Final Report on Speech Recognition Research

December 1984 to April 1990

Research Documents Group
J.S. Denton, C.R. Taylor, editors

School of Computer Science
Carnegie Mellon University

This research is sponsored by the Defense Advanced Research Projects Agency, DoD, through DARPA Order 5167, and monitored by the Air Force Avionics Laboratory under contract N00039-85-C-0163. Views and conclusions in this document are those of the authors and should not be interpreted as representing official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or of the United States Government.

Introduction

This report documents speech recognition research conducted at Carnegie Mellon's School of Computer Science. The work was supported by the Information Science and Technology Office of the Defense Advanced Research Projects Agency (DARPA) between December 1984 and April 1990.

Chapter 1 describes Angel, the speaker-independent system we developed for recognizing continuous speech with a large vocabulary. This recognizer uses feature extraction, a knowledge-based approach that we had successfully employed under the preceding contract, when we developed the first speaker-independent system capable of recognizing isolated letters with 89% accuracy.

For integrating the components of our knowledge-based recognizer, we built an extensive software support environment, which is presented in Chapter 2. In addition, this chapter describes our hardware accelerator design for Angel's computationally intensive search.

In 1987, we began exploring a stochastic approach to speech modeling and built a recognizer that outperformed Angel and state-of-the-art systems at other sites. Encouraged by this success, we shifted away from the knowledge-based method and dedicated ourselves to further develop and refine Sphinx.

Chapter 3 describes Sphinx' hidden Markov model approach and our basic system's operation. It also highlights the extensions and refinements we incorporated that enable our speaker-independent Sphinx to recognize continuous speech in a 997-word task with 96% word accuracy.

Sphinx' performance enabled us to begin research in three new areas that are discussed in Chapter 4 and aim at:

- Designing voice-driven interfaces for problem-solving applications
- Parsing poorly pronounced and ill-structured spontaneous speech
- Enhancing automatic speech understanding through dialog modeling.

When emphasis shifted to stochastic systems, we retargeted our hardware accelerator to Sphinx' search, with the goal of making the architecture adaptable to any large, computation-intensive application. This effort is described in Chapter 5.

In the context of both approaches to speech recognition, we explored techniques that make recognizers robust to the speakers' acoustic variations and tolerant to noisy office environments. This work is the subject of the final chapter.

1. A knowledge-based speech recognition system

Angel, our knowledge-based recognition system, is composed of a signal processor and separate components for generating acoustic/phonetic units, combining the phonetic units into word hypotheses, and generating sentences from the hypothesized words. Each subsystem activates the following one with the data it produces. Only the word and sentence hypothesizers exchange data in both directions.

1.1 Signal processing

We initially used a wirewrapped version of a signal processor designed for our previous recognizer. Based on this prototype, a programmable hardware signal accelerator was later built for Angel. This accelerator processes the raw signal through low-pass filtering at 8 kHz followed by digitizing at 16 kHz. The digital representation is then transformed into the system's basic graphic/visual representation, a DFT computed every 3 ms. Since the signal-processing component was taken over from the previous system and not modified during this report period, we will not discuss it further.

1.2 Acoustic/phonetic subsystem

This subsystem analyzes the data produced by the signal processor, attempts to recover the phonetic structure of the spoken utterance, and generates hypotheses of acoustic/phonetic units. Due to Angel's bottom-up approach, this component's output affects, to a large degree, the amount of branching and searching required by the word and sentence components and, hence, the recognition accuracy of the entire system.

Our group's goal was to develop a subsystem that could achieve a spectrogram reader's accuracy. As Researchers at Carnegie Mellon and MIT had demonstrated, expert spectrogram readers could locate almost all segments identified by phoneticians (97% for continuous speech) and rarely had to rely on higher-level linguistic knowledge, such as syntactic or semantic structure, for making their interpretations.

The approach we chose for analyzing the signal processor's graphic/visual representations is feature-based. It combines automatic methods with specific speech knowledge and proceeds in the following manner. We examine the signal representation and identify prominent, speaker-invariant continuous features, such as formant transitions, zero crossings, pitch, energy profiles, peak-to-peak amplitude, and formant frequency. Next, we develop rule-based algorithms for determining the ''signal location'' (begin- and end time) of the selected features and extract the parameters. The final step in this procedure consists in training ''classifiers'' with the feature-segment parameters. These classifiers hypothesize segments according to certain feature combinations that represent a particular phonetic category.

In addition to developing automatic Gaussian classifiers, we used ''Classgraph,'' an interactive tool, throughout the Angel project. This tool was developed for our previous recognizer and helps in making phonetic decisions about specific target segments (segment sets with common acoustic properties) when the feature data do not follow a typical Gaussian distribution but are binary or multimodal, or have ‘‘flag’’ values. We simply display the data with Classgraph and incorporate optional features that provide strong evidence about a particular category but are present only some of the time. Toward the end of this project we also used this tool to build automatic multivariate-Gaussian classifiers.

1.2.1 A four-module prototype

We designed a prototype subsystem [Cole 86] composed of four independent modules, each of which processes a particular phonetic target group of incoming waveform features. This division of labor allowed each of our team members to gain expert knowledge in one phonetic category. Furthermore, phonetic grouping reduced the number of segments each module’s classifier had to label and, within a target category, the number of features required for making fine phonetic distinctions.

Three of the modules process speech events, the fourth decodes nonspeech segments. The *sonorant module* analyzes voiced segments, specifically sonorants and vocalics; the *fricative module* handles unvoiced, or fricative events; and the *stop module* deals with stop and affricate consonants that precede sonorants. The *closure/silence module* handles such nonspeech segments as closures, background noises, and pauses. Each module carries out the entire feature-based analysis for its group of target segments: It locates begin and end times of those continuous features that characterize the target group and extracts feature parameters; it trains its classifier on these parameters; and, finally, it hypothesizes segments, assigns labels and probabilities to the hypothesized segments, and combines these labeled hypotheses into a lattice.

Evaluation

We evaluated our four-module subsystem with 100 sentences produced by 10 speakers: 50 phonetically balanced sentences and 50 sentences from then electronic mail task. These test sentences were not used to train the modules. We compared each module’s locator/classifier output to the corresponding hand-labeled target segment and had the following results:

- Sonorant module—located 98% of the 1209 sonorant targets with first-choice classification of 56%
- Fricative module—located 96% of the 361 target segments with first-choice classification of 51%
- Stop module—located 96% of the 314 target segments with first-choice classification of 54%
- Closure module—located 98% of the 628 target regions with first-choice classification of 95%

Refinements

To improve the performance of our four-module prototype, we made the following modifications:

We built a separate locator and classifier for each sonorant allophone and added context-based, vowel-specific knowledge to improve formant tracking. In addition, we modified the sonorant module’s algorithm to represent glottalized regions of the speech signal so that they can be located separately. This information of glottalized segments produced cleaner, higher-quality sonorant segments and improved the first-choice classification accuracy of our speech-events modules. It also was useful for the word hypothesizer, since glottalization primarily occurs at word boundaries and sentence ends.

A major problem of the closure module was that it frequently identified noises (for example, door slams and coughs) as utterance beginnings. To reduce the number of such ‘‘false alarms,’’ we designed an algorithm that differentiates between nonspeech (noise) segments and actual utterance beginnings. This algorithm is based on our knowledge that most nonspeech events lack sufficient formant structure to qualify as speech events and refers to rules about such features as low- and high-frequency energy and silence durations.

Furthermore, we integrated the four-lattice output that contained many overlapping and contradictory phonetic segments. While the word hypothesizer had developed and was using its own lattice-integration procedure, our module was better equipped to resolve conflicting acoustic/phonetic hypotheses. We developed a rule-based postprocessing algorithm that applies context-specific information to sort all segments found in the lattices into sonorant, fricative, stop, and silent groups. These grouped segments were then combined in a network that provides information about phonetic sequences. No path probabilities were included—all transitions between nodes were considered equally likely.

1.2.2 A two-module subsystem

While our integration procedure improved the output and made it more usable for the word module, it did not solve the following basic problems:

- Poor identification of boundaries between phonetic segments
- Classification of some segments by more than one module, resulting in contradictory and overlapping output
- No classification of those segments that did not fit well into a particular phonetic group.

Since these problems are primarily due to a lack of information sharing among the four modules, we redesigned the acoustic/phonetic subsystem and unified segmentation, broad classification, and network generation of all phonetic categories in one module. Its output is passed to a second module that reclassifies fricatives and sonorants [Chigier 88].

To ensure that our network contains the ‘‘correct’’ phonetic segments as identified by the spectrogram readers, we used MIT’s *segmentation* algorithm. This method examines the signal from left to right, determines similarities between adjacent spectral regions, joins similar regions, establishes region boundaries, and generates, for each utterance, a dendrogram hierarchy of alternative segmentations ranging from most similar to most dissimilar.

The located segments are then passed to the *broad classifier*, which we built with the internal classification components of our four-module design. This classifier sorts all segments into three broad classes of sonorants, fricatives, silences/closures, ranks the broadly classified segments, and assigns labels and corresponding probabilities.

While the hierarchical segmentation technique produces the correct segments, it also creates a large search space. To control excessive branching, we used three pruning methods in our *network-generating process*:

Our first method uses a fixed selection band to prune segments. We set the band’s minimum similarity threshold and maximum dissimilarity limit so that segmentation accuracy would be high and the branching factor low. Yet, while this technique considerably reduces the number of alternative segments, it also prunes many correct segments. For example, some fricatives are excluded since their spectral representation tends to join higher in the dendrogram than that of sonorant regions.

By varying the band dynamically according to spectral-distance information—which includes widening the band during fricative regions, we generated a network that captures a higher percentage of correct segments while its branching factor is only slightly higher than that of the fixed-band method.

To also include flap- or dip-like events that the variable-band network generator misses, our third pruning algorithm is based on the dynamic-band method to which adds a set of postprocessing rules. These rules are derived from knowledge about the signal (such as duration, similarity of region spectra, and waveform contours) and information obtained in the broad-classification procedure.

These networks are then analyzed by the *fine classification module*, which looks for and classifies more precisely all stop/fricative and sonorant segments. Closure/silence segments are not further refined. Since stops and fricatives tend to have few Gaussian features, we used a combination of Classgraph and Gaussian classifiers for fine classification. Due to the inherent Gaussian nature of sonorant features, the tree of Gaussian classifiers that we used for the sonorant segments proved very successful.

Evaluation

We evaluated our two-module system on a set of 200 new test sentences covering the Resource Management task. These sentences from the TIMIT acoustic/phonetic database had not been used previously for testing or developing our algorithms. The network generated with the fixed selection band contained 88.2% of the correct segments. Correct segments increased to 91.1% with the varied band and to 93.3% when we added postprocessing rules. The latter result compares well with that of the unpruned dendrogram that contained 94.2% of the correct segments. Furthermore, pruning the network with postprocessing rules reduced the branching factor of 4.5 in the unpruned dendrogram to 1.8.

Our broad classification procedure initially achieved 90% first-choice accuracy. By modifying the parameters of the Gaussian broad classifiers to fit the feature-data distributions more closely we increased this accuracy to 97%.

These results are encouraging since they reflect the accuracy of our integrated approach. Unfortunately, we cannot compare the performance data of our integrated two-module subsystem with our prototype's results, which indicate the performance of each separate module.

We concluded our work in acoustic/phonetic analysis by building an automatic tree-based classifier. This design combines Gaussian classifiers and Fisher discriminant functions for developing the tree structure, selecting features at each node, and performing the final classification. Compared to the manually-designed tree structure, this automatic procedure provided a 3% improvement in stop-consonant classification. It is also faster and more reliable, since it is not subjective.

1.3 Word Hypothesizer

The word component's input consists of the labeled phonetic segments generated by the acoustic/phonetic subsystem. Our approach to producing word hypotheses from this input is based on the following conviction: Given a good acoustic/phonetic analysis of the speech signal, we can hypothesize words in completely bottom-up fashion and achieve high accuracy without applying any higher-level constraints, such as restrictive grammars or narrow task definitions. We only need to supplement the information contained in the acoustic/phonetic transcription with alternative knowledge about the signal's properties and with stored knowledge about pronunciation variations caused by continuous speech and coarticulatory effects.

The word hypothesizing strategy we developed traverses the phonetic label sequence from left to right and matches it to possible pronunciations specified by a lexicon. Over the course of this project, we implemented several matching algorithms and created our own lexicon, an extensive knowledge base of word-level, continuous-speech phenomena. Some of our research effort was spent on developing a ''lattice integrator'' to make the four lattices initially produced by the acoustic/phonetic subsystem compatible with our matching algorithms. Furthermore, we built a ''word-juncture verifier'' that exchanges information with the sentence hypothesizer and assists in making decisions about logical word sequences.

1.3.1 Lattice integrator

The initial output produced by the acoustic/phonetic component consisted of four lattices containing many overlapping segments and contradictory hypotheses. To merge these lattices into a single linear sequence, we first had to resolve the conflicts. We used phonetic rules to control the stretching and truncating of overlapping segments so that we could align their endpoints. To settle contradictory hypotheses, we developed a rule-based arbitration scheme. Finally, we recalculated the probabilities of the overlapping and contradictory segments and integrated the relabeled segments.

When the acoustic/phonetic group switched to producing a network of labeled and ranked phones, we discarded our integrator and modified our matching algorithm to accept the already integrated input.

1.3.2 Matching algorithms

Our matching algorithm controls the number of possible word candidates by using a single pruning parameter that determines the search beamwidth. This parameter is initially preset and dynamically recalculated according to the score of the best existing path: It narrows when the input is of high quality and widens for input regions of uncertainty. We found that a normalized beam with time-weighted likelihood controls the search most effectively. To further limit the number of hypotheses we developed a *hypotheses-filtering technique* that assigns penalties to paths based on knowledge about expected phone durations and the occurrence frequency of pronunciation variants. Each hypothesized word is assigned a score that reflects the sum of all its phones' arc probabilities.

The first matching algorithm we developed and implemented was based on *anchor-selection*. As the task vocabulary increased from 325 to 1000 words, this method became ineffective and we developed an *unanchored matching* algorithm that initiates a search on every phonetic segment of the input lattice. When the acoustic/phonetic module produced a network of phonetic labels, we modified the unanchored matcher and the word-juncture verifier to accept this type of input.

Anchor generation

The main purpose of anchor-based matching [Rudnicky et al. 87] is to reduce the number of positions from which we begin searching in the phonetic lattice. The strategy we developed for identifying possible word locations in the signal is based on our knowledge that words are composed of syllables containing a vocalic center (nucleus) and that syllable and word breaks must occur in regions between nuclei. Our anchor-generating algorithm segments the utterance into vocalic centers and boundary regions. The syllable nuclei are then expanded into ‘‘anchor regions’’ that are matched to the lexicon.

For identifying syllable nuclei we used an improved version of the recursive Zapdash algorithm previously developed at Carnegie Mellon. This procedure coarsely segments the speech wave based on four parameters extracted every centisecond (peak-to-peak and zero-crossing counts for both low-pass and high-pass transforms of the signal) and provides information about vocalic, nonvocalic, true silence regions, and energy dips within vocalic regions. Syllable nuclei are located in those regions that exhibit energy typical of vocalic peaks. Our segmentation procedure misses fewer than 1% of all syllable nuclei and is speaker-independent and robust over a large dynamic range.

Finally, the selected syllable nuclei are combined to form anchor regions that have a *begin boundary* before a vocalic center, an *end boundary* after the last nucleus, and one or more vocalic centers. Ambiguities in regions and word boundaries are settled by applying specific rules based on contextual information.

Anchor-based matching

The anchor regions are phonetically transcribed and matched against the lexicon's phonetic network. The time-synchronous matching search expands the search tree segment by segment. We modified the beam search to explore paths of different lengths simultaneously and normalized the resulting score of each hypothesis according to its path length. The anchor-based matching algorithm is designed to generate, for a given word in the lexicon, only the ''best'' hypothesis, the one that has the lowest cost for traversing the lexical network.

We first implemented a *single-anchor* matching algorithm that predetermines the length of an anchor and doesn't allow the search to terminate until the anchor's end region is reached. The entire lexicon is searched separately for each anchor region. While this algorithm is simple and straightforward, we found it inefficient for two reasons: First, it applies the entire pronunciation network to each anchor, which results frequently in matching anchor regions with one or two syllables against a multisyllable word. Secondly, while some word regions are included in several anchors and hence are scanned repeatedly, this algorithm does not make the information from one scan available to subsequent scans.

To avoid forcing word hypotheses into inappropriate anchors, we developed a *multiple-anchor* strategy that is somewhat more complex. This procedure generates anchors with fixed begin regions and multiple end regions. Search paths may terminate when the number of nuclei in the anchor region is compatible with the number of syllables in the candidate word.

Evaluation

We evaluated our anchor-based matchers on the 325-word electronic mail task. The single-anchor algorithm located 98% of all syllables with an average lattice depth of 40 words and reduced the amount of computation necessary to generate word candidates by a factor of six when compared to unanchored word spotting. The multiple-anchor algorithm reduced computation only by a factor of three, but it produced a higher-quality word lattice that contained 60% fewer hypotheses than the lattice generated by the single-anchor algorithm.

Unanchored matching

When the system's vocabulary was expanded to handle the 997-word Resource Management task, the accuracy of our anchor-based word hypothesizer dropped. We decided to abandon our anchor-generating approach and developed a matching algorithm that initiates and ends searches on all segments of the linear input lattice [Rudnicky et al. 88a]. Although this algorithm increases the search space and triples the number of hypotheses generated, the unanchored matching algorithm does not miss words due to syllabification failures that occurred in anchor-based matching. To reduce the number of hypotheses, we developed several hypotheses-filtering techniques that use local pruning decisions to eliminate search paths.

1.3.3 The lexicon

Our lexicon consists of word pronunciations stored in the form of phonetic networks. We created it by first assembling a database of basic pronunciations for each of the 997 words in the Resource Management task. To determine these baseforms we either selected them from a dictionary (the Shoup online dictionary or Webster's), generated them with the MITalk letter-to-sound compiler, or derived them from transcriptions of spoken input.

Each baseform pronunciation is then transformed into a phonetic network that includes pronunciation variants caused by continuous speech and coarticulatory effects. We automated this transformation by developing a procedure that applies phonological rules and generates lexical representations of intra-word and word-juncture phenomena. Over the course of this project we derived more than 250 intra-word and word-juncture rules from transcription data. Intra-word rules generate pronunciation variants of phonetic segments and account for the effects of context and stress on these segments. They also assign penalties to those pronunciation alternatives that occur below a certain frequency threshold. Our word-juncture rules model such phenomena as flapping or closure deletions across word boundaries.

The automatic generation of word-pronunciation networks from baseforms is a special accomplishment of our team. We found that this method produces good recognition while keeping the amount of computation manageable. In addition, it greatly simplifies adapting the lexicon to different tasks, particularly those that involve vocabularies of 1000 or more words, and forces us to develop an explicit model of speech phenomena.

The final step in building our lexicon was combining the individual word-pronunciation nets into a single graph. To avoid unnecessary searches, thereby reducing computation, we employed an automatic network-merging procedure that collapses common initial subsequences.

1.3.4 Juncture verification

Since we generate word hypotheses in completely bottom-up fashion without applying syntactic, semantic, or task-based constraints, the output lattice of unlinked words is large. To assist the sentence hypothesizer in selecting words, we developed a module that computes the acoustic/phonetic cost of three types of word junctures: abuts, gaps, and overlaps. Abuts occur where one word hypothesis ends and another begins without overlap or intervening segments. This type of word juncture is ‘‘legal’’ by definition, accounting for about half of all junctures, and we associate no cost with it. Gaps and overlaps, on the other hand, require complex decisions based on a phonetic analysis of those cases where phonetic segments intervene between the end of one word hypothesis and the beginning of another. When two words share one or more segments we speak of an overlap. In general, we disallow gaps that contain significant speech events and overlaps that involve inconsistent interpretations of the speech signal.

Our initial verifier employed a simple table-lookup scheme to find legal word junctures derived from phonetic analyses. To make the verifier computationally more efficient, we derived its information from the data produced by our coarse segmenter. We also replaced table-lookup with a finite-state network compiled from declarative rules. Finally, we modified both our matching and juncture verification algorithms by including prosodic knowledge and new, more precise descriptions of juncture phenomena. This enhanced version of our coarse-segmentation-based verifier identified word junctures more accurately than the initial approach that used phonetic labels and increased performance by 9%.

1.4 Sentence Hypothesizer

Input to the sentence hypothesizer module is a large number of unlinked candidate words annotated with begin- and end times and acoustic/phonetic plausibility scores. The quality of this ‘‘word lattice’’ determines the quality of the sentence hypothesizer’s output: The fewer ‘‘correct’’ words (those actually spoken) are missing from the lattice and the higher the acoustic/phonetic scores of those present, the higher the sentence accuracy will be.

Our basic approach to hypothesizing sentences consists in selecting words from the input lattice, combining them into candidate phrases, and extending these phrases with new words until the entire utterance is parsed. To develop a robust, highly accurate, and computationally efficient parser, we implemented and evaluated algorithms that employ various methods for selecting words, different knowledge sources for constraining the selection, and several knowledge representations. We first implemented a caseframe parser and analyzed its potential benefits. Our second major study centered on word-transition parsers that employ different types of syntactic and semantic knowledge representations. We concluded by comparing left-to-right to island-driven implementations of word-transition parsers.

1.4.1 Evaluating caseframe parsing techniques

Caseframe grammars represent the words of a task domain according to their semantic relationships. Parsing is initiated by identifying a ‘‘head concept,’’ typically the verb, that describes the sentence’s primary semantic meaning or action. Subsequent words are selected according to the role they play in the sentence, relative to the head concept. Since this representation can tolerate syntactic irregularities, caseframe parsing seemed an attractive strategy. In addition, we expected that Angel’s acoustic/phonetic and word modules would hypothesize head-concept words with greater accuracy.

We wrote a caseframe grammar for DARPA’s 200-word electronic mail task and built a corresponding parser [Hayes et al. 86]. Our evaluation revealed that the overall recognition accuracy of this parser was poor. This low performance was largely due to the task itself, which contained many two-syllable words such as `copy` and `forward`. The word module often did not recognize these short words. Therefore major content words were lost and function words (articles and prepositions), which depend on content words, could not be recovered. Another drawback of the caseframe parser was that it was slow in generating sentence hypotheses.

To speed the sentence generation, we manually derived simple phrase-structure rules from the caseframe grammar and compiled them into a *finite-state network*. In addition, we updated the grammar to the 325-word task vocabulary. These phrase-structure rules encode both syntactic information and the semantic information contained in the case frames. The finite-state network provides ‘‘tight’’ semantic interpretations and generates only grammatically correct sentences.

Our finite-state-network parser performed well and executed rapidly when the word lattice contained all words of the spoken sentence. It also inserted function words, as long as no more than one function word was missing between adjacent major content words. However, when correct words were missing from the word lattice, this semantic-network parser was prone to produce incorrect sentence hypotheses.

1.4.2 Parsers based on word transitions

With the goal of finding a parsing method that is computationally more efficient and able to handle word-hypothesization errors more gracefully than the caseframe approach, we developed and compared several stochastic trigram parsers that employ weak statistical models for representing syntactic and semantic knowledge [Stern et al. 87]. These parsers basically develop phrase and sentence hypotheses by considering the probability of a given word based on the preceding two words.

While most trigram representations are based on the transitional probabilities of words in the task domain, we constructed grammars based on ‘‘tags’’ that represent syntactic or semantic categories. We implemented two tag-trigram parsers, one that uses syntactic information and another that employs semantic knowledge, in addition to word scores and word-juncture information, for evaluating candidate words and phrases. The *syntactic trigram parser*’s tags convey information about a word’s syntactic attributes, such as noun, verb, singular or plural, and complete conjugations of frequently used auxiliary verbs. The tags of the *semantic trigram parser* are more domain-specific and correspond to the node labels used by our semantic network parser. Tag trigrams represent the probabilities of observing the syntactic or semantic classes of three consecutive words and estimate the joint probability that the syntactic or semantic structure of the word sequence is correct. This approach is especially effective in tasks with large vocabularies, since it reduces the amount of storage and training required to model word usage. In addition, this simple representation of syntactic and semantic knowledge can be compiled to permit highly efficient table lookup.

We implemented both tag-trigram parsers with a strategy that selects words from the input lattice in left-to-right fashion and adds them to the end of the most recently hypothesized word or phrase. To limit the number of possible word choices, we used a beam search that only retains candidate phrases within a predefined scoring range. Phrase and sentence scores are established by combining the measures of the candidate word's phonetic quality, the utterance's grammaticality, and the nature of junctions between hypothesized words.

To evaluate our word-transition parsers we compared their performance to that of our finite-state network parser and to an ''all-words'' parser. The latter uses only acoustic scores and word-juncture information and allows any word to legally be followed by any other word in the lexicon. This comparison showed that the two parsers using semantic knowledge perform significantly better than the all-words parser. Our semantic trigram and network parsers achieved 87% and 83% accuracy respectively on the original input lattice, significantly better performance than the all-words parser's 58%. On high-quality lattices the semantic parsers even achieved 98% and 92% accuracy, compared to the all-word parser's 59%.

Although the semantic trigram parser uses the same set of semantic concepts and relationships as the semantic network parser, it performed slightly better. This advantage is, most likely, due to the *scores* the trigram parser produces that indicate the relative likelihood of a candidate sentence being grammatically correct. The semantic network parser, on the other hand, does not provide quality ratings and merely accepts or rejects sentence hypotheses.

Our performance comparison showed that stronger semantic and syntactic constraints produce more accurate parsing even for lower-quality word lattices—as long as all correct words are present. When the quality of the word lattice is poor and, in addition, correct words are missing from the input lattice, *all* parsers hypothesize fewer correct words. The semantic-network parser degrades most rapidly because it has the tightest grammatical constraints.

1.4.3 Island-driven parsing

Low-quality word lattices, inevitable in speaker-independent, continuous speech recognition with large-vocabulary tasks, cause serious problems for the left-to-right parsing strategy. If a correct word is missing in the input lattice, the future portion of an otherwise correct partial parse will, most likely, be incorrectly extended. If the correct word is present but has a low score, left-to-right parsers are prone to pursue ‘‘garden paths’’ that are higher-scoring but incorrect, while pruning the lower-scoring, correct path.

Island-driven parsing promised to handle these problems more gracefully but was, at that time, neither widely used nor well-documented in the general research community. An island-driven search considers words with very high scores to be major content words and uses them as ‘‘seed words,’’ stable islands from which phrase hypotheses can be grown. Newly selected words are combined with those existing phrase hypotheses that have the highest score. To avoid generating a large number of connections between islands, a typical result in semantic *network* representations, we again chose the *trigram* approach that significantly reduces the search complexity [Ward et al. 88].

For a performance comparison between this island-driven trigram parser and the semantic trigram parser that employs the left-to-right strategy, we used test data consisting of 100 sentences from the Resource Management task and three types of word lattices that we prepared specifically for this study: original, blind-labeled, and missing-word lattices. The original lattices were generated by the word hypothesizer. The blind-labeled lattices were manually created by expert spectrogram readers and were better than the lattices generated by the word hypothesizer. We created the missing-word lattices by randomly deleting 10% and 20% of the correct words from the blind-labeled lattices. To improve the comparability of the two architectures in terms of the computational resources available to them, we widened the beam of the left-to-right parser considerably and disabled the island-driven parser’s capability to insert words in top-down fashion. The average run time per sentence was about 375 seconds for the left-to-right parser versus 330 for the island-driven parser. The number of word junctures examined per utterance approximated 610,000 for the left-to-right parser and 700,000 for the island-driven parser. At any given time, the left-to-right parser pursued an average of 1100 active phrase candidates, while the island-driven parser had about 490 phrase candidates in its queues.

Our results show that the island-driven parser performs more accurately on each data set. The performance of both parsers deteriorated on lower-quality word lattices that have no missing words but contain correct word candidates with poor scores and incorrect words with higher scores. However, the island-driven parser still hypothesized a greater number of correct words than the left-to-right parser, which failed completely when words were missing from the lattice. The main difference in the performance of the two parsers is the island-driven parser’s superior ability to handle input lattices with missing words.

A final improvement to our parsing techniques was the implementation of Bayesian word-score updating, an effective scheme for small amounts of training data. This technique groups words according to acoustic similarity and estimates the frequency with which each word was correctly hypothesized in the training data. Words are then rescored based on these *estimated* likelihoods. The Bayesian updating method reduced error rates by 50% for our left-to-right parsers with all-word and word-pair grammars. Including this scoring scheme in the island-driven parser produced an additional 30% reduction in parsing errors.

2. Support for large, heterogeneous applications

Large applications, among them AI systems for speech recognition and image understanding, typically are computationally demanding and include modules coded in several languages or tuned for specific architectures. Integrating and executing such large, heterogeneous systems in real time requires support that most general-purpose software and hardware cannot provide.

In this project, our goal was to develop, atop existing operating systems, software facilities that allow heterogeneous components to interact efficiently and execute on multiple architectures. We intended to make this support accessible to system builders through an intelligent programming environment that provides basic tools and facilitates designing custom tools for assembling, evaluating, and debugging complex systems. Furthermore, we endeavored to increase the performance speed of computationally intensive systems by building semicustom architectures that combine the advantages of general-purpose and fully specialized machines.

2.1 Software facilities

2.1.1 A prototype system

Our initial work focused on integrating Carnegie Mellon's Angel system whose front end and acoustic/phonetic analyzer were written in **C**, while the word hypothesizer and the sentence parser were coded in Lisp. These components ran on separate Perq and VAX workstations and communicated, via files, in a basically sequential manner. The operating system available on these platforms provided a simple message passing protocol.

We incorporated the client/server discipline in a central overseer process, the ''console.'' A separate, remote server on each machine was started manually. The console's first task was to start the distributed clients, the actual speech modules, via an RPC-based spawning/windowing mechanism we developed. These client processes would send messages containing their own communication ports for return messages.

The console, a VAX-resident Lisp process, read the system description from a configuration/control file. To create that file, we provided a language-oriented editor developed by the Gandalf project and a system-description language and libraries of our own design that shielded application programmers from the intricacies of message coding. For each message, the console automatically determined the appropriate port type, verified that the intended recipient existed, created a destination port, and generated code to connect sender and receiver.

By the end of 1985 we had implemented a prototype system [Bisiani et al. 86a] that ran the speech recognizer on a network of Perqs and MicroVAXes. In addition, our work reduced the time to partition, assemble, and run the distributed application from six man-months to about one.

2.1.2 Shared-memory interaction among heterogeneous components

While we had succeeded in integrating a large, heterogeneous application, our prototype support system had several shortcomings. The simple message-passing mechanism caused heavy message traffic and unacceptably slow transfer rates. The monolithic system structure proved too inflexible, requiring much time for coding the individual components and making modifications difficult. Even minor alterations in the system or one of its components required recompiling the entire configuration program. With about thirty researchers working on the project, changes were common, resulting in considerable system instability.

We learned that efficient performance in large, distributed systems depends on low-latency and high-bandwidth communication, flexibility, intermodule synchronization, and an intelligent programming environment. To provide these features, we shifted from transferring data across components to a machine-independent data representation and from a process- to a data-centered view. Mapping page-size entities to a shared memory avoids the high cost of data copying and protection. For implementing this approach, we needed a shared-memory multiprocessor with the appropriate operating system support, which we received at the end of 1985. It was a four-processor, shared-memory VAX that ran Carnegie Mellon's newly developed Mach operating system.

At that time, Mach's interprocess communication protocol provided a means for integrating **C** code in a strictly sequential Lisp program running on a multiprocessor. However, it did not support shared-memory communication among heterogeneous components executing in parallel. Our work focused on providing this missing support on top of the Mach operating system.

We developed a simple metalanguage for defining the structure of the data files that constitute the interfaces to Angel's heterogeneous components. These interfacing data structures are translated by our compiler that will be described in section 2.2.1. For each supported language, this compiler generates runtime type descriptors and appropriate access functions for the shared data structures. We provide standard library functions that allow the user or a software component to load type descriptor files directly into shared memory. Data structures can be created dynamically, with their type indicated at creation time. Modifying a structure constitutes an ''event,'' which is reported to all interested processes/agents. In this sense our *Agora environment* offers event-driven control. The only operating system support needed are simple locks and sleep/wakeup primitives, easily implemented with UNIX signals.

2.1.3 Communication among heterogeneous machines

We next turned to supporting communication among processes on distributed, heterogeneous machines and developed a ''network-shared-memory'' facility that creates the illusion of heterogeneous machines sharing one central memory. This mechanism is completely transparent to the user and realized via separate processes, the Agora servers, that communicate through Mach's network-communication primitives.

Shared data structures reside in shared memory local to their creating processes. Agora Library functions then replicate them on remote machines whose processes use the data. When a structure with remote copies changes, the server notifies the client hosts. This copy-on-reference scheme optimizes performance and allows allocating data structures on different machines without sacrificing global visibility. Agora's network communication facility demonstrates that memory-based communication for loosely-coupled architectures is not only feasible but more efficient than a message-based approach.

2.1.4 Availability

We enhanced availability of our Agora facilities by porting them to the Sun-3 workstation and the Encore Multimax 16-processor system. The Multimax was the first machine with enough processors to develop efficient parallel applications. We completed a working description of the Angel speech system within the Agora environment and ran it on the Multimax.

Work culminated in 1987 with the release of a stable and well-documented Agora system —now part of the standard Mach distribution— including the shared-data-type compiler and language interfaces for **C** and Common Lisp.

2.2 Programming support

We implemented Agora's support facilities [Forin et al. 87] as a collection of programs and libraries, with specific versions for each language and machine type. The programming system includes: interfaces for **C** and Lisp, the Agora compiler, a runtime library, and the Agora servers. We also constructed several basic tool examples, including a distributed debugger, a pattern matcher, a performance monitor, a parallel and incremental garbage collector, and a utility to save/restore runtime data structures to/from disk.

2.2.1 The Agora programming environment

Programmers and users can interact with the Agora facilities through two interfaces: the *Agora shell* —an extension of the UNIX **C** shell— or the *Lisp interpreter*. We elected to build on the **C** shell in order to provide a set of familiar and effective tools along with a means to combine and extend them. Both interfaces offer interpretive access to the Agora primitives and permit inspecting and monitoring the shared memory. Agora provides these capabilities without sacrificing any features of the **C** shell or Lisp interpreter. Our design thus supports easy transition into a parallel environment and facilitates developing parallel programming tools, such as a parallel ''make'' utility that can spread compilations over several machines.

The Agora *compiler* processes descriptions of shared data structures, expressed in a simple Lisp subset, along with any user-provided access functions. The descriptions provide the only specification of interactions among processes. Using the metalanguage specification, the compiler generates representations for each supported language as required. It also produces type descriptors that Agora's runtime system uses for type checking, transforming data between heterogeneous machines, and transferring structures between shared memory and disk. Instead of relying on a scheduler to activate application components, modifications to shared data structures are reported to interested processes through the compiler. The Agora compiler can even exploit the parallelism a particular machine offers, building complicated event hierarchies as appropriate.

Agora's *runtime library* offers, as procedural extensions for each supported language, standard functions for creating, reading, writing, and destroying shared data structures. These routines implement both the basic methods that Agora provides and user-defined access functions.

The Agora *server* translates typed data across heterogeneous machine boundaries at run time. Current alternative strategies for such communication typically require either compiling type descriptions along with the application code or packaging type information with each data message. In Agora, instead, each message bears a tag identifying the originating machine. Messages between identical or compatible architectures need not be transformed. Translation between incompatible hosts proceeds by swapping data-element bytes field by field, removing or inserting gaps, and shuffling bits for data alignment.

2.2.2 Basic tools

When we began, no existing debugger could deal with the complexity of large systems and with the specific problems of heterogeneous processors and programming languages. We addressed this lack by developing a *distributed debugger* that can handle both the parallel, distributed, and heterogeneous issues and provides a higher-level view of the application rather than of just a set of program counters. The Agora debugger is a parallel program comprising a core component and language-specific subprograms. Through the core, the developer controls the familiar **C** and Lisp debuggers, which we slightly modified to allow this control. The debugger uses a distributed database (See Section 2.2.3) that describes the user's computational models and their synchronization primitives. Moreover, the debugger can provide an exact replay of the sequence of events that led to an error, making it possible to capture bugs that manifest themselves only under rare conditions.

At run time, the user can employ the *pattern matcher* to describe, in a simple, interpreted language, a specific combination of events for activating a component, as opposed to letting the component filter and sequence modifications to the appropriate data structures.

Agora's *performance monitor* facilitates identifying bottlenecks by displaying, graphically and numerically, an application's performance parameters during execution. With this subsystem, a developer can start and stop the system, monitor time spent in the various processes, and debug the distributed system from a single machine.

The user can start Agora's *garbage collector* at any time to recover reusable space from the shared memory. Recovery proceeds incrementally and in parallel, without interfering with the running application. When an application runs on distributed machines, totally independent instances of the garbage collector run on each machine.

For *saving and/or restoring* runtime data structures on disk files, Agora uses a machine-independent file representation that allows heterogeneous machines to access all data files on central file servers. This direct file access provides a considerable advantage for applications that use large databases, since it avoids space- and time-consuming file duplication and conversion.

2.2.3 Custom system-building tools

To demonstrate how Agora can assist in building advanced programming environments and managing large AI systems, we constructed several tools that such systems might use. These sample tools include: a distributed database containing information about an application's components, a planner that helps the user define and execute complex environment tasks, and an interactive graphic user interface.

Agora's *distributed database* stores comprehensive medium- and coarse-grain information about the environment, that is, about the Agora programming support and the application system's components. The user describes application component properties and the most efficient procedures for interactions among them. These specifications are preserved in the database at compile time. We augmented the database with an object system that allows the user to attach code to components already described in the database. These dynamic changes are retained in the database at run time, a strategy that greatly enhances the debugger's replay functionality and prevents the user from having to resort to low-level mechanisms for debugging a distributed program. Source files, executables, and data reside in separate UNIX files. This compromise with a fully integrated approach eliminated the need to adapt existing tools and allowed us to concentrate on Agora's innovative aspects.

The *planner* aids the user in configuring and running a distributed application system. It creates generic scripts from the user's system description, goals, and constraints. These automatic scripts invoke the appropriate tools as well as all control and communication mechanisms required for execution, for example: initializing components, ensuring that each component reads the appropriate configuration files, and initiating compilation on a selected machine. The planner thus provides a means for easy, dynamic system modification. Components may be started and modified at any time. The scripts reconfigure the system and dynamically tailor available computing resources to the application's needs.

The *graphic user interface* displays icons of all objects and their relations in the database, showing name, class, and attribute/value. The interface allows the user to edit and modify existing object relations and provides great system building flexibility. Application builders can browse through the database and modify the system while evolving its components. Agora thus supports a comprehensive system description that precedes component development and obviates adding one later as an afterthought.

2.3 Custom accelerator architectures

Carnegie Mellon's own applications groups, specifically the speech recognition project, provided both a design target and an evaluation context for our research in designing hardware accelerators suitable for a wide range of large systems.

In speech recognition, as in many AI applications, search is a major consumer of computational resources. Devices adequate for front-end signal processing have been commercially available for several years, but optimal search hardware remains a moving target. This instability derives, partly, from the diversity among search technologies under investigation. Applications research continually extends old algorithms and introduces new ones. One requirement that all search algorithms have in common, however, is high performance.

The extreme search efficiency necessary for realtime recognition can only be achieved through highly parallel computing. High-performance, fine-grain parallelism requires memory access and synchronization at both high speed and high bandwidth. While some state-of-the-art hardware has the necessary raw speed, such machines are relatively expensive. Furthermore, beyond a certain level their performance is hampered by the lack of a memory bandwidth necessary for a highly-parallel search.

Specialized accelerators can resolve memory bandwidth limitations and synchronization bottlenecks but have their drawbacks, too. They usually exhibit long development times, do not scale easily to larger tasks, and become obsolete when the search algorithm changes.

Our objective was to find an effective compromise between general-purpose machines and fully customized accelerators. We focused on developing search-accelerator architectures that incorporate fast commercial processors, provide high-speed, high-bandwidth memory and synchronization operations, as well as a memory layout and synchronization mechanisms that can easily be optimized by developers.

This section presents our accelerator design for Angel's knowledge-based search. Chapter 5 describes our subsequent work on accelerator architectures.

Accelerating a knowledge-based search

As recognizers attempt to process increasingly larger vocabularies, the word hypothesizer carries the greatest burden. Hence, speeding Angel's word search promised the highest payoff for increasing the overall recognition speed. The word module used a ''best match'' beam search requiring, for a 240-word vocabulary, 20 to 40 MIPS per one second of speech. The variance of required processor performance depends on how much knowledge is used to constrain the search. Since computation grows approximately linearly with the vocabulary, realtime performance on a 5000-word lexicon would require 500 to 1000 MIPS per speech second. Even close-to-realtime performance for such a large vocabulary would be beyond reach on the largest general-purpose machines then available.

Our strategy for realtime recognition was to compile the time-consuming word search algorithm into custom hardware [Anantharaman and Bisiani 86] and employ a general-purpose processor for initialization and input/output.

We began by analyzing Angel's beam search to identify those sections of the sequential algorithm that can be parallelized. To generate the parallel code, we used the following set of transformations derived from vector-compilation techniques:

- Spreads—to schedule multiple instructions for parallel execution
- Pipelines—to execute successive processes with the same code
- Loops—to pipeline multiple instances of a nested loop
- Conditionals—to pipeline multiple instances of an `if` or `case` statement contained in a loop.

The parallelized search is mapped directly onto special-purpose architectures that we generated by decomposing a single custom RISC processor into sets of simpler processors. Communication is based on dedicated links and shared-memory modules. Dedicated links provide an efficient mechanism for transferring local variables generated by one stage and immediately consumed by another. Shared memory is critical for storing data effectively and for synchronizing data dependencies between stages. In addition, we developed a scheme to handle data dependencies that cannot be resolved at compile time and would require more than $O(n^2)$ general-purpose hardware elements.

To evaluate the effectiveness of our compilation technique, we simulated (in **C**$^{++}$) the single processor, a five-, and a 28-processor architecture and measured the performance of these versions with preprocessed speech input.

The single processor, running in parallel with the host computer, executed the search approximately 85× faster than a VAX-11/780 machine, largely because of our RISC design.

The five-processor architecture provided a low speedup compared to the 28-processor version, indicating that the transformation sequence we used was less optimal for the five-processor than for the 28-processor design.

The speedup for the 28-processor architecture was about 15× that achieved with the single processor. While this improvement may not appear impressive at first glance, note that each of the 28 processors is much smaller —in terms of hardware complexity— than the original single processor. Furthermore, the architecture's speedup in terms of transistor count is even super-linear. This can be explained by the fact that additional processors increase the number of instructions and data registers only very slowly.

We also compared these results to the speedup achievable with a multiprocessor version created by statistically dividing the search algorithm among a network of identical processors. While this alternate configuration provides better speedup in terms of processor count, it requires more hardware than our custom-hardware accelerator.

3. Statistically-based speech recognition

When we began this project, state-of-the-art recognizers performed impressively primarily because they imposed major constraints. Input was restricted to separately spoken words from a particular speaker on whom the system was trained and limited to a vocabulary of less than 500 words. Often these systems were designed for simple applications, such as recognizing letters of the alphabet or individual numbers.

Our team's goal was to build a recognizer without such limitations, a system that achieves high recognition accuracy while allowing continuously spoken input, various speakers, and large-vocabulary tasks.

Speech recognition, the process of identifying spoken words from the input signal, proceeds in roughly four stages:

- Designing acoustic models that represent speech units
- Training model parameters
- Processing the input signal and converting it into a sequence of information-bearing ''frames''
- Matching the frames to the acoustic models and determining the best match with the help of a language model and/or grammar.

Designing effective acoustic models is the central and most complex issue in speech recognition research. The most successful data-driven techniques for acoustic modeling are based on dynamic time-warping, hidden Markov modeling, or neural networks. When we embarked on this project, the hidden Markov approach was commonly preferred to dynamic time-warping algorithms since it produced better results for continuous speech with moderate-size vocabularies. Neural networks, on the other hand, were neither well-understood nor well-documented at that time.

3.1 The hidden Markov approach

The underlying assumption of the stochastic approach to speech recognition is that the signal can be effectively modeled as a random process and the process parameters estimated in a precise, well-defined manner from training data. In addition, hidden Markov models (HMMs), are particularly well suited for representing speech variabilities caused by different speakers, confusable sounds, contextual effects, and homophones.

The hidden Markov model consists of two embedded stochastic processes. One process models the signal's changing statistical properties as a sequences of states connected by transitions. These state sequences are ''hidden'' and can only be inferred from the observable symbols, which are probabilistic functions of the states and produced by the second process.

The two sets of parameters needed to specify an HMM are the transition and the output probability density functions. The transition distribution indicates the likelihood of taking a particular transition from a state and represents timescale variations of the signal. Output distributions can be based on *discrete* probability density functions or on *continuous* densities. In discrete HMMs, the output probability distribution describes the likelihood of emitting, at each state, a distinct symbol. Continuous output densities model the acoustic signal directly as vectors, typically, linear combinations of Gaussian functions.

3.2 Basic system operation

To establish a performance benchmark for an HMM-based recognizer trained and tested on the Resource Management task, we built the following basic system [Lee 88a].

Signal processing and data encoding

We sampled the incoming speech waveform at 16 kHz and blocked it into 20 ms frames that we overlapped by 10 ms and smoothed with a Hamming window. From these smoothed samples we computed, through autocorrelation, LPC coefficients and a set of 12 cepstral coefficients.

Modeling and training

We chose *discrete* HMMs for our benchmark system because they require less training and recognition time than continuous-density HMMs. A vector quantization process reduces the continuous data to discrete symbols from a finite alphabet. We generated this alphabet from a large sample of training data and it contains 256 discrete symbols (prototype vectors). With minimal signal distortion, these symbols represent the distribution of the training vectors and are stored in a single codebook. Mapping between continuous vectors and codebook indices becomes a simple ‘‘nearest-neighbor’’ computation that assigns the continuous vector to the codebook vector that is closest to it in spectral distance.

We modeled an initial set of 45 phones based on the set used in the Angel system for generating its pronunciation dictionary. The topology of each of our phone HMMs is shown in Figure 3-1. Transitions with the same label share the same output pdf.

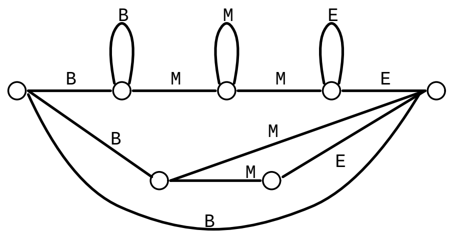

Figure 3-1: Phone HMM used in Sphinx

Training HMMs is an attempt to optimize the model parameters that describe a given state sequence observed from the training data, with the goal of creating the best model for a real event.

We initialized and trained the phone model parameters on the hand-segmented and -labeled TIMIT database with one iteration of the forward/backward algorithm. This algorithm has several desirable properties: It requires minimal supervision (simply an orthographic transcription of the training data), has a well-developed, mathematical basis that guarantees convergence, and scales gracefully to increased training.

By concatenating phones into words we created sentence models that we initialized with the trained phone models. The sentence models were then trained on actual input speech combined in the Resource Management training sentences (TIRM database). These trained models represent the expected pronunciation of particular input sentences.

Search and recognition

Hidden Markov models facilitate an integrated, goal-driven search that considers —at every decision— all available knowledge about acoustics, temporal distortion, and grammatical constraints. The HMM-based search thus has a considerable advantage over bottom-up approaches, which propagate errors and cannot integrate top-down knowledge, and top-down approaches, which are often intractable. In addition, the probabilistic nature of HMMs provides a consistent mechanism for scoring hypotheses.

An HMM search constitutes finding the optimal state sequence associated with the given observation. The difficulty lies with the definition of ''optimal.'' The most widely used approach is to use the Viterbi algorithm that finds —in a large hidden Markov model representing all sentences— the sentence that has the optimal state sequence. At each time frame all accessible states are updated and, at the end of the search, the path with the highest probability is recovered.

While the full Viterbi search is efficient for moderate tasks, it is extremely time-consuming for large ones. To reduce the search space we modified this algorithm into a beam search, a relatively easy change since the procedure is time-synchronous. A hypothesis is pruned when its log probability is less than that of the best hypothesis by more than a preset threshold. We found that it is possible to prune 80–90% of the hypotheses without any loss in accuracy.

Results

We evaluated our baseline Sphinx on the 991-word DARPA Resource Management task using an ''all-words'' grammar (where any word can follow any other) and obtained 26% word accuracy. When we incorporated a word-pair grammar with perplexity 60, accuracy increased to 58%. While by no means impressive, these results surpassed those of the Angel system and compared favorably to the performance of BBN's speaker-*dependent* BYBLOS recognizer that used a similar approach.

3.3 Extending and refining the basic system

Encouraged by our system's performance, we focused on extending and improving our approach in all stages of the recognition process. This work included creating more accurate models, modeling more versatile and sensitive speech units, integrating and improving training algorithms, and refining the search.

3.3.1 Incorporating knowledge in the modeling process

We can easily enhance simple stochastic modeling techniques by incorporating knowledge. This is accomplished by modifying the parameters computed for a fixed-size frame according to information based on the continuous signal. To enhance our HMMs, we incorporated in their parameter sets knowledge about human frequency perception and dynamic feature changes.

Encoding the speech signal with LPC cepstral coefficients results in linear frequency mapping. Human frequency perception, on the other hand, is better represented as a logarithmic function of frequency, and logarithmically scaled coefficients have shown to yield superior recognition accuracy compared to linearly scaled ones. To increase our models' discriminating ability, we used a bilinear transform on our LPC cepstral coefficients, a technique that is comparable to frequency warping of mel or Bark scales.

HMMs have no vision of the past or future: Their output events are assumed to depend on the current frame only. Specifically, discrete HMMs are limited since they produce a sequence of vector-quantized symbols representing *stationary* LPC coefficients computed for each fixed-size frame. To extend our models' scope beyond a single time slice, we incorporated information about interframe differences. Specifically, we added information about dynamic ''slope'' and power. Temporal power changes, represented as formant slopes, play an important role in auditory perception and are relatively invariant across speakers. Knowledge about power changes can help in such a simple yet important task as distinguishing speech from silence.

We computed a simple slope measure: differenced LPC cepstral coefficients. In addition to including power, we added differenced power to our parameter sets. Differenced power provides information about relative changes in amplitude or loudness and is computed in the same manner as differenced LPC cepstral coefficients.

Finding an effective method for integrating these additional parameters was somewhat of a challenge. Combining all parameters in a monolithic, multidimensional codebook would have produced poor matches between observed vectors and their corresponding prototypes and increased the number of quantization errors considerably. To reduce the parameter space and minimize VQ distortion, we used multiple codebooks. We generated 256 prototype vectors for each of our three parameter sets—12 LPC cepstral coefficients, 12 differenced cepstral coefficients, and a weighted combination of power and differenced power—and quantized each set in a separate codebook. With three codebooks, the discrete HMM algorithm produces three symbols at each time frame. To generate a joint output probability, we assume that the three individual output probabilities are independent and compute their product.

By adding knowledge-based parameters and integrating them in multiple codebooks, Sphinx's accuracy increased from 26% to 45% (null grammar), and from 58% to 84% (word-pair grammar) [Lee et al. 89a].

3.3.2 Versatile and sensitive speech units

Specialized phone models

An HMM can represent any unit of speech. However, to achieve good recognition it is crucial to model speech units that are ‘‘sensitive’’ and at the same time ‘‘general.’’ A speech unit is sensitive when it can capture contextual variations and general when it can be used to represent several similar speech events, thereby reducing the amount of data required for training.

The most natural speech unit to model is a complete word. Words are exactly what we want to recognize and they capture some contextual effects. However, separate word models share no information among each other. They are not generalizable and each word must be trained separately on extensive training data. Thus word models require memory space which grows linearly with the vocabulary size.

A better fundamental speech unit to model is the phone. Phone models are well understood and, unlike word models, highly trainable. Their main disadvantage is that they ignore coarticulation effects. These effects include different pronunciations, articulations, and stress of the same phone when it occurs in different contexts. They present a major problem for recognizing continuous speech. In some contexts, coarticulation may cause phones or words to be truncated or omitted. Function words (prepositions, conjunctions, pronouns, articles, and one-syllable verbs) are most noticeably affected by coarticulation. Since the English language has a limited number of function words and they occur frequently, we decided to separately model each phone in a set of 42 function words. Through this strategy we created 105 *function-word-dependent* phones.

Function words are even more difficult to recognize when they occur in phrases, such as `that are, of the, is the`. Such ‘‘function phrases’’ are seldom clearly articulated and have strong inter-word coarticulatory effects. We identified 12 such phrases, modified their pronunciations according to phonological rules, and created models for the phones they contain.

Modeling frequently occurring phones in function words and function phrases increased the number of models by a factor of five and further improved Sphinx' accuracy from 50% to 59% (all-words grammar) and from 84% to 88% (word-pair grammar).

Enhanced subword units

Since our speech articulators cannot move instantly from one position to another, we don't pronounce words as concatenated phones. Instead, the pronunciation of each phone is affected by its context. Thus, modeling phones without considering their context is an inadequate representation of continuous speech. A more sensitive unit is the *triphone*: It reflects a phone in its context, specifically, its immediate left and right neighbors. In addition, triphones are relatively task-independent, a particular advantage when the new vocabulary is not a subset of the old one.

While several studies show that modeling triphones reduces error rates by about 50%, it also produces many more models than when modeling context-independent phones. Additional models require more memory and training data, creating a considerable problem for large-vocabulary tasks.

To reduce the number of triphone models, we clustered similar triphones, those containing phones that have a similar effect on neighboring phones. Our agglomerative clustering procedure is equivalent to entropy clustering and employs a distance metric consistent with the maximum-likelihood criterion of the forward/backward algorithm. This approach to generalizing triphones enables us to determine empirically the largest amount of models that can be sufficiently trained, given a fixed amount of training data.

While the number of models increased fivefold when we modeled generalized triphones rather than simple phones, recognition accuracy improved from 59% to 79% (null grammar), from 88% to 94% (word-pair grammar).

Modeling between-word coarticulation

Our basic triphone models accounted for the left and right phonetic contexts *within* words but did not consider *between-word* coarticulation effects. Simply adding models for between-word coarticulation would have increased the total number of triphones for the 997-word task to 7057, too many to train reliably. By again clustering similar models, we reduced this number to a total of 1100 models representing in-word as well as between-word coarticulation.

Accounting for between-word coarticulation further improved Sphinx's recognition accuracy from 73% to 78% (null grammar), and from 94% to 95% (word-pair grammar).

Vocabulary-independent models

While modern recognizers provide considerable speaker independence, they still require additional training for each new task vocabulary. This process involves several weeks or even months of collecting data, days or weeks of generating the dictionary, and hours, sometimes days, of model training. Our longrange goal is to completely eliminate the need for vocabulary-specific training by generating ‘‘vocabulary-independent’’ models that are trained to maximal coverage on a large database of English phonetic variations.

Recognition on new tasks degrades because phonetic events in the testing vocabulary are not covered by the training data and the models are inconsistent across tasks. To make up for the lack of coverage, we can improvise with a more general model. However, if we create models that are too general, or inconsistent, they will generate different realizations of the same speech event.

After evaluating various subword units to find one that is generalizable yet consistent, we selected the triphone. In addition to reflecting a phone's immediate context, the triphone can easily represent other causes of variability, such as stress or a phone's position within a syllable or word. Since the number of triphone models in a given task is very large, they can only be sparsely trained. To increase trainability, we combined similar triphones according to a maximum-likelihood criterion.

The key to achieving vocabulary independence lies in generating models that can generalize to contexts not observed in the training data. To have this capability, we must train the generalized triphone models on a large database that covers most phonetic variations. We began collecting and processing a database of *general English pronunciations* and, as it grows, we will train more models and improve our system's coverage. We are also considering modeling additional acoustic/phonetic details, such as stress and syllable positions, and using units larger than triphones.

Training our models on 15,000 sentences provided over 99% coverage of the testing vocabulary's triphones. A preliminary evaluation of Sphinx with our vocabulary-independent models showed an encouraging 86.7% word accuracy, compared to 88.6% word accuracy with vocabulary-dependent models. When we interpolated vocabulary-*dependent* and vocabulary-*independent* models, word accuracy increased further, to 90.7% [Hon and Lee 90].

3.3.3 Incorporating and improving training

Semicontinuous hidden Markov models

The disadvantage of discrete HMMs is that the vector quantization process uses a simple distance metric unrelated to the MLE criterion, yielding the closest codeword from the codebook for each acoustic observation. This mapping from the continuous acoustic space to the discrete codeword space causes ‘‘quantization errors.’’ Continuous HMMs, on the other hand, use estimated continuous probability density functions to model acoustic observations directly—without vector quantization. While continuous HMMs eliminate quantization errors, they increase computational complexity and the number of parameters that can be reliably estimated.

To combine the advantages of discrete and continuous HMMs and integrate training in the modeling process, we introduced *semicontinuous* HMMs. They allow modeling a rich mixture of probability density functions, while requiring fewer training data and lower computational complexity than continuous HMMs. With appropriately chosen acoustic parameters and probability density functions, semicontinuous HMMs greatly enhance robustness.

Our semicontinuous HMMs combine discrete (model-dependent) output probabilities and the top four or six continuous (model-independent) probability density functions. This strategy provides mutual optimization of VQ vectors and HMM parameters under a unified, probabilistic framework. Implementing semicontinuous HMMs in Sphinx reduced the error rate by 13% compared to the system with discrete HMMs [Huang et al. 90a].

Corrective and reinforcement training

The forward/backward algorithm automatically refines the hidden Markov model by adjusting its parameters to increase the probability that it generated the training data. This type of training is based on the assumption that the underlying model is correct, while HMMs are poor models of natural speech. Thus, the forward/backward procedure increases the overall score of a path and not the number of correctly recognized sentences.

To maximize our models' recognition rate on the training data, we added a corrective and reinforcement algorithm to the training process. This procedure derives from a corrective-training procedure for speaker-*dependent*, isolated-word recognition which we extended to speaker-*independent*, continuous speech. Our algorithm has two components: *error-correction training*, to improve correct phrases and suppress misrecognized ones, and *reinforcement training*, to improve correct sentences and suppress near misses.

In isolated-word recognition, it is relatively simple to substitute misrecognized words and reinforce near misses. In continuous speech, on the other hand, word boundaries are not easily identified. Therefore, we must insert, delete, or substitute entire phrases, a process that requires extensive training data and an algorithm to generate a list of phrase substitutions for near misses.

To better exploit our limited training data, we developed a *cross-validation* procedure that uses the training database more effectively and produces realistic examples of misrecognized sentences for each training sentence. A dynamic programming algorithm then matches each of these misrecognized sentences against the corresponding correct one and generates an ordered list of likely *phrase* substitutions. Finally, we hypothesize likely near-miss *sentences* by randomly replacing phrases in the correct sentence with corresponding confusable phrases from our list.

To modify the HMM parameters, we preserve the counts from the dynamic alignment: An HMM's count is increased if it contributed to a correct alignment, subtracted if it contributed to an incorrect one. This increases the likelihood of correct sentences and decreases the probability of confusable ones.

Our corrective training procedure reduced Sphinx' error rate by 20% over the standard, MLE-trained system, improving recognition accuracy from 95.5% to 96.2% (word-pair grammar) [Lee and Mahajan 89].

Speaker-adaptive training

The Sphinx recognizer is designed to be speaker-independent and does not require retraining for new speakers. However, as the vocabulary increases, the number of pronunciation variabilities and speech characteristics will grow correspondingly and cause recognition accuracy to degrade. Thus, an effective approach to improving recognition accuracy would consist in adapting the model parameters gradually and nonintrusively to fit new speakers.

We evaluated two speaker-adaptation algorithms and had success with interpolating speaker-dependent parameters derived from a small number of adaptation sentences. This algorithm assumes a fixed codebook but adapts the HMM parameters to individual speakers. Speaker-dependent parameters are derived by running the forward-backward algorithm. These parameters, which are appropriate but poorly trained, are then combined with the better trained but less appropriate speaker-independent parameters. In addition, we derive from the adaptation sentences parameters for tied-phone states, co-occurrence smoothing, and similar speakers, and combine all parameters using a deleted interpolation procedure.

With only 30 adaptation sentences, this algorithm reduced speaker-independent Sphinx' error rate by about 5–10%.

3.3.4 Duration modeling

Since HMMs do not enforce any global constraints, a 50-state word HMM may have reasonable durations at each state, while the total word duration may be quite unreasonable. To improve recognition, we decided to incorporate in the Viterbi search higher-level knowledge about word durations. We modeled this knowledge by a univariate Gaussian distribution, with mean and variance estimated from a supervised Viterbi segmentation of the training set. Since possible duration scores for all words are precomputed, this technique incurs essentially no runtime overhead. While duration modeling did not increase accuracy in our system when no grammar was used, it resulted in a considerable improvement (45–50%) with a grammar.

4. Speech understanding systems for problem-solving tasks

The ability of modern systems, such as Sphinx, to recognize speech with high accuracy offers new possibilities for myriad applications. We can now strive for ''natural'' man/machine communication and contemplate a new generation of systems that not only recognize words and sentences but are able to ''understand'' their meaning.

The major challenge facing us in developing these new systems is the recognition and understanding of spontaneous speech. We addressed this issue in four projects with the following major goals:

- Building efficient, user-friendly interfaces for voice-driven problem-solving systems
- Developing an effective parsing strategy for processing spontaneous and nongrammatical speech
- Exploiting natural task constraints and the flow of user/computer dialogs to reduce perplexity and increase the system's understanding
- Making voice-driven systems robust to changes in their acoustical environments.

4.1 Interfaces for voice-driven systems

Interacting verbally with computers in problem-solving tasks seems to offer several advantages. Speech input is natural and easy to use since it doesn't require learning a new skill, as keyboard input does. It is also more efficient since it frees the user's hands for other tasks, a considerable asset, for example, in piloting aircraft.

Our longrange research goal is to build efficient, user-friendly interfaces for speech-driven, problem-solving applications and to develop programming environments that facilitate rapid prototyping of such interfaces. Under this contract, our work focused on identifying behavior and requirements of people interacting with voice-driven, problem-solving applications. We also analyzed system and task characteristics that must be considered in making voice-driven interfaces effective and efficient.

4.1.1 Two prototypes

To understand how voice-driven systems would perform that are built on Sphinx recognition technology, we designed two simple systems, one for a programmable calculator and a multiprocess ''office manager'' system whose application comprises an appointment calendar, a telephone directory, and an electronic mail facility [Rudnicky et al. 88b].

The main system components are the recognizer, the voice interface, and the application. The user's utterance is processed by the free-running recognizer and the stream of recognized words sent to the interface. The interface then places the recognized utterance into an edit buffer and, after approval or correction by the user, routes it to the application's parser. The result of the parse finally produces an application-specific system action.

Since Sphinx's models were trained on the Resource Management task and we intended to use the recognizer in the voice-calculator and office-manager domains, we wanted to assess the degree of model overlap among domains and how it effects performance. Our analysis revealed that different tasks share approximately 60–70% of their models. Further, we found that the recognizer achieved only 74% word accuracy on the voice-calculator task (a 130-word vocabulary). After additional training on a corpus of 2000 utterances from the calculator domain, the recognizer's word accuracy rose to 89%. Since task-specific training significantly improves recognition accuracy, it will have an important impact on the overall performance of voice-driven systems. Unfortunately, training the recognizer for each new task is time-consuming and requires collecting large amounts of additional data. However, task-specific training will not be necessary in the future when ''vocabulary-independent modeling'' (see Section 3.3.2) has been perfected.

As we developed our prototype systems' language models, we found that the domain language has two major components: task-specific expressions (such as `two plus two` in the calculator domain) and a ''core'' language, which comprises three types of linguistic units:

- Control words and expressions that provide access to system-level functions, such as `Standby`
- Terms that facilitate switching between applications, for example, `Select` (e.g. calendar)
- Generic terms, such as numbers.

This decomposition makes the language-building activity more manageable and introduces into the design the concept of reusability.

4.1.2 A voice-driven spreadsheet

For our subsequent studies we developed a voice-driven spreadsheet system [Rudnicky 89a]. The spreadsheet task has several advantages over the tasks we used in our pilot systems. It encompasses a wider range of user activities, from simply entering data to solving complex problems. Furthermore, many people are familiar with spreadsheet applications, quite a few are even experts. The ready availability of experienced users eliminates training and allows us to concentrate on research issues.

Designing the spreadsheet language

The first step in building our new system's language component was to collect words and expressions from a variety of users, novices to experts, completing certain spreadsheet tasks. To avoid the limitations an automatic speech recognizer would impose, we used a ''wizard'' design. This technique employs a hidden operator who translates user expressions into appropriate computer commands, while the users' utterances are simultaneously recorded.

Transcriptions of our wizard-system data provided the basis for the second stage in our language design: establishing a habitable language for first-time users. We analyzed the data, incorporated the most common syntactic expressions and words in the task language, and formalized the language description in a BNF grammar. With this grammar we generated a large corpus of sentences that, combined with the previously collected calculator sentences, provided the training data for our spreadsheet system.

System architecture

Our voice-driven spreadsheet system, shown in Figure 4-1, has four components. The mode control module allows users to gate the input. The preparser analyzes core sub-vocabularies, collapses synonyms, determines whether the spoken input is syntactically correct, and notifies the user if it is unacceptable. The main parser maps spoken commands to the symbolic form required by the application. The edit buffer allows the user to correct a recognized string from the keyboard.

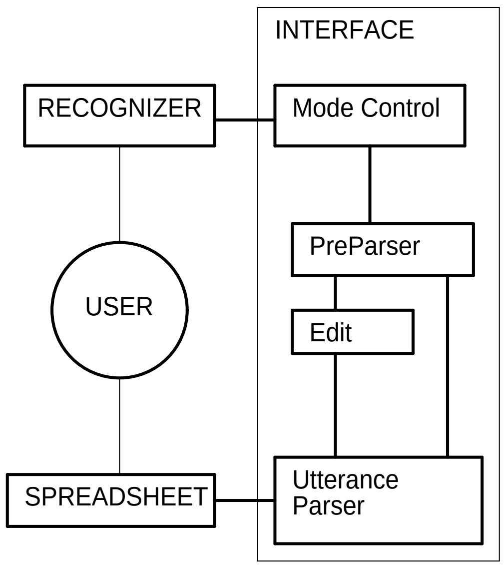

Figure 4-1: Architecture of the voice spreadsheet system

We implemented this system using a Sun 3/60 as host, the Beam search accelerator (See section 5.1), and custom-built digitizing and signal-processing hardware.

Two interaction studies

We designed our first study to compare the utility of three different input modes: keyboard, voice, and voice with keyboard for confirmation/correction [Rudnicky et al. 89]. We asked subjects to use three versions of our system for entering a set of financial data into a worksheet. The baseline version responded to keyboard input only. For spoken input we configured the system to listen continuously and to act immediately on the recognized string. The version for voice and keyboard input displayed the recognized utterance and required confirmation or correction before acting.

Participants completed the task most rapidly in keyboard mode (12.7 minutes), least rapidly by voice alone (19.3 minutes). We attributed most of the time deficit for voice input to the recognizer's high word-error rate on this task, amounting to 25%. The intermediate task-completion time for voice-with-keyboard input (15.2 minutes) indicates that voice entry is enhanced by providing the more reliable keyboard mode to deal with recognition failures. We also collected data on the number of commands used in each mode and found that participants used fewer commands when entering data by voice (134) than by keyboard (187).

The goal of our second study was to identify those factors that reduce user and system performance when speech input is used. We wrote scripts for a total of 40 spreadsheet tasks and asked eight participants to complete the first 30 in one session, using voice and keyboard input alternately for successive tasks. To assess any changes in performance over time, we asked our participants one month later to complete the remaining ten tasks in the same manner.

Figure 4-2 shows the completion times for the first 30 tasks: Keyboard input was faster than voice input and improved for both over time. Not shown are completion times for the last 10 tasks, carried out a month later. These tasks were initially completed in less time than the last few tasks of the previous session and, as in the previous session, performance improved after several tasks. The relationship between keyboard and voice input was the same in both sessions.

To understand why voice input was consistently slower, we measured how long users took to complete subtasks, such as moving between cells and entering data, and excluded the system's response time. We found that these activities were accomplished more rapidly by voice than by keyboard. This suggests that the poor completion time for voice was not due to an intrinsic shortcoming of this input mode but can be attributed to slow recognition and sluggish system response. Factoring out these two delays revealed a better total task-completion time in voice mode, as shown by the dashed curves in Figure 4-2.

4.1.3 Evaluating voice-driven systems

Traditionally we measure a recognizer's performance by its word accuracy. However, when comparing voice-driven systems, we are less interested in how well a system recognizes individual words and more in whether it understands and correctly interprets user utterances to produce appropriate actions. Using the data from our second spreadsheet study, we compared four approaches to evaluating voice-driven systems. (See Figure 4-3)

The most conservative approach, which we call *exact-transcription accuracy* (Metric

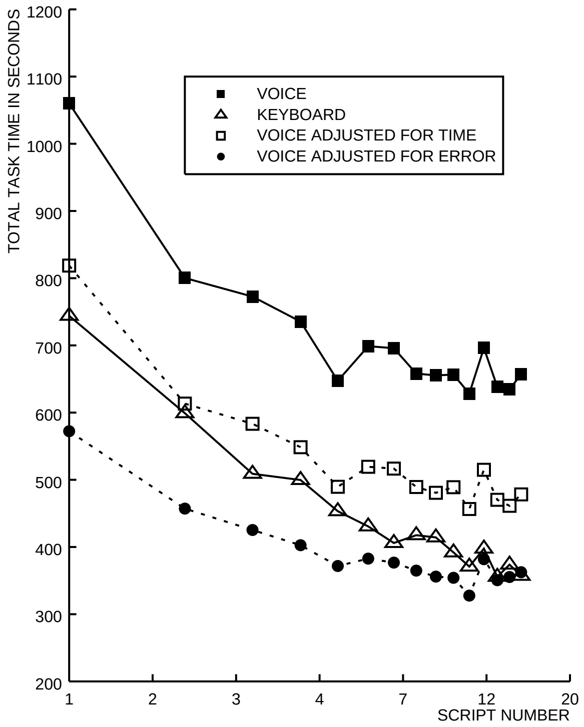

Figure 4-2: Total task completion times for the second spreadsheet study, with adjustments for system performance

1) is to match the system's output to a transcription of the utterance. While this metric indicates the system's ability to identify words and deal with extraneous, non-lexical events, it does not reflect how often the user's intentions were understood.

Semantic accuracy (Metric 2) indicates whether the system produced the desired result, regardless of the actual match between recognized string and the transcription of the expected input, for example, `hundred fifty` and `one hundred and fifty`. By registering transaction successes, this measure reflects the system's actual performance. The minimal spread between the exact-transcription and semantic-accuracy curves in Figure 4-3 indicates that our system did not incorporate higher-level knowledge for interpreting user intentions. Sophisticated systems that have this capability should display a substantial difference between these two metrics.

Extraneous-event accuracy (Metric 3) measures the number of nongrammatical or extraneous events in an utterance and indicates how well users conform to system-imposed language constraints. The decreasing number of nongrammatical utterances toward the end of this curve shows that users learn to control their input language.

Grammatical accuracy (Metric 4) reflects the system's ''habitability'' and ''coverage,'' the range of task-related options and expressions it allows. The corresponding curve in

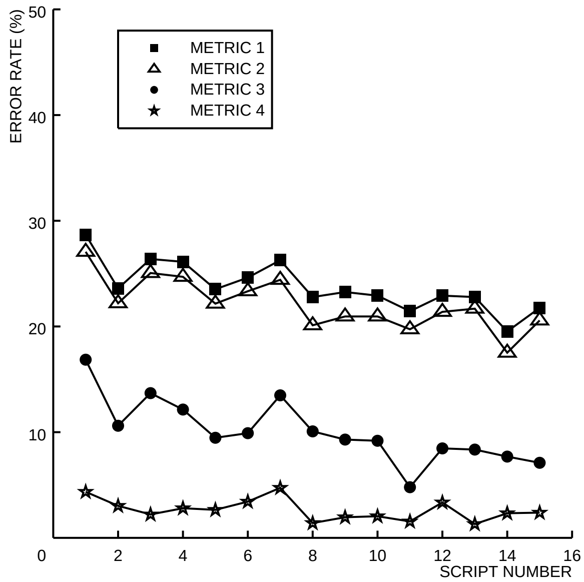

Figure 4-3: Different characterizations of recognizer performance in the second spreadsheet study

Figure 4-3 indicates that our spreadsheet grammar covered approximately 97% of the utterances participants used in the first 30 tasks. This rather high percentage probably reflects the *inherent* constraints imposed by the simple tasks. In more complex tasks, expressions that lie outside the system's grammar will increase in number, resulting in higher error rates [Rudnicky and Sakamoto 89].

4.1.4 Design requirements for voice-driven systems

Our practical experience highlighted three features that are critical for the effectiveness and success of voice-driven systems. These features are interaction control, an error-correction facility, and realtime system response. Our preliminary explorations of these features produced the following insights [Rudnicky and Hauptmann 89a].

Interaction control

People take turns in conversations by using mutually understood verbal or facial expressions. Likewise, interacting with a computer from the keyboard or with voice input requires protocols that define the rules of the two-way communication. The effectiveness and facility of a protocol for keyboard interaction hinges on the keyboard's dependability, for voice-driven interaction on the recognizer's accuracy. When accuracy is low, the user must repeat the utterance until it is correctly recognized. If the system performs unintended actions on misrecognized input, the user may have to employ rather complex procedures to undo them. To avoid the high cost of correction and retries, an interaction protocol with explicit confirmation is required: The system waits for a specific confirmation word or keyboard entry before proceeding. Such confirmation step can include a facility for correcting a misrecognized string. When the cost of explicit confirmation exceeds that of occasional error recovery—as it will in a sufficiently accurate recognizer—the fixed confirmation step can be eliminated. In place of confirmation, the interaction protocol could then include a ‘‘disconfirmation’’ option to intercept incorrect actions, for instance, with a keyword.

Error correction

A voice-driven system’s usability depends, to a large degree, on the efficiency of its error-correction facility: Making corrections must not unduly interrupt the interaction flow, in other words, it must not be time-consuming nor require a significant shift of attention. The facility can be either intramodal (using voice input) or multimodal (using keyboard and/or pointer input). Data from our first spreadsheet study indicate that, given the current state of the recognition technology, keyboard correction is more efficient than an intramodal strategy. To confirm this we conducted an additional study, in which we compared three input modes for entering a list of simple digit strings in a spreadsheet: voice (including a specific confirmation word); voice with keyboard for corrections and a carriage return for confirmation; and keyboard entry, correction, and confirmation. The results show that keyboard correction increases task throughput: The overall task-completion time for voice with keyboard correction averaged 9.6 seconds versus 11.1 seconds in voice mode. Furthermore, bimodal input was faster than voice-only input for making corrections and confirmations.

Realtime system response

One of our experiments showed how sensitive users are to small differences in system-response characteristics. We asked users to enter, from the keyboard, data in specific spreadsheet cells and compared the times they required for completing this simple task in three versions of our system: the baseline system, which essentially responds instantaneously; a version with a fixed 820 ms delay after each input; and a third system with a random delay averaging 703 ms.

Both the fixed and random delays encouraged users to pack more information into each command. However, rather than speeding the total amount of time spent in the interaction, these complex commands degraded system performance and required numerous corrections. We suspect, slow system response will have a similar effect on user interactions with voice-driven systems.

4.2 Parsing spontaneous speech

Standard recognizers perform well when their input consists of sentences read from prepared scripts. However, their accuracy deteriorates critically when they have to process unrehearsed, spontaneous speech. While sentences read from scripts are complete and grammatically well-formed, unrehearsed, spontaneous speech typically includes sentence fragments and nongrammatical word strings, pauses, ellipses, interjections, repeats, and restarts that often even introduce a new concept. In addition, read speech is usually pronounced carefully, while naturally-spoken language is frequently truncated, contains slurred or mispronounced words, new words that are not in the lexicon, or syntax that is not in the grammar. Our goal was first to identify the phenomena that distinguish spontaneous speech and then to develop a system that can parse and understand naturally spoken input.

Characteristics of spontaneous speech

To collect examples of spontaneously-spoken speech, we recorded utterances of users interacting with a simulated spreadsheet system in a typical office environment. When we analyzed the transcriptions, we discerned two major types of nonspeech phenomena: ‘‘noise events’’ generated either by the user or the environment. User-generated noise often fills speech pauses and includes sounds such as ‘‘uhm,’’ ‘‘ah,’’ coughs, and paper rustling. Examples of environmental noise events are door slams and phone rings. In addition to nonspeech events, we identified various structural peculiarities characteristic of spontaneous speech. These include restarts with repeats of complete or partial words or phrases; interjections; ellipses or sentence fragments that can only be understood in a context; and nongrammatical constructions, caused by unusual word orderings or subject/verb agreement errors. Such irregularities violate the constraints that standard recognizers employ to increase performance. Both non-speech phenomena and ungrammatical speech structures lead to recognition errors that can cause total parsing failure.

A spontaneous-speech parsing strategy

Once we understood the characteristics of spontaneous speech, we set out to build a prototype system [Ward 89a] that can parse naturally spoken input and carry out appropriate actions. The main challenge we faced was to reduce the large search space generated by spontaneous speech without unduly restricting natural user/system interaction. To satisfy these conflicting goals, we developed a ‘‘flexible’’ parser and created models for common noise events.

Our system processes the Sphinx recognizer’s output, a ‘‘single, best hypothesis’’ for each utterance. Rather than analyzing each recognized word string for complete sentences with well-defined boundaries, our flexible parser generates *phrases* and assigns them, according to their *function* and *meaning*, to slots in frame-like structures. Each slot in a particular frame represents a concept and the slots’ combined information will produce a specific system action or database-system response. While frames enforce global constraints for combining phrases, local constraints control how slots are filled by guiding the word search and the grouping of words into phrases. To reduce the search space, each slot is represented as a separate finite-state network, specifically a recursive transition network. The parser matches these networks against the input word string to identify the function or meaning of an isolated phrase and can also ‘‘call’’ other nets if a short phrase is embedded in a longer phrase.

The parser fills slots with phrase candidates as it processes the input utterances from left to right. To accept unusual orderings of the recognized utterance's constituents, slots within a frame can be filled in any order. Each partially filled frame is regarded as a possible utterance interpretation. The parser attempts to extend all interpretations by assigning new candidate phrases to empty slots. When the parser finds a candidate phrase for an already filled slot, it produces an alternate interpretation. The system pursues several possible interpretations simultaneously and scores each frame according to its filled slots and the number of words contained in the input utterance. After the entire utterance has been parsed, the highest-scoring frame is selected as output and its associated function executed or a query built. The query is then sent to a database management system and the result displayed.

System evaluation

The system we evaluated at a DARPA demonstration incorporated a lexicon of 484 words from the Airline Travel Information Service (ATIS) task, a word-pair grammar with perplexity 85, and our flexible parser. It did not yet include noise-event models.

We processed both transcript and recorded data comprising 93 utterances from five speakers. An utterance was considered ''correctly parsed'' when the system produced the appropriate database output. Our prototype produced the wrong response to 52% of the transcript data and to 61% of the speech data. This result is encouraging, since the recognizer's string-error rate on this task was as high as 79%. Yet a significant number of utterances was judged ''incorrect'' due to misunderstandings about the type of output that should be generated. After modifying the output generation, we re-evaluated our prototype and found that wrong responses dropped to 35% on transcript data and 58% on spoken input.

In a separate test we demonstrated that noise-event models significantly improve the performance of HMM-based recognizers on noisy input [Ward 89b]. We built these models by first analyzing spontaneous utterances recorded in a spreadsheet task experiment. From these data we extracted a set of the most common noise events. We determined a context-independent phone for each of these events and trained new acoustic hidden Markov models that recognize noise events in the same manner as HMMs recognize actual words.

4.3 Dialog modeling

Recognizing continuous speech is for a computer significantly more difficult than recognizing isolated words. Increasingly larger vocabularies and spontaneous, ''natural'' speech complicate this challenge further. Recognition problems introduced by continuous speech result from its properties: blurred word boundaries, pronunciations that vary with word contexts and individual speakers, and poor articulation of function words, such as articles and prepositions. When the vocabulary size exceeds 1000 words, the number of confusable words increases rapidly and the search becomes considerably more complex. Spontaneously spoken input, characterized by incomplete sentences, repetitions, restarts, pauses, and noises, additionally increases the search space's size and complexity and further degrades recognition performance.

The most common approach to controlling the search space has been to impose the syntactic and/or semantic constraints of grammars. