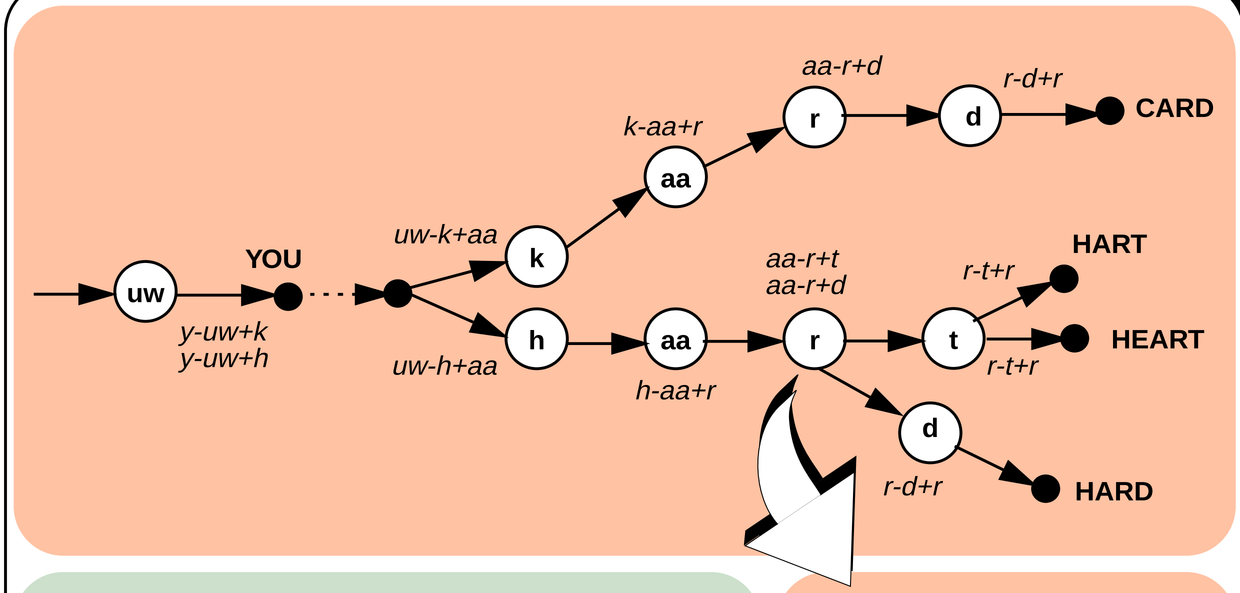


Figure 17. A schematic diagram of the control flow of the decoder in the ISIP automatic speech recognition system, for a single utterance N frames long. The shaded region represents the core search. The preprocessing (data loading etc.) and postprocessing (best path backtrace, word graph generation) are also described.



Trace	
History* history_d;	// word history of this path
Instance* instance_d;	// the instance associated with
	// this path
Lex_node* lex_next_d;	// the next node in the lexical
	// tree
int_4 level_d;	// level in the search network
	// hierarchy
int_4 phone_index_d;	// the actual phone index
int_4 state_index_d;	// the current state index
int_4 max_hist_d;	// max number of previous
	// histories
float_8 score_d;	// path probability at this trace

Instance	
Hash_cell* word_d;	// history word node
Lex_node* lex_node_d;	// position in the lexical tree
int_4 phone_index_d;	// index of the model
Link_list* token_list_d;	// list of tokens for each state
float_8 max_score_d;	// max path score for this
	// instance
float_8 lm_score_d;	// best lm score for this
	// instance
int_4 frame_d;	// frame in which this instance
	// was last active

history = YOU
lex node = [r]
model = aa-r+t
lm score = max {p(x/YOU) :
x$\in$ {HART, HEART, HARD}}

history = YOU
lex node = [r]
model = aa-r+d
lm score = max {p(x/YOU) :
x$\in$ {HART, HEART, HARD}}

The tree node with the phone “r” in the above lexical tree corresponds to two triphones and represents paths that can possibly terminate in three different words. Therefore, there are two instances associated with it for each word history, and the language model lookahead score is computed based on that word history.

Figure 18. An illustration of the definition of a path instance for two paths in the lexical tree of Figure 15. Also shown are the actual C++ class definitions for the path marker or Trace class and the Instance class.

