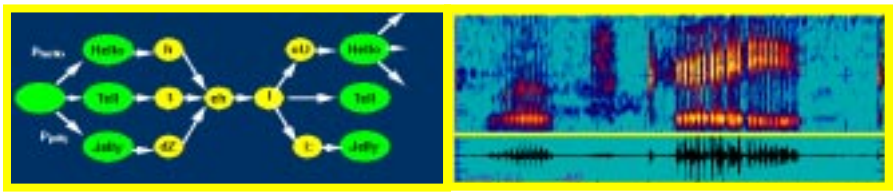
final report for

Fast Recognition Techniques for Large Vocabulary Speech Recognition

submitted to:

Dr. Yu-Hung Kao
Texas Instruments, Inc.
MS 8374, PO Box 655303
Dallas, Texas 75265
Email: yhkao@csc.ti.com

August 15, 1999



submitted by:

J. Zhao[1], J. Hamaker, N. Deshmukh, A. Ganapathiraju and J. Picone

Institute for Signal and Information Processing
Department of Electrical and Computer Engineering
Mississippi State University
Box 9571, 413 Simrall, Hardy Road
Mississippi State, Mississippi 39762
Tel: 662-325-3149, Fax: 662-325-3149
primary email contacts: {zhao, hamaker, picone}@isip.msstate.edu

1. J. Zhao was the student funded by this grant.

EXECUTIVE SUMMARY

The most important component of a state-of-the-art speech recognition system is the decoder, which finds the most likely word sequence given all the knowledge sources. However, for a large vocabulary speech recognition (LVCSR) system, the decoding process is very time-consuming and resource-intensive. Therefore, there is a need for algorithms that can achieve fast recognition while maintaining accuracy. In this work, we intended to investigate such algorithms. Our work was based on the Institute for Signal and Information Processing (ISIP) decoder, which is a hierarchical variation of the standard Viterbi-style decoder. Previously, the ISIP decoder only had the capability of word graph rescoring, which uses the word graph generated by other tools to constrain the search space. In the past year, we significantly enhanced the ISIP decoder, and implemented efficient N-gram decoding and word graph generation.

N-gram decoding is a fundamental capability of most state-of-the-art decoders. The ISIP decoder has been modified to make this process more efficient. Only one lexical tree is constructed for the lexicon. Language model weights are acquired on an as-needed basis. We also overhauled the search strategy to support generation of word graphs. Unlike N-gram decoding, where we keep only the one best hypothesis, word graph generation keeps track of multiple hypotheses at each word end. The word graph can be rescored, giving a much faster second-pass of decoding and using more complex acoustic and language models. For word graph generation, the decoder has been fundamentally modified to reflect a hierarchical implementation of paths at the state, phone and word levels; it allows coexistence of paths with multiple histories, *i.e.*, N-best lists. Now the system can generate word graphs using both word-internal and cross-word triphone models. In addition, we added a module for word graph WER computation.

The ISIP decoder is equipped with advanced pruning techniques, including beam pruning and maximum active phone model instance (MAMPI) pruning. We also tried a two-pass fast match pruning strategy, where the first pass quickly finds an approximate best hypothesis, and its partial path score is then used as the pruning threshold for the second pass decoding. These pruning techniques significantly reduced the memory usage and improved decoding speed.

The N-gram decoding capability of the system has been evaluated using bigrams and cross-word triphones. To expedite the evaluation, we tested the system on a subset of the standard SWB evaluation data and used tight pruning thresholds. A word error rate (WER) of 53.2% was obtained, which is comparable to WERs measured by similar research systems at other sites using comparable data and models. We generated word graphs for the WS'97 Switchboard (SWB) test set using word-internal triphone models. These word graphs were rescored using cross-word triphone models and achieved a WER of 47.3%, which was further reduced to 45.6% with trigram language models (compared to 46.2% achieved during WS'97). Our system improvements allow us to generate rich word graphs (the measured word graph WER was 15.6%). Word-internal word-graph generation runs at ~200 xRT and cross-word word graph generation runs at ~400 xRT. Although these speeds are not close to real-time, with only minor degradations in performance (typically, no more than a 25% increase in WER), the system can be transformed into one that operates at 10xRT or less. We expect significant gains by improving the way we handle pruning of instances. Also by using fast Gaussian computation techniques, the system is expected to speed up dramatically without much performance loss.

TABLE OF CONTENTS

ABSTRACT

A decoder or search engine is the most important and resource-intensive component in a large vocabulary speech recognition (LVCSR) system [1]. Given this importance, it is no surprise that many algorithms have been devised which attempt to increase the efficiency of the search process while maintaining the quality of the recognition [2-4].

In this work, based on the previous Institute for Signal and Information Processing (ISIP) decoder [5], we have implemented efficient N-gram decoding and word graph generation. The N-gram decoding capability of the system has been evaluated using bigrams and cross-word triphones. To expedite the evaluation, we chose a subset of the standard SWB evaluation data as the test data, used tight pruning thresholds, and obtained a word error rate (WER) of 53.2%. As a comparison, for this dataset, rescoring of word graphs gives a WER of 50.6%. The system has been used to generate word graphs for the entire WS'97 development test set. These word graphs have an inherent word error rate of 15.6%. The word graphs were then rescored using cross-word triphone models and trigram language models. Using cross-word triphones, we achieve a WER of 47.3%, which is further reduced to 45.6% with trigram language models. Word graph generation runs at about 200 xRT using word-internal triphones and a bigram language model. The system using cross-word models is not very efficient yet.

1. INTRODUCTION

In recent years, speaker-independent speech recognition technology has made significant progress from the days of isolated word recognition. Large vocabulary continuous speech recognition systems are now finding their way into the marketplace, mainly in the desktop and server markets. Typically these systems are restricted to a particular domain such as automatic dictation or command-and-control applications. With these restrictions, developers are able to create highly efficient systems which run in real-time with very low error rates. However, the primary goal of speech research is to produce systems that allow users to interact naturally without restrictions on either content or style of speech. Unfortunately, the resources required for a state-of-the-art conversational speech recognition system to be commercially viable are beyond the capabilities of available hardware technology. For some applications, decoding a simple one-word sentence may take tens of megabytes of memory and may run at hundreds of times real-time. The majority of this resource consumption is owed to the search process, which selects a word sequence with the highest probability given the observed acoustic data [6]. Here, we will give a brief introduction on the typically-used search algorithm —the Viterbi search algorithm. This is also what we use as the core of the ISIP decoder.

1.1. Viterbi Search

Viterbi search and its variant forms belong to a class of breadth-first dynamic programming techniques [7]. 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 builds a breadth-first search tree out of this network by the following steps [8]:

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.

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. Therefore, other constraints such as language models and pruning techniques have to be applied to reduce the search space.

1.2. Search Space Organization

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
- acoustic models: Hidden Markov 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)

The search space grows exponentially as a function of the lexicon size, the number of HMMs and the language model constraints; and this growth imposes formidable requirements on the computation and storage capability of the system for the implementation of the search algorithm. 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. Therefore, careful design of these components and reduction of the search space are crucial to the development of a successful system.

Because the number of possible unique phonetic contexts is much smaller than the number of possible next words, sharing the phonetic context across all the words allows the number of paths to be grown at each word end to be reduced drastically. Therefore, in large vocabulary speech recognition, the search space is often organized as a phonetic lexical tree [9]. A lexical tree is a pronunciation prefix tree which 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 and can be shared by multiple words with the same partial pronunciations. 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. The possible next words are defined by the language model or word graph, which gives the constraints on the search space at the word level.

1.3. N-gram Decoding and Word Graph Generation

Typically, one-pass decoding using an N-gram language model is called N-gram decoding, which is a fundamental capability of most state-of-the-art speech recognition systems [10, 11]. 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 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 [12]. For instance, in a bigram language model the probability of a word sequence (w_i, w_j) is given by

$$p(w_i, w_j) = \begin{cases} p(w_j | w_i) & \ldots (w_i, w_j) \text{ exists in LM} \\ b(w_j)\, p(w_i) & \ldots \text{otherwise} \end{cases} \tag{1}$$

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.

However, even though N-gram language models store only a small subset of all the possible sequences of N words, they are still significantly large for large vocabulary applications. Therefore the efficiency problem of N-gram decoding is an important issue toward achieving fast recognition. Many systems have been devised which try to improve the efficiency of N-gram decoding. An alternative to the one-pass N-gram decoding is word graph generation. Word graph generation is similar to N-gram decoding, but instead of keeping only the 1-best hypothesis, it keeps multiple hypotheses at each word end, and outputs them as a word graph. This word graph incorporates a large number of hypothesis and defines which words can follow which words. It then can be used as the grammar constraints in a second pass decoding, which is typically referred to as the word graph rescoring. Because the search space is strictly constrained at the word level

by the word graph, word graph rescoring is much more efficient than the N-gram decoding. Although the process of word graph generation is time-consuming, once a word graph is generated, it can be rescored using more complex acoustic models and language models. This will give an overall better performances in terms of both speed and accuracy.

Our intention for this work was to improve decoding speed while maintaining recognition accuracy [13]. We enhanced the previous ISIP decoder's functionality and implemented efficient N-gram decoding and word graph generation [14]. In our implementation, we applied carefully designed data structures, memory management and advanced pruning techniques. With the capability of word graph rescoring which the previous ISIP decoder already had, we now have constructed a full-fledged and efficient LVCSR decoder.

2. EFFICIENT ISIP DECODER

To facilitate the implementation of N-gram decoding and word graph generation, in the past year, the ISIP decoder has undergone numerous design changes, as well as enhancements to its functionality.

2.1. Previous ISIP Decoder

Construction of the ISIP's decoder was started one and a half years ago. The previous version of the ISIP decoder was based on a standard dynamic programming paradigm, *i.e.*, Viterbi time-synchronous breadth-first search. The decoder has the capability of word graph rescoring. In word graph rescoring, a word graph is used as a grammar, providing constrains on the occurrence of particular words and word sequences. This word graph is generated by an initial N-best search, which is known as word graph generation.

First the decoder reads all necessary resources into the corresponding data structures. These sources include monophones, lexicon, HMMs, transitions, states, triphones, word graphs and the input speech feature data.

Special scoring data structures (markers) are used to propagate path information along the frames. The decoder uses two sets of path markers — one at the phone (or model) level, and the other at a model-internal or state level. Each path marker contains an index to the current word graph node, the current lexical tree node, and the present active triphone, thus providing a multilevel or hierarchical representation for each instance in the search space. In addition, the path marker contains the current frame index, a backpointer to its previous marker, and the overall path score. Triphones are instantiated dynamically based on the lexical tree context. At the beginning of each triphone, a state-level marker is projected from the previous (phone-level) marker and added to a list of markers corresponding to that state of the triphone. Each state is evaluated only once per frame, and only if it is active in that frame. The state score is stored in the state data structure and added to the state-level marker path score at the time of projecting the marker to the next state. At each successive frame, these markers are compared with each other as per the Viterbi paradigm and the best marker for each different instance of the state is projected to the next states as governed by the state transition probabilities. A marker exiting the HMM is added to the phone-level marker list and used to project the next triphone markers.

Different paths along the search network can be merged provided they meet at a point in the search space wherefore their future is identical. This allows the decoder to share the computation of similar parts of different hypotheses, and prevents the amount of computation from increasing exponentially with the expansion of the search space as newer words are added to the hypotheses. Paths are therefore merged at word ends, by keeping only the highest-scoring marker for each triphone instance at the end of a particular word (or word graph node). More specifically, at any frame, if more than one active marker leads to the end of a word (as represented in the word graph), and they have the same acoustic context (same triphone), then only the best path marker among them is propagated further, and the rest are returned to the program memory for reuse. The word graph and lexical tree structure of the decoder framework automatically ensure that all the partial hypotheses represented by these markers have identical linguistic context.

Once the feature data (mel-frequency cepstral coefficients or mfcc) for all the frames have been processed, the decoder finds the highest scoring path marker that terminates in the sentence-end symbol and then follows the back-pointers of the successive markers to the sentence-start symbol to generate the best sentence hypothesis for that data.

This version of the decoder had the capability of word graph rescoring, but did not have the capability of N-gram decoding and word graph generation, which are essential features for decoders used in LVCSR tasks. Another drawback of the previous design was the fact that the same data structure was used to store history and path information. This creates problems in the computation of acoustic scores of paths in the word graph generation mode. In order to correct this flaw, we needed to design a new data structure that holds history information and is independent of the path information that is propagated at every frame. The word-graph rescoring module's implementation involved the creation of several copies of the lexical tree during decoding. However, this is prohibitively expensive in the N-gram decoding mode in terms of system memory, especially when dealing with trees representing a vocabulary in the order of a few tens of thousands words. The decoding algorithm needed to be modified to tackle this inefficiency.

2.2. N-gram Decoding

In the N-gram decoding mode of the ISIP decoder, the N-gram models are read from the input N-gram file into two special data structures —Ngram and Ngram_Node, which are specially designed to efficiently compute the language model (LM) scores. We use a hash-table mechanism to store these N-gram nodes for quick access during decoding. During N-gram decoding, the search space is organized as a lexical tree, and several techniques have been used to reduce the search space.

N-gram Lexical Tree

The decoder constructs a lexical tree encompassing all the words in the lexicon. An example of the lexical tree is shown in Figure 1. Each lexical node contains a list of the words on that path covered by the monophone held in the lexical node. The dark circles represent starts and ends of words, but the word identity is unknown until a word-end lexical node is reached. This lexical tree is used to generate phone hypotheses. The scores for word transitions are computed based on the

position in this lexical tree and the current N-gram history. A virtual copy of the tree is created for each N-gram word history. This is a major enhancement of the ISIP decoder as it avoids the memory explosion of constructing many N-gram lexical trees.

Typically, in lexical tree-based searches, the LM scores are stored in the lexical tree nodes [15]. When a word end is reached during the decoding process, the decoder constructs a new lexical tree encompassing all the possible next words which are used to generate the next phone hypotheses. For word graph rescoring, constructing new lexical trees is necessary since the possible next words are different for each word, and the number of possible next words is relatively small. However in N-gram decoding, each word (except the sentence start word) can be followed by any other word, so for large vocabulary speech recognition, the lexical tree would be very large. Even a few copies of the complete lexical tree will quickly overshoot the available memory. Since we know that the N-gram lexical tree structure is the same for all occurrences of each word, and that only the LM scores vary according to their N-gram histories, making many trees with the same structure is a waste of both time and memory. Therefore, instead of making many copies of the lexical tree, we reuse the same N-gram lexical tree for each occurrence of a word. This is done by dissociating the LM scores from the lexical tree and making the single tree independent of the predecessor words. The triphones are generated dynamically for each of the lexical tree nodes, and language model weights are acquired 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.

Language Model Lookahead

Another feature which makes our N-gram decoding efficient is language model lookahead. This feature plays a crucial role in limiting the size of the search space in the lexical tree approach.

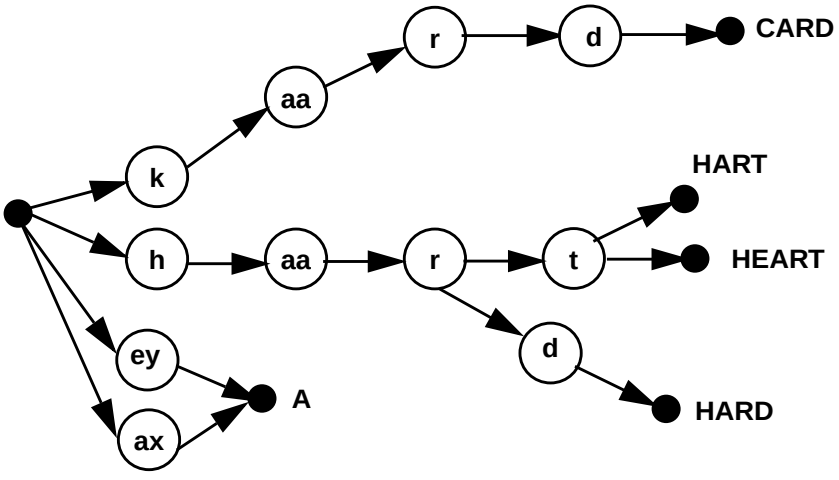


Figure 1. An example of a lexical tree used in the decoder.

Language model lookahead is designed to apply the language model constraints on the search space as early as possible [16]. 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 afterwards. 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 inefficiency 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 evaluation involves computation of the Mahalanobis distance of the feature vector from a weighted mixture of multivariate Gaussian distributions, as described in Equation 2. The decoder deals with likelihood scores (log probabilities) 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

$$l(\underline{x}/s) = \sum_{i=1}^{m} [(\underline{x} - \underline{\mu}_i)^T \Sigma_i^{-1} (\underline{x} - \underline{\mu}_i) + K_i] \quad (2)$$

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

This calculation is typically the most expensive computation in searching, since it needs to be conducted for every active state for each frame. The HMMs we used are modeled after state-tying [17], which allows similar states in different models to share the same distributions and hence, reduce the total count of distinct states. Now, since the same state can be accessed by different models, 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.

2.3. Word Graph Generation

One-pass N-gram decoding is very resource-intensive for large vocabulary speech recognition tasks. An alternative to one-pass N-gram decoding is to generate word graphs with a simpler

language model (*e.g.*, a bigram) to conserve computing resources, and then to run another pass of decoding to rescore these word graphs with more advanced acoustic and language models [18, 19]. In this way, we can get overall better performance. The algorithm for word graph generation is almost same as that for N-gram decoding. At the word level, the linguistic search space is still constrained by the N-gram language model. Theoretically, the only difference is that, instead of generating the 1-best hypothesis, word graph generation creates an ordered list of multiple hypotheses at each word end, keeping track of the different path histories of that word, and in the end, outputs all the hypothesis as a word graph. Although this difference appears to be very easy to understand, it is not that easy to implement. Implementing word graph generation caused a fundamental reorganization of the search space in our decoder.

Search Space Reorganization

The decoder breaks the search space into a hierarchy of different levels based on the selection of the acoustic unit. For instance, we can view an utterance in conversational speech at a **sentence level** including the whole utterance, a **phrase level** consisting of the phrases making up the sentence, a **word level** including each word in the sentence, a **phone level** consisting of the phones comprising each word (also called a **model level** when the acoustic modeling unit is the phone), or a **state level** that is internal to the models.

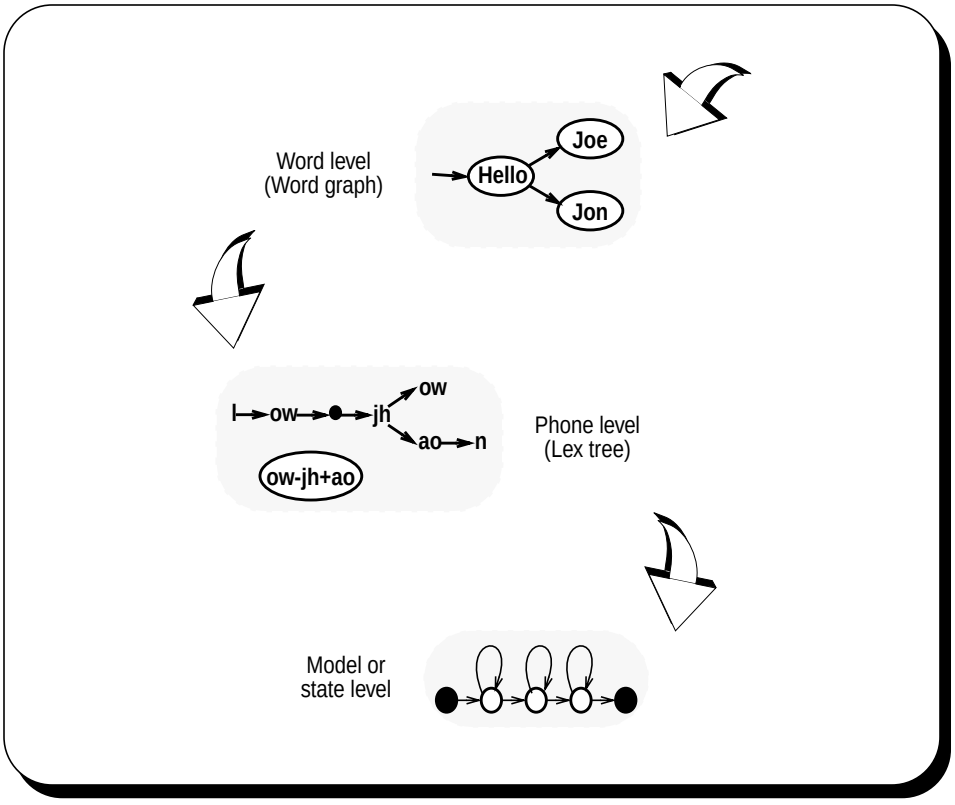


Figure 2. Hierarchical representation of the search space.

For word graph generation, the decoder software has been modified fundamentally to reflect a hierarchical implementation of paths at the state, phone and word levels (extensible to higher levels such as phrases and sentences), and to allow coexistence of paths with multiple histories, *i.e.*, N-best lists. Figure 2 shows the organization of the search space in the updated ISIP decoder. The current decoder explicitly maintains the three levels by creating word-level markers that point back to the word-end phone-level markers. Maintaining them is not only more correct intuitively, but it is also found to be more efficient. Since word graph generation inherently involves tracking multiple histories at the end of each word, a careful design of the structure that carries this information is needed. In the previous version of the decoder, this information was part of the path marker. However, computing the acoustic scores for each arc of the word graph at the end of the decoding process is inaccurate in this case. To remedy this problem and to keep the history and path information separate, we decided to add a new data-structure to hold the history information. Though this change marginally increased memory usage, it was deemed not significant enough to warrant making the decoder less flexible.

Path Merging

A benefit of decoding using the N-Gram method is that different paths at the same instant of time can be differentiated only based on the phone model and the word history of each path. Thus, paths with very different origins can be merged later in time, if they have the same current instance. The phone model and the N-gram history word sequence now delineate this situation.

During word graph generation, when merging word-level paths, the word path 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 can be merged when a path reaches the exit state of an acoustic model.

At the end of the decoding process, all the paths terminating at a proper end-of-sentence position in the search space are output in the form of a word graph that consists of a word at each node. The arcs indicate word transitions that constitute different paths. Each arc is associated with an acoustic score for the end word and a linguistic likelihood of the end word given the start word for that arc.

Word Graph WER Computation

In addition, we have added a module for word graph word error rate (WER) computation. This new capability is based on a dynamic programming algorithm in which various types of word errors (substitutions, deletions, and insertions) can be adjusted to achieve prescribed operating points.

- 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.

We have described the two main functionalities of our ISIP decoder —N-gram decoding and word graph generation. Next, we will describe the general work flow of the ISIP decoder.

2.4. ISIP Decoder Work Flow

The search algorithm used in the ISIP decoder is based on a hierarchical variation of the standard Viterbi-style time-synchronous search paradigm [14]. As discussed earlier, the search space is separated into a hierarchy of state, phone, and word levels, and is organized as a lexical tree.

Data Loading

At first, the decoder loads all the necessary input sources, the lexicon, HMMs, N-grams, etc., into the corresponding data structures. The necessary input parameters are shown in Appendix A.

- monophones — a list of the basic monophones that constitute the acoustic context
- lexicon — a list of all the words and pronunciations used for the current application
- transitions — an indexed list of various state transition matrices for the HMMs
- states — a set of state parameters (number of mixtures, an index pointer to the correct transition matrix and for each mixture the mixture weight, Gaussian scale factor, mean vector and covariance matrix) for each HMM state
- models — a list of HMMs with index pointers to the states
- triphones — a list of all the possible triphones mapped to the corresponding constituent monophones, and an index to the correct HMM
- N-grams/word graph — a file containing the N-gram language models or a word graph

Hierarchical Search

At each frame of the utterance, the system maintains the complete history for each active path at each level in the search hierarchy via special scoring data structures (markers). Each path marker keeps its bearings in the search space hierarchy by indexing the current history (or word graph) node, lexical tree node and the triphone model. It also maintains the path score and a backpointer to its predecessor. Figure 3 shows a schematic representation of the control flow involved in the search process. 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 transitions from one phone to the next in the

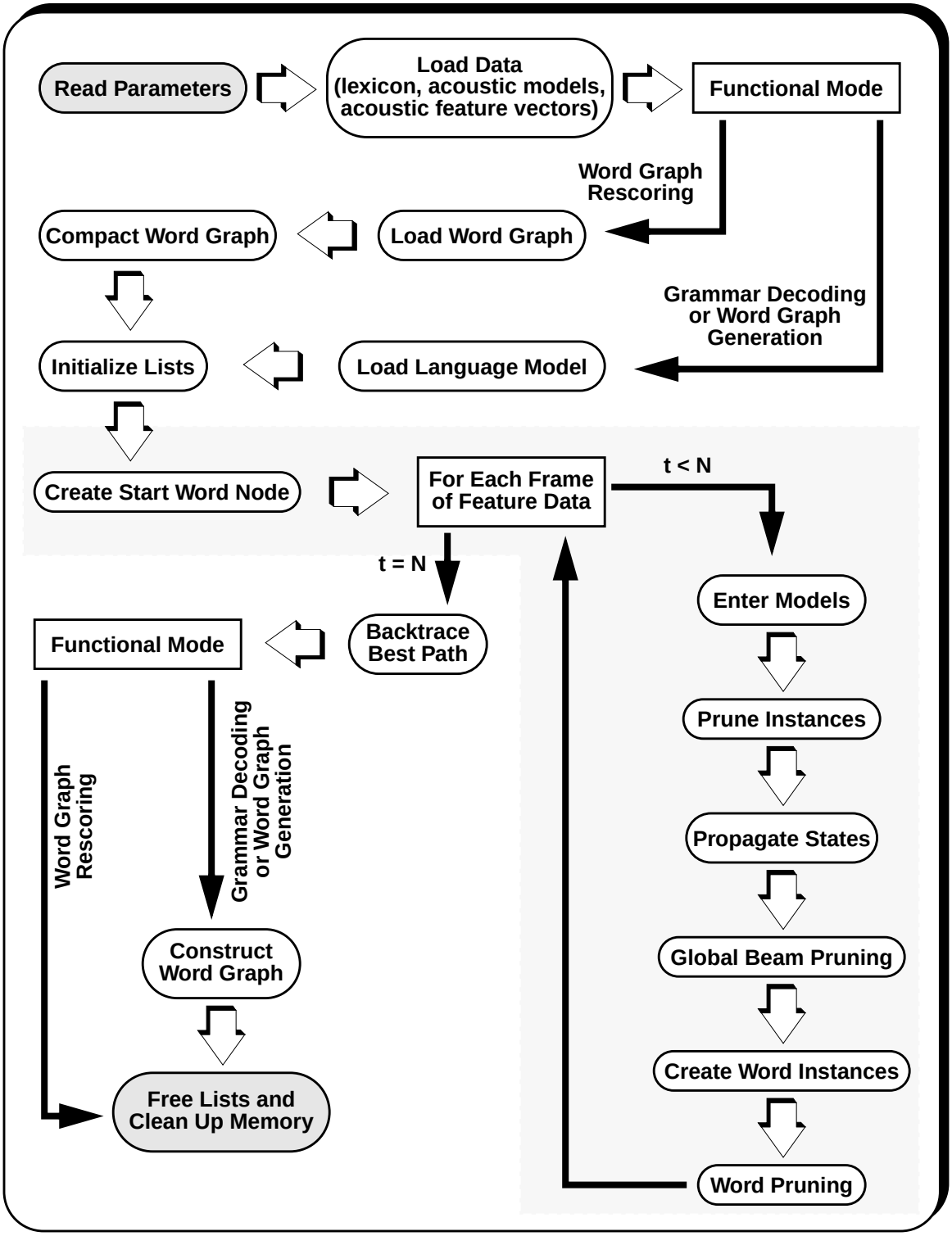


Figure 3. A schematic diagram of the Control Flow of the ISIP automatic speech recognition decoder for a single utterance N frames long. The shaded region represents the core search.

pronunciation of the word.

In the word graph generation / N-gram decoding mode and in the rescoring mode, the decoder uses a scoring information data structure, or a marker, that maintains all the information of the current path. 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 back-pointers to the previous nodes on the current path.

At the HMM state level in the acoustic models, the markers corresponding to the same instance are always 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 them, 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.

At each instantiation of an acoustic model such as a triphone, a state-level path marker is projected from the previous phone-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 ends of words are promoted to the word level and used to project paths into the subsequent words.

After all the feature data have been processed, the decoder finds the one or more path markers that terminate in the sentence-end symbol and then follows the back-pointers of the successive markers to the sentence-start symbol to generate the 1-best hypothesis or generate a word graph.

Memory Management

It is essential for an efficient decoder to manage its memory resources so that unused memory is periodically freed and reused. The higher the memory requirements of a decoder, the slower and more inefficient it is. Based on this principle, the ISIP decoder uses a central memory manager to efficiently create and reuse memory for those data structures used extensively in the decoder, such as word graph nodes, lexical tree nodes, hash table cells, *etc*. The decoder creates bulk memory at the start for a large number of such objects which are used to hold the active hypotheses until they are exhausted. At this point, another large number of objects is newly allocated. The data structure objects that are no longer in use are returned to the memory manager, which re-issues them whenever a new object is required by the decoder next.

In the decoding process, memory usage is further reduced by applying several advanced pruning techniques, which are going to be discussed in the following section.

3. ADVANCED PRUNING TECHNIQUES

In order to conserve computing and memory resources, it is imperative to prune away low-scoring partial paths that have a very low probability of getting any better, and to stop propagating them further. The process of removing such paths from the search space is known as pruning. In our ISIP decoder, currently, we use beam pruning, and maximum active phone model instance (MAPMI) pruning.

3.1. Beam Pruning

Beam pruning advances only paths whose scores fall within a specified range. Consider the path which is in state s at frame t and whose score is given by $q(s, t)$. At each frame, the state with the highest path score, $q_{max}(s, t)$, is found. In beam pruning, the pruning threshold is set to be

$$q_{Th} = q_{max}(s, t) + b(t) \tag{3}$$

where $b(t)$ is the beam width chosen to be appropriate for the application in question. All states whose path scores falls inside this threshold are considered active and extended further; any others are pruned away.

Our decoder allows the user to set a separate beam at each level in the search hierarchy — words, phones, and states. 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. 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. State-level pruning is also referred to as 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 a new word hypothesis.

Since the identity of a word is known with a much higher likelihood at the end of the word than at the beginning, 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's list of possible next words. Therefore, the word-level threshold is usually tighter than the state and phone-level beams.

3.2. MAPMI Pruning

The total memory requirements and 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 instances which are active at any time, by using an approach referred to as maximum active phone model instance (MAPMI) pruning.

The number of such instances active at any frame displays sizeable surges at word boundaries since a large number of new models are activated at word boundaries in order to accommodate the generation of new words. This can pose very severe requirements for 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. 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). All paths having an instance whose best score is better than the threshold score are allowed to propagate.

3.3. Fast Match Pruning

In addition to the above two pruning techniques, we explored a two-pass fast-match decoding strategy [20]. The first pass is the fast-match search, and the second pass is a detailed search. In the first pass, the fast-match search extends only the M word-level paths with highest scores at that time frame in the search. All other path objects are regarded as inactive and do not propagate. The basic idea behind the fast-match search is shown in Figure 4. In this figure, the dotted line indicates that only the words ending in the same frame can be compared and that only a fixed number of words are active at any one time. Therefore, it can quickly find an approximate best

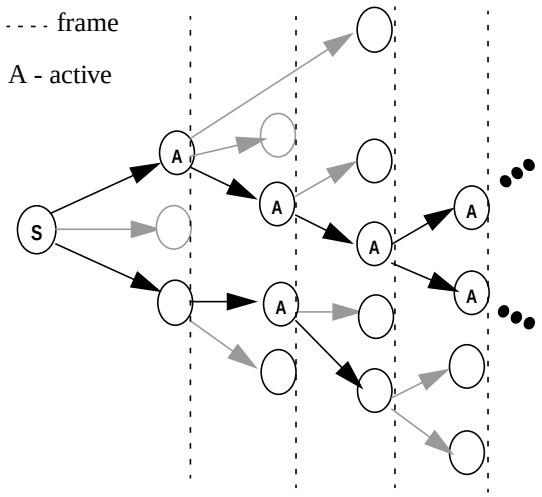


Figure 4. Fast-match search, *M*=2

hypothesis. The path score of this fast-match hypothesis then is used as the pruning threshold in the second pass decoding. We refer to this technique as fast match pruning. The aim of fast match pruning is to keep only those paths that have a reasonable chance of being the overall best hypothesis. A limitation of fast match pruning is seen when the overall best hypothesis may have a score which is worse than the fast-match threshold for some frame in the search, but which would get better if allowed to continue. Since the fast-match search prunes based on information in only the current frame, this best path will be pruned away, resulting in a search error.

There are also some other pruning techniques that we are planning to try, such as dynamic frame skipping (DFS) techniques [21], which re-evaluate the acoustic models only if the acoustic vector has significant changes during a particular frame.

4. EVALUATIONS

Our system has been evaluated on the OGI alphadigit corpus [22] or Switchboard Data set [23]. The OGI Alphadigits Corpus (OGI-AD) is a database of telephone speech collected from approximately 3,000 subjects. The vocabulary consists of the letters of the alphabet as well as the digits 0 through 9, a total of approximately 40 words. The Switchboard (SWB) consists 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. The word error rates on SWB are typically in the mid-30% range with sophisticated systems, and in the mid-40% range for simple baseline systems. Decoding with cross-word acoustic models is challenging in the SWB recognition task, because a large amount of pruning is required to limit the search space from expanding without limit, due to the large vocabulary size.

4.1. N-gram Decoding Using Bigram Language Models

N-gram decoding is highly resource intensive compared to word graph rescoring. It typically requires about 5 times as much time as word graph rescoring with a similar scale factor for memory usage. With the intention of expediting evaluations, we choose to test this feature on a subset of the standard SWB evaluation data, and used tight pruning thresholds. A word error rate

	Bigram Decoding	Word Graph Rescoring
WER	53.2%	50.6%
Subs.	36.1%	35.4%
Dels.	13.6%	9.8%
Ins.	3.4%	5.3%

Table 1.Bigram Decoding vs. Rescoring of Bigram Word Graphs.

(WER) of 53.2% was obtained. The bigram language model used has been used to generate word graphs over the past few workshops at Johns Hopkins University. It contains 22,000 unigrams and 300,000 word pairs with backoff [24].

For this dataset, rescoring of word graphs gives a WER of 50.6%. This comparison of bigram decoding and word graph rescoring is summarized in Table 1. Note that the word graphs are bound to give better performance because they were generated using triphone models enhanced by speaker adaptation and vocal tract length normalization.

4.2. Word-Internal Word Graph Generation

As a first test to quantify the efficiency of the decoder in generating word graphs, we generated word graphs for the WS'97 development test set using word-internal models and bigram language models. A word-internal context was chosen to expedite the evaluation process. The word-graph generation process runs at about ~200 xRT which is comparable to most other recognizers. These word graphs were rescored using cross-word triphone models and achieved a WER of 47.3% (compared to 46.2% achieved during WS'97). It was further reduced to 45.6% with trigram language models. These word graphs have a WER of 15.6% which is a very encouraging result. The richness of the word graphs compares favorably to the word graphs that were generated during WS'97 using cross-word models with advanced features such as vocal tract length normalization and speaker adaptation.

The memory requirements typically vary with the inherent confusion in the acoustics and the length of the utterance. Figures 5 and 6 show the resource requirements for typical utterances in SWB. Cross-word word graph generation is not efficient yet; it runs at ~400 xRT.

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

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 1. Summary of the decoder performance on the SWB task for word graph generation using a bigram language model. Note that the 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.

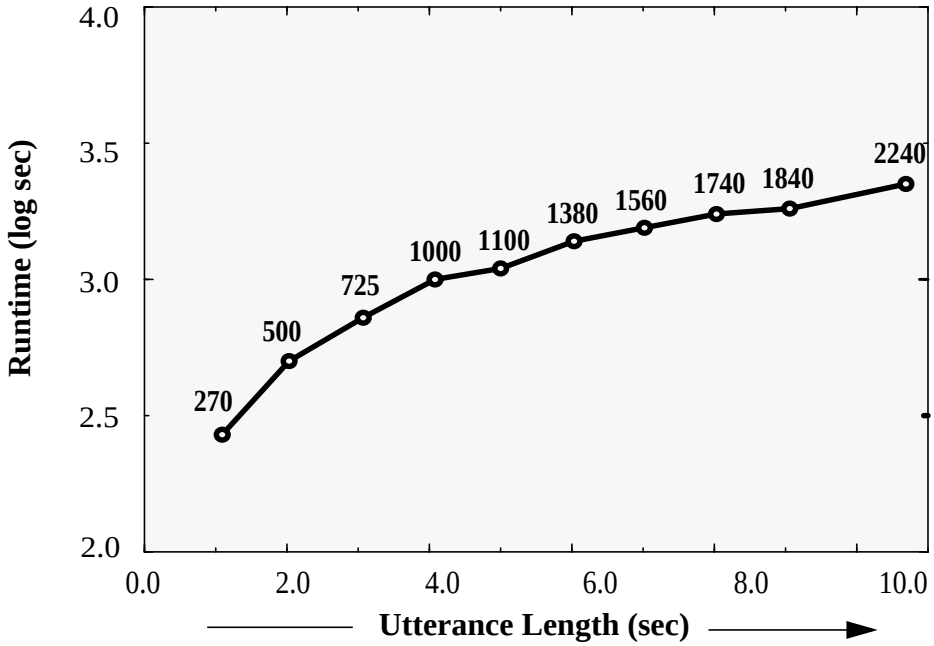


Figure 5. Resource usage for word-graph generation in using word-internal triphones and a bigram (Numbers on the plot indicate actual processing time.)

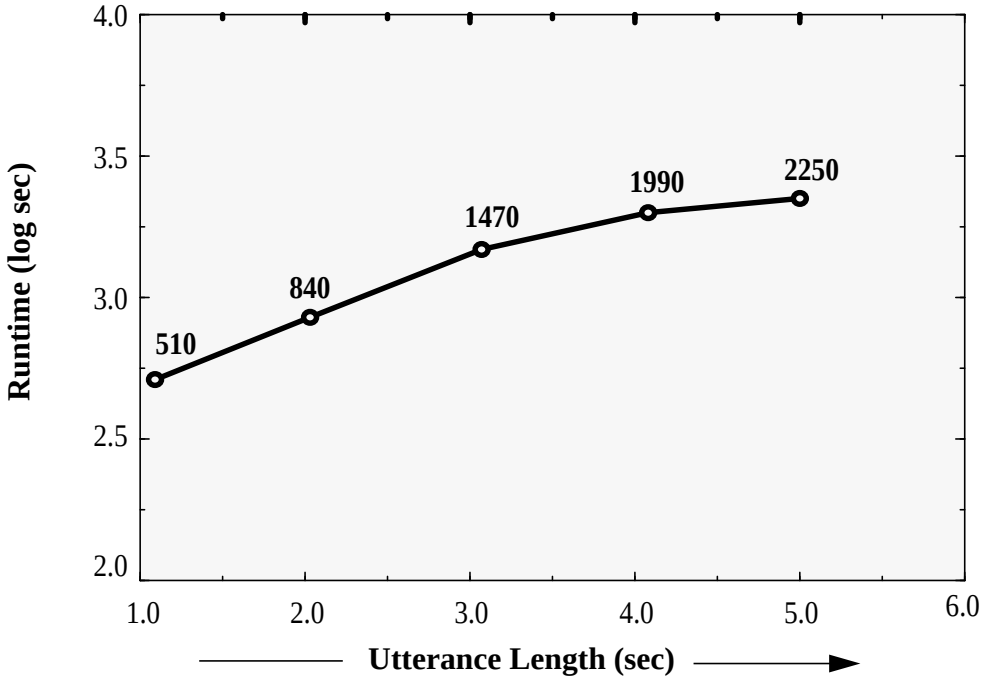


Figure 6. Resource usage for word-graph generation using cross-word triphones and a bigram (Numbers on the plot indicate actual processing time.)

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 in the decoding process causing the word error rate to degrade significantly. On the other hand, if the pruning is too relaxed, then a large number of competing paths are allowed to propagate. Table 1 shows the performance of word graph generation using a bigram LM and context-dependent models under certain pruning thresholds. 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.

4.3. Effect of Pruning

The computation time is directly proportional to the memory required by the system — the more memory the system needs, the more computation time is required to search through the hypotheses occupying this memory. Figure 7 shows the effect of beam pruning on recognition accuracy and computation time. In these, 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.

MAPMI pruning also has a critical impact on decoder performance. It has a direct bearing on the memory usage of the decoder, since the number of active paths at each frame is directly proportional to the number of acoustic models active at that frame. Memory usage is especially critical in word graph generation and grammar decoding, as illustrated in Figure 9. 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 instances is allowed to propagate. As a result, the memory required for the example utterance dropped to 135 MB. The decoding time also improved from 650 xRT to 330 xRT.

Figure 8 shows the overall effect of MAPMI pruning on decoder performance and efficiency during rescoring of word graphs. In this case, the pruning beam widths were fixed at 300, 150 and 150 respectively for the state, phone and word levels. 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 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.

To evaluate the performance of the fast match pruning, we compared it against beam pruning on a small subset of the OGI Alphadigits Corpus. We chose 26 utterances from the official Alphadigits Corpus test set with lengths ranging from 1.3 seconds to 6.1 seconds, to get a measure of the dependence of each method on the data. It was found that the fast-match search produced a much

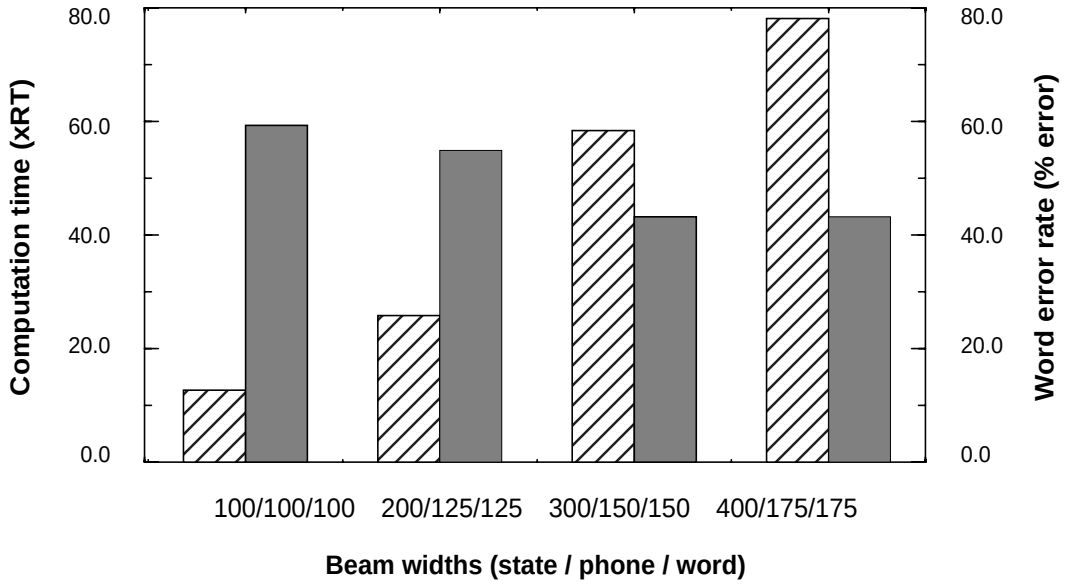


Figure 7. Effect of beam widths on the recognition accuracy and complexity of the search, on a subset of the SWB corpus for word graph rescoring with cross-word triphone acoustic models.

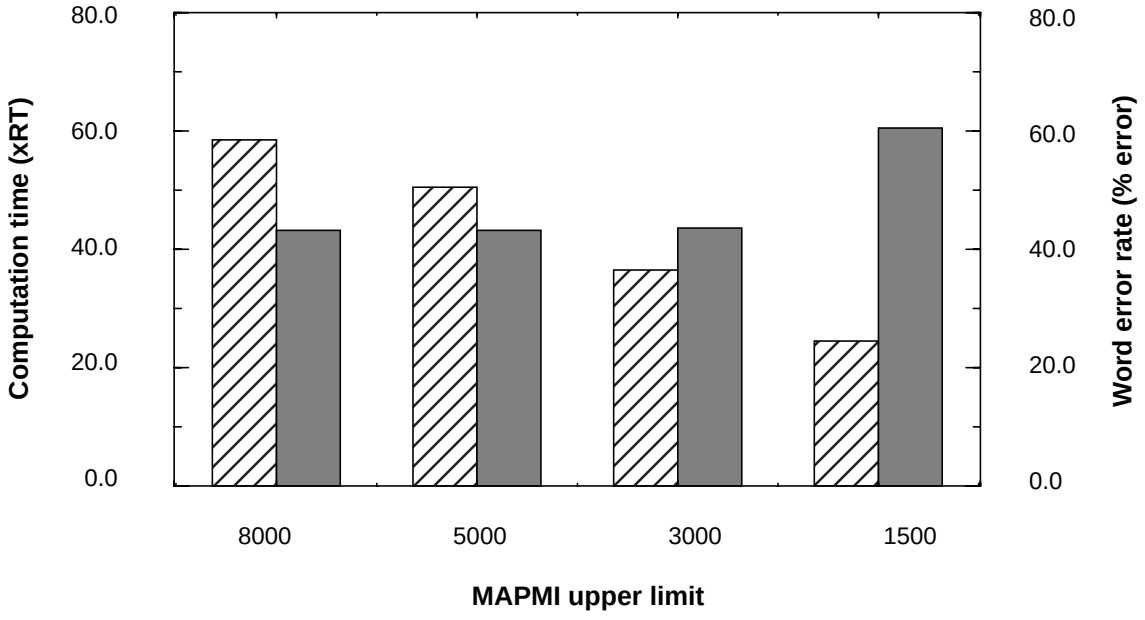


Figure 8. Effect of MAPMI pruning on the recognition accuracy and complexity of the search, on a subset of the SWB corpus for word graph rescoring with cross-word triphone acoustic models.

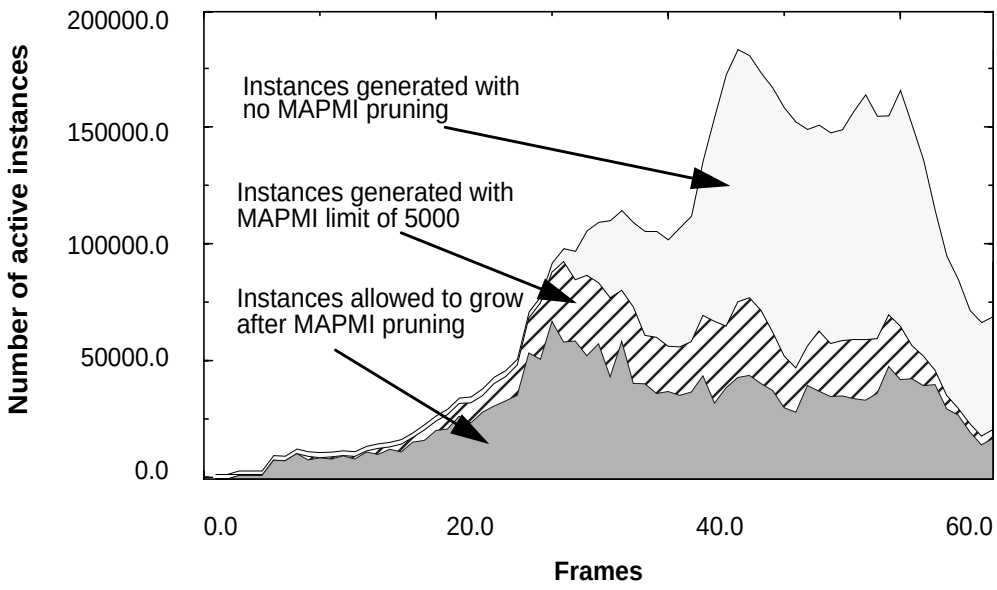


Figure 9. Effect of MAPMI pruning on memory usage as illustrated on a 68 frames long utterance from the SWB corpus for word graph generation using cross-word triphone acoustic models and a bigram language model.

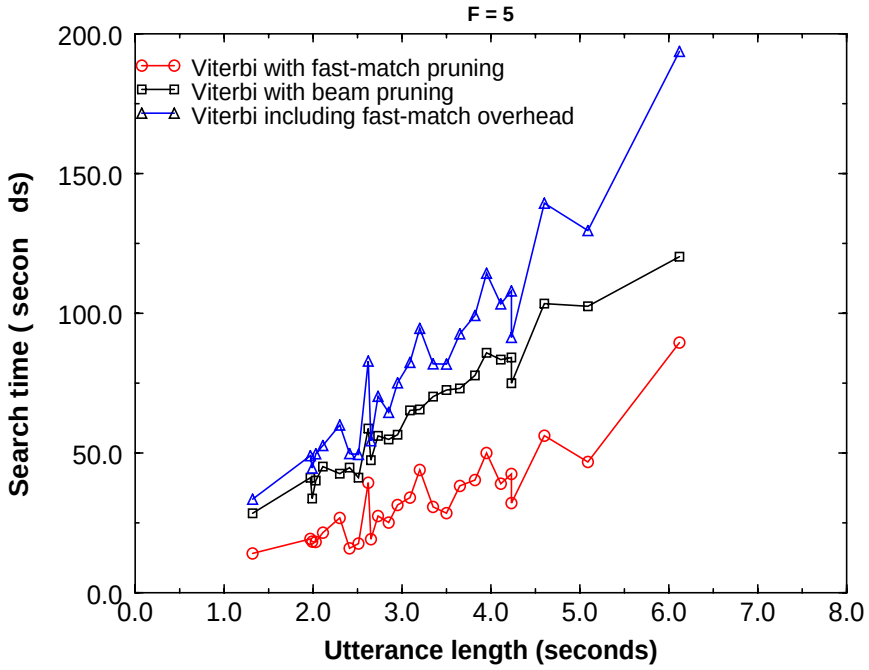


Figure 10. Comparison of search efficiency when using the fast-match look-ahead and efficient beam pruning. The fast-match look-ahead produces a more restrictive beam than the beam pruning technique, but the overhead required is excessive. Clearly a more efficient look-ahead algorithm could improve the efficiency and quality of the search.

tighter beam with little effect on WER. The fast-match scheme also gave a second pass which was significantly faster than Viterbi with beam pruning. However, the overhead required to find the thresholds in the first pass was substantial, as demonstrated in Figure 10.

5. SUMMARY

In the past year, we have efficiently implemented N-gram decoding, which typically is very resource-intensive in the LVCSR system. Our N-gram decoding using bigrams has given a word error rate (WER) of 53.2% on a subset of SWB, which is comparable to WER's measured by similar research systems using comparable data and models. Also, we overhauled the search strategy to support generation of word-graphs. Our improvements allow the system to generate rich word graphs -- the measured word graph WER is 15.6%, a very encouraging number. Using the word graph we generated for the WS'97 SWB test set, we achieved a WER of 47.3% on word-graph rescoring using a bigram LM, and 45.6% on word-graph rescoring using a trigram LM. Word-internal word-graph generation runs at ~200 xRT, and cross-word word-graph generation runs at ~400 xRT. In addition, the advanced pruning techniques we used have been proven to effectively reduce the search space and improve the search speed, while maintaining system performance.

In summary, the progress we have made for this work is encouraging in terms of recognition accuracy and speed. However, there is still much work to be done before large vocabulary tasks can be performed in real-time on a DSP platform. Much of this work will involve developing efficient structures to hold the complex language models associated with continuous speech recognition, and computing the acoustic scores and language model scores more efficiently.

6. FUTURE DIRECTIONS

Future directions in searching can be summarized in one goal: 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 we explored in this work can be transformed into systems that operate at about 10xRT [25]. 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. Look-ahead and N-best strategies at all levels of the system are key to achieving such large reductions in the search space. We also expect significant gains by changing the way we handle pruning of instances [14]. In the past, before processing a frame of data all active instances are collected and pruned based on a maximum allowable number specified by the user. Analysis of the system developed at other research sites shows that instead of pruning instances based on a global instance count, pruning at the lexical node level is far more effective [21]. 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 consumer of CPU time, speeding up the Gaussian

distance computation can have dramatic effect on the decoding speed. In [25], a decision tree based fast Gaussian computation (FGC) algorithm is proposed. The idea is to build a decision tree for the Gaussians in the system using binary clustering where similar Gaussians go into the same leaf node. When decoding, it only needs to take $2 \cdot depth$ distance calculations to find the leaf node or the group of Gaussians associated with a state; then, it can quickly find the most likely Gaussian from this group of Gaussians. Another way to speed up Gaussian computation is based on Vector Quantization [27], where the speech frames are vector quantized and the list of Gaussians at each VQ cell are precomputed. It has been shown that FGC techniques successfully improved the decoding speed with only a little increase in the WER [25-27].

In all, with some trade-off between speed and accuracy, the ISIP decoder can be made much faster. Now with greater knowledge than we had one year ago, we believe we can improve the efficiency of our ISIP decoder further. All of the software for this work is implemented in C++ using the public domain GNU compiler and is available at [28].

7. ACKNOWLEDGMENTS

We wish to acknowledge Dr. Yu-Hung Kao at Texas Instruments, Inc., for his continuous support of this project.

8. REFERENCES

[1] N. Deshmukh, A. Ganapathiraju and J. Picone, “Hierarchical Search for Large Vocabulary Conversational Speech Recognition,” to appear in *IEEE Signal Processing Magazine*, September 1999.

[2] 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.

[3] J. J. Odell, V. Valtchev, P. C. Woodland and S. J. Young, “A One-Pass Decoder Design for Large Vocabulary Recognition,” *Proceedings of the DARPA Human Language Technology Workshop*, pp. 405-410, March 1995.

[4] 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.

[5] N. Deshmukh, A. Ganapathiraju, J. Hamaker and J. Picone, “An Efficient Public Domain LVCSR Decoder,” *Proceedings of the Hub-5 Conversational Speech Recognition (LVCSR) Workshop*, Linthicum Heights, Maryland, USA, September 1998.

[6] 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.

[7] 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.

[8] L.R. Rabiner and B.H. Juang, *Fundamentals of Speech Recognition*, Prentice Hall, Englewood Cliffs, New Jersey, USA, 1993.

[9] M. Ravishankar, "Efficient Algorithms for Speech Recognition", Ph.D. Thesis, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA, 1996.

[10] P. Woodland, et.al., HTK Version 1.5: User, Reference and Programmer Manuals, Cambridge University Engineering Department & Entropic Research Labs. Inc., 1995.

[11] 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.

[12] 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.

[13] J. Picone, "Fast Recognition Techniques for Large Vocabulary Recognition", Institute for Signal and Information Processing, Mississippi State University, December 1997.

[14] A. Ganapathiraju, N. Deshmukh, J. Hamaker, V. Mantha, Y. Wu, X. Zhang, J. Zhao and J. Picone, "ISIP Public Domain LVCSR System," *Proceedings of the Hub-5 Conversational Speech Recognition (LVCSR) Workshop*, Linthicum Heights, Maryland, USA, June 1999.

[15] E. Eide and L. Bahl, "A Time-Synchronous Tree Based Search Strategy in the Acoustic Fast Match of an Asynchronous Speech Recognition System," *Proceedings of the International Conference on Spoken Language Processing*, pp. 2915-2918, Sydney, Australia, November 1998.

[16] S. Ortmanns, H. Ney and A. Eiden, "Language Model Look-ahead for Large Vocabulary Speech Recognition", *Proceedings of the Fourth International Conference on Spoken Language Processing*, pp. 2095-2098, Philadelphia, Pennsylvania, USA, October 1996.

[17] J. Zhao, X. Zhang, A. Ganapathiraju, N. Deshmukh, and J. Picone, "Decision Tree-based State Tying for Acoustic Modeling," Institute for Signal and Information Processing, Mississippi State University, June 1999.

[18] 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.

[19] 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.

[20] J. Zhao, J. Hamaker, N. Deshmukh, A. Ganapathiraju and J. Picone, "Fast Search Algorithms for Continuous Speech Recognition", *Proceedings of the IEEE Southeastcon*, Lexington, Kentucky, USA, March 1999.

[21] M, Finke, J. Fritsch and D. Koll, "Modelling and Efficient Decoding of Large Vocabulary Conversational Speech," *Proceedings of the Hub-5 Conversational Speech Recognition (LVCSR) Workshop*, Linthicum Heights, Maryland, USA, June 1999.

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

[23] 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*, San Francisco, California, USA, pp. 517-520, March 1992.

[24] R. Rosenfeld, "A Maximum Entropy Approach to Adaptive Statistical Language Modeling," *Computer, Speech and Language*, vol. 10, pp. 187-228, 1996.

[25] J. Davenport, L. Nguyen, S. Matsoukas, R. Schwartz and J. Makhoul, "1998 BBN Byblos 10x System," *Proceedings of the DARPA Broadcast News Workshop*, Hilton, Virginia, USA, February 28-March 3, 1999.

[26] J. Fritsch and I. Rogina, "The Bucket Box Intersection (BBI) Algorithm for Fast Approximative Evaluation of Diagonal Mixture Gaussians," *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 837-840, Atlanta, Georgia, USA, May 1996.

[27] T. Colthurst, O. Kimball, H. Gish, et. al., "Speeding Up Recognition," *Proceedings of the Hub-5 Conversational Speech Recognition (LVCSR) Workshop*, Linthicum Heights, Maryland, USA, June 1999.

[28] *http://www.isip.msstate.edu/projects/speech/*, Institute for Signal and Information Processing, Mississippi State University, May 1999.

APPENDIX A. SOFTWARE OVERVIEW

A.1. Utilities Usage

The ISIP utility is called trace_projector. The usage of it is:

trace_projector -p < param_file > { options }

options: -demo

outputs partial hypothesis after processing a user-specified number of frames.

A.2. Parameters

- function: forced_alignment, lattice_rescoring, ngram_decoding, lattice_generation, lattice_verification (lattice rescoring with a different LM), or lattice_to_lattice (lattice rescoring with a different acoustic models)
- monophones_file: context-independent model names
- transitions_file: file containing the transitions used in the acoustic models
- states_file: HMM state data (Gaussians)
- models_files: context-dependent model map
- input_format: ascii or binary (for feature files)
- lexicon_file: lexicon
- lm_file: ngram language model
- context_mode: monophone, word_internal or cross_word
- history_paths: number of word back-pointers for lattice generation
- nbest_paths: number of partial hypotheses in demo mode
- beam_pruning: < state_beam > < model_based > < word_end_beam >
- mapmi_limit: maximum number of active model instances
- align_mode: word, model or frame (output formats)
- transcription_file: transcription file for forced alignment
- input_lattice_list: list of input lattices for lattice based modes
- output_lattice_list: list of output lattice files for modes generating lattices
- mfcc_list: list of input feature files
- output_list: list of output files

```
# parameters file for decoder demo
#

# define function mode
#
# function = forced_alignment
# function = lattice_rescoring
function = ngram_decoding
# function = lattice_generation
# function = lattice_verification
# function = lattice_to_lattice

# acoustic models data
#
monophones_file = monophones.text
transitions_file = trans.text
states_file = states.bin_sparc
models_file = models.text
phones_file = phones.text

# define lexicon file
#
lexicon_file = lexicon.text

# alignment mode
#
align_mode = word

# define ngram language model
#
lm_file = bigram.text
ngram_order = 2
lm_scale = 12.0
word_penalty = -10

# define context mode
#
context_mode = word_internal

# define number of paths
#
history_paths = 1
nbest_paths = 1

# define pruning thresholds
#
beam_pruning = 300 150 150
mapmi_limit = 3000
num_allowed_words = 50

# define input and output lists
#
mfcc_list = ../lists/mfcc.list
output_list = ../lists/output.list
# define input format
#
input_format = ascii

# end of file
#
```

Figure A1. Example format of the parameter file used with the ISIP decoder.

APPENDIX B. DATA FILE FORMATTING CONVENTIONS

The ISIP decoder requires the input data such as the lexicon, lattices (word graphs) or N-grams, the mfcc features, and the HMMs used for the acoustic models to adhere to specific file formats to work correctly. This section describes the various formatting conventions used.

B.1. The Lexicon File Format

The lexicon consists of the words and pronunciations (phonetic transcriptions) that define the speech recognition task applied to the ISIP decoder. The lexicon contains a single word on each line followed by its phonetic transcription. The word and its pronunciation are separated by one or more tab-spaces, while individual phones in the transcription are separated by one white space each. If a word has multiple pronunciations, all the corresponding word-pronunciations entries must appear on consecutive lines in the lexicon. The words may or may not be in alphabetical or any other sorted order after the first four required entries; however, the first four words in the lexicon are fixed to be sentence start (usually denoted as !SENT_START), sentence end (typically !SENT_END), the short pause [S] and silence [SILENCE], respectively. Blank lines, and lines beginning with a *pound sign* (#) are ignored as comments. Figure B1 shows a sample of the lexicon file format.

B.2. The Word Graph File Format

Both the input word graph (in word graph rescoring mode) and the output word graph (in word graph generation mode) are of the same format, as shown in Figure B2.

B.3. The N-gram File Format

Figure B3 shows a bigram language model. The higher-order language model is in a similar format. For the higher-order gram, such as the bigram in the bigram file, the first column is the grammar score, the last column is the back-off score. If the bigram for a word pair doesn’t exist, it will back off to its unigram.

B.4. The MFCC File Format

The number of features to be used to evaluate each frame in the ISIP decoder is read from the states file. The input mfcc file simply contains the requisite number of feature values on each line, and each line corresponds to the feature vector for one frame of acoustic data. No comments or any other characters are allowed in this file.

B.5. The Monophones File Format

The monophones file format is somewhat similar to the lexicon file format. Besides the introductory comments, it contains a line with the number of monophones followed by a list of all the monophones used. See Figure B4 for an example, and note the order of the first few monophones that represent sentence boundaries, short pause and silence.

B.6. The Triphone List Format

The ISIP decoder enlists all the phones that form the acoustic model set in a format illustrated in Figure B5. The first non-comment line gives the number of phones in the list. Subsequently, each line contains one phone entry. This consists of the symbol used for the phone, followed by the context size (*e.g.*, 1 for context-independent phones, 2 for diphones and 3 for triphones, *etc.*). If the context size is n, the next n entries on that line correspond to the integer indices x_i of the monophones that constitute the n-phone (The indices are synchronized with the monophone list.). This is followed by an integer N that uniquely indexes the current phone, calculated as

$$N = \sum_{i=0}^{n-1} x_i M^i \tag{B.1}$$

where M is the total number of monophones. The final entry on the line is a number that maps the phone to the appropriate HMM. It is possible for different phones to be mapped to the same HMM due to model tying.

B.7. The Models File Format

The models file contains the total number of HMMs used in the system, followed by a listing of each model. The models are referenced by indices, and the indices need not be in consecutive order in the file. The file format is displayed in B6. Note that because of state-tying, the same state may be referenced by more than one model.

B.8. The State Transitions File Format

The transition likelihoods for each state are stored as matrices in a file as shown in Figure B7. Different models may have different numbers of states, and therefore, different transition characteristics. It is also possible for different states to share the same transition matrix. The states file, which is in binary format (All the files discussed in this appendix are in ASCII text.), contains reference indices to the appropriate transition matrices.

B.9. The Output File Format

In word graph rescoring, forced alignment, and N-gram decoding mode, the decoder outputs the total resources consumed in terms of the total number of path markers created, destroyed and active at the end of the decoding process, along with the phone and word alignment of the input utterance as recognized. An example is shown in Figure B8. The alignment format is: start frame, end frame, path score for that duration, phone symbol, and a word symbol, and word-score if it is also a word end. In word graph generation mode, the output file is a word graph, whose file format is the same as the format of the input word graph. Thus, it can be used as the input word graph in the word graph rescoring mode.

```
# file: swb_lexicon.text
#
# lexicon for this application
#
# format: word phonetic_pronunciation
#
# note that !SENT_START is always first, !SENT_END al-
ways # second, [S] # always the third and [SILENCE] the
fourth word # in the lexicon
#

!SENT_START     sil
!SENT_END       sil
[S]             sp
[SILENCE]       sil
 •••
ABOUT           ax b aw t sil
ABOUT           ax b aw t sp
 •••
ZUNIGA'S        z uw n ih g ax z sil
ZUNIGA'S        z uw n ih g ax z sp
ZURICH          z uh r ih k sil
ZURICH          z uh r ih k sp
```

Figure B1. Example format of the lexicon file used with the ISIP decoder.

```
VERSION=1.0
UTTERANCE=some_utterance_id
lmname=some_language_model_id
lmscale=12.00
wdpenalty=-10.00
vocab=lexicon.text
hmms=list_of_hmms_used
N=6536 L=65367
I=0 t=0.00
I=1 t=0.02
I=2 t=0.03
 •••
I=6534 t=3.03
I=6535 t=3.05
J=0 S=0 E=1 W=!SENT_START v=1 a=-315.14 l=0.833 n=0.000
J=1 S=0 E=2 W=!SENT_START v=1 a=-462.96 l=0.833 n=0.000
 •••
J=65363 S=6531 E=6535 W=!SENT_END v=1 a=-306.37 l=-1.608
n=0.000
J=65364 S=6532 E=6535 W=!SENT_END v=1 a=-306.37 l=-0.874
n=0.000
```

Figure B2. Example format of a SWB word graph used with the ISIP decoder

```
\data\
ngram 1=22362
ngram 2=309231

\1-grams:
-6.485136    'VETTE -0.1232755
-0.9280869   !SENT_END
0.361912     !SENT_START   -1.494555
-1.682954    A    -0.4137365
-5.444224    A'S   -0.1925565
-6.485136    ABACK  -0.1774414
-6.158157    ABALONE -0.173158
-5.977332    ABANDON -0.1633228
•••
\2-grams:
-0.4737018   'VETTE !SENT_END
-2.659457    !SENT_START A
-5.892071    !SENT_START ABILITY
-3.509884    !SENT_START ABOUT
-5.892071    !SENT_START ABSENTEE
 •••
\end\
```

Figure B3. Example format of a bigram language model used with the ISIP decoder.

```
# file: monophones.text
#
# monophones for this application
#
# format: monophone
#
# note that !S is always first, !S always second and SP al-
ways # the third and SIL always the fourth in the monopho-
nes list
#

num_monophones = 46

!s
s!
sp
sil
aa
ae
•••
y
z
zh
```

Figure B4. Example format of the monophones file used with the ISIP decoder.

```
# file: phones.text
#
# list of all possible triphones for this application of alphadig-
its
# the second column is the phone index based on the phone
content of
# the triphone. the third column is the index of the hmm mod-
el
# representing this triphone.
#

num_phones = 81314

aa 1 4 184 809
aa+aa 2 4 4 8648 170
aa+ae 2 4 5 10764 170
aa+ah 2 4 6 12880 170

 •••
zh-zh+y 3 45 45 43 93103 165
zh-zh+z 3 45 45 44 95219 165
zh-zh+zh 3 45 45 45 97335 165
```

Figure B5. Example format of the file containing a list of all the phones used with the ISIP decoder.

```
# file : models.text
#
# data for HMMs
#

num_models = 20943

index: 0
phone: !S
num_states: 2
transitions: 0
states: 0 0
 •••
index: 47
phone: dh-t+ow
num_states: 5
transitions: 24
states: 0 1619 5360 5304 0
 •••
index: 20939
phone: dh-iy+dh
num_states: 5
transitions: 37
states: 0 376 5207 93 0
```

Figure B6. Example format of the HMM list file used with the ISIP decoder.

```
# file : transitions.text
#
# data for acoustic models --- list of all the transition probs

num_transitions = 45

0. 2
 0.000000e+00 0.000000e+00
 0.000000e+00 0.000000e+00

1. 5
 0.000000e+00 1.000000e+00 0.000000e+00 0.000000e+00
0.000000e+00
 0.000000e+00 5.504225e-01 4.495775e-01 0.000000e+00
0.000000e+00
 0.000000e+00 0.000000e+00 6.527667e-01 3.472333e-01
0.000000e+00
 0.000000e+00 0.000000e+00 0.000000e+00 6.171211e-01
3.828789e-01
 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
0.000000e+00
 •••
44. 3
 0.000000e+00 1.295657e-01 8.704343e-01
 •••
```

Figure B7. Example format of the file containing state transition matrices used with the ISIP decoder.

```
Time  =  92

     Word  Phone  State  Total
Live     2680   7413  47200  57293
New    218143  535712 7115272 7869127
Deleted  215463  528299 7068072 7811834

0000 0019 !SENT_START  -3907.043502
0019 0031 I'LL      -1052.150754
0031 0071 TAKE      -2953.049042
0071 0090 THE      -1240.619770
0090 0092 !SENT_END   -187.536116
```

Figure B8. Example format of an output file for the ISIP decoder in N-gram decoding and word graph rescoring mode.