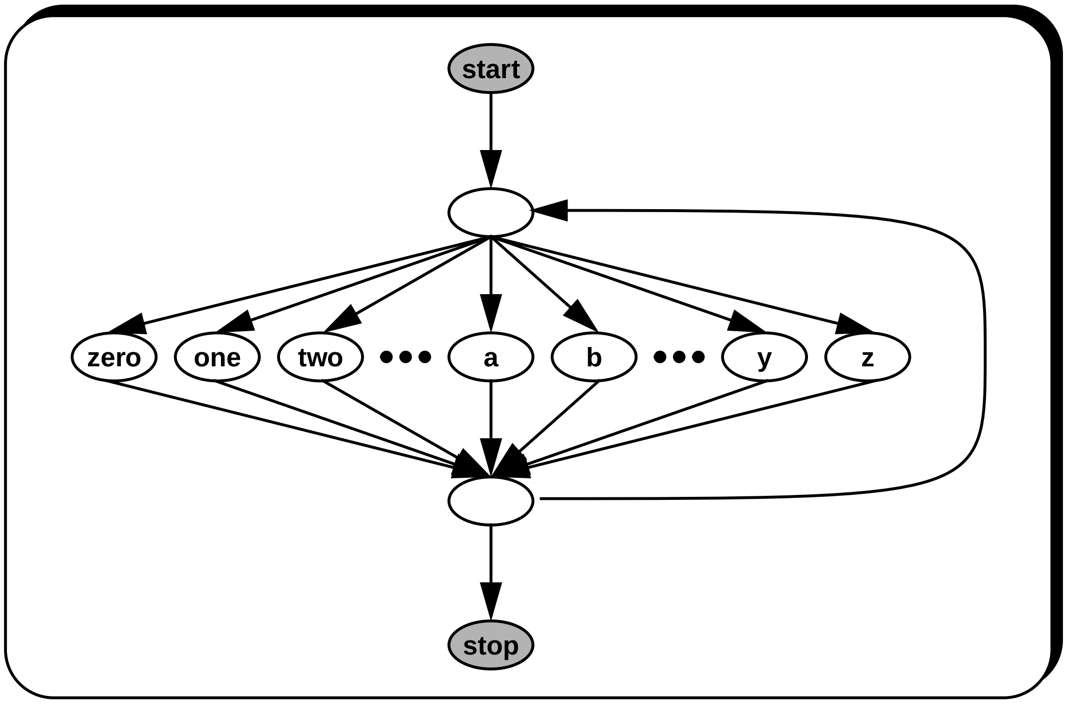


Figure 19. The language model for the Alphadigits corpus is a fully connected grammar. The empty word cells do not correspond to a word, but are used to denote the loop-back for the grammar.

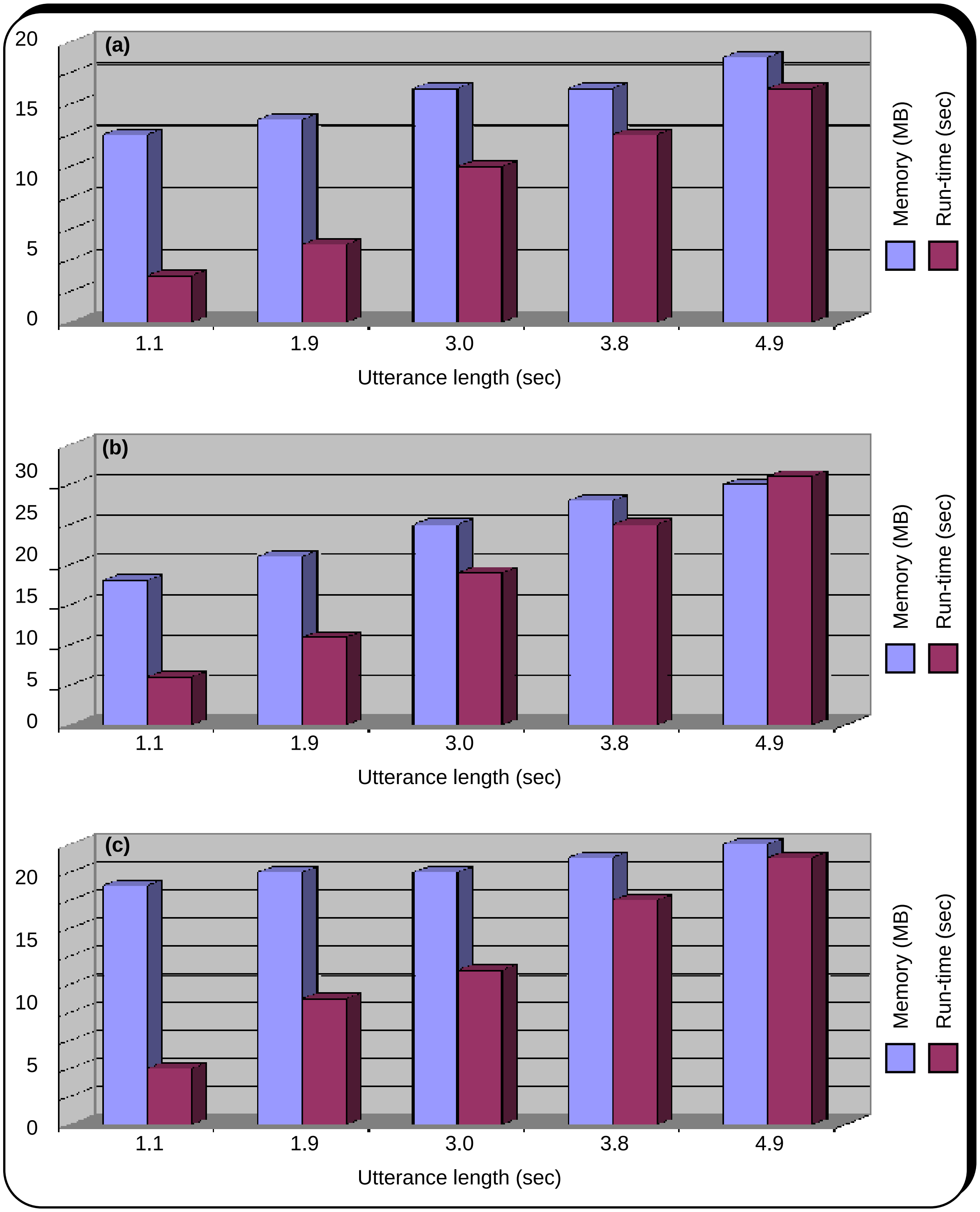


Figure 20. Memory and run-time for word graph rescoring and generation as a function of utterance length. In (a), we use word-internal models to rescore word graphs. In (b), we use cross-word models. In (c), word-internal models are used for word graph generation. Both memory and run-time tend to vary linearly with utterance length. This is highly desirable in a search algorithm.

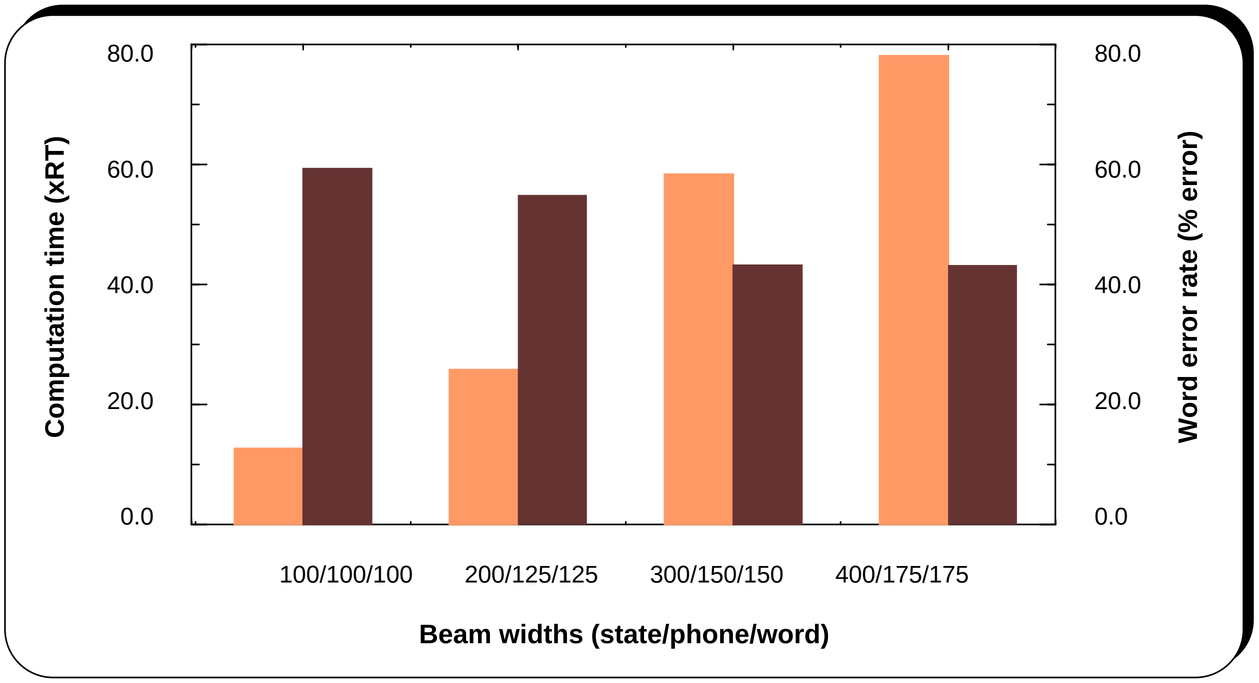


Figure 21. Effect of beam widths on the recognition accuracy and complexity of the search, on a subset of the SWB corpus for word graph rescoring with cross-word triphone acoustic models. A fixed MAPMI limit of 8000 was used. As the beams get wider, the computation time increases rapidly. However, very narrow beams cause significant degradation in recognition performance. Beam widths larger than some limiting values do not contribute to any improvement in performance.

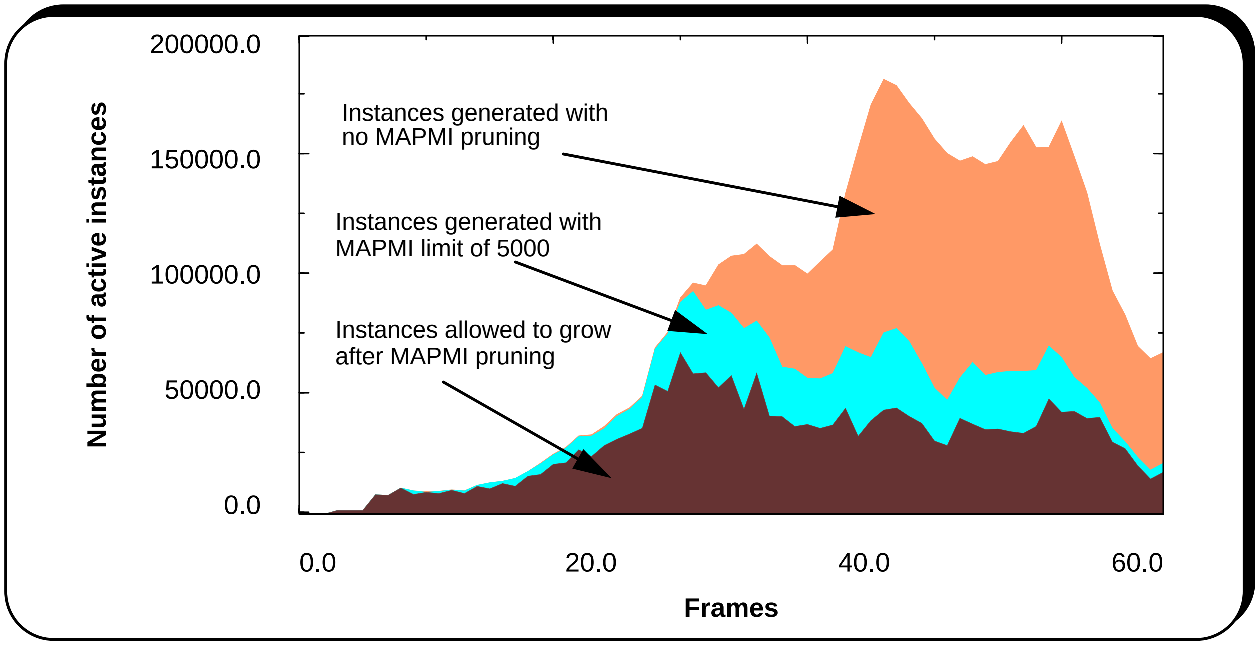


Figure 22. Effect of MAPMI pruning on memory usage as illustrated on a 68 frames long utterance from the SWB corpus for word graph generation using cross-word triphone acoustic models and a bigram language model. The number of instances active without any pruning, and therefore the memory required, is significantly larger than when a MAPMI bound is in place.

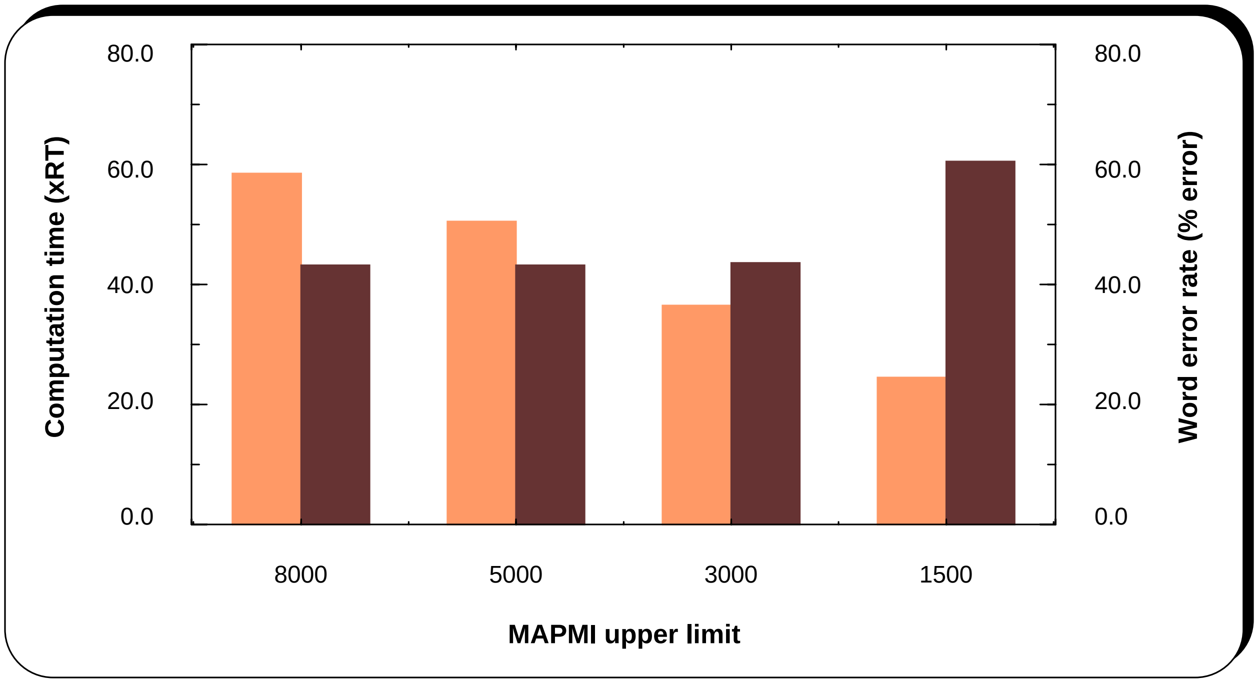


Figure 23. Effect of MAPMI pruning on the recognition accuracy and complexity of the search, on a subset of the SWB corpus for word graph rescoring with cross-word triphone acoustic models. The pruning beam widths were fixed at 300, 150 and 150 respectively for the state, phone and word levels. As the MAPMI threshold decreases, the computation time decreases rapidly. However, a very low limit on the MAPMI bound causes degradation in recognition performance.

System	Pruning		% Error Rate				Max Memory (MB)	Real-Time† (xRT)
	state model word	MAPMI	Sub	Del	Ins	WER		
Context-Independent Phones	200 100 100	300	14.3	1.7	1.5	17.5	18	3.3
Context-Dependent Word-Internal Phones	250 125 125	400	11.6	0.6	1.4	13.6	24	4.3
Context-Dependent Cross-Word Phones	300 150 150	500	7.4	0.9	1.1	9.4	41	9.6

† Comparisons were performed on a 333 MHz Pentium II processor with 512MB RAM.

Table 1. An analysis of performance on the OGI-AD task for network decoding. WER and the maximum amount of memory used both vary exponentially with real-time performance. For example, a 50% reduction in error rate requires an increase by a factor of three in xRT.

System	Pruning		% Error Rate				Max Memory (MB)	Real-Time† (xRT)
	state model word	MAPMI	Sub	Del	Ins	WER		
Context-Independent Phones	200 100 100	3000	40.6	19.7	2.4	62.8	120	42
Context-Dependent Word-Internal Phones	250 125 125	5000	32.2	14.8	2.9	49.8	160	48
Context-Dependent Cross-Word Phones	300 150 150	8000	30.9	10.1	4.6	45.6	200	60

† Comparisons were performed on a 333 MHz Pentium II processor with 512MB RAM.

Table 2. An analysis of performance on the LDC-SWB task for rescoring word graphs generated using a bigram language model. The amount of memory increases by 67% when WER decreases by 27%. Similarly, xRT increases by 43%.

System	**Pruning**		**% Error rate**				**Max Memory (MB)**	**Real-Time† (xRT)**
	state model word	**MAPMI**	**Sub**	**Del**	**Ins**	**WER**		
Context-Dependent Word-Internal Phones	250 125 125	5000	33.2	17.3	2.2	52.6	220	240
Context-Dependent Cross-Word Phones	300 150 150	8000	30.4	16.3	2.1	48.7	300	470

† Comparisons were performed on a 333 MHz Pentium II processor with 512MB RAM.

Table 3. Summary of the decoder performance on the LDC-SWB task for word graph generation using a bigram language model. Note that WER is slightly higher because word graphs are generated with tighter pruning thresholds than decoding. Also, real-time rates double when cross-word models are used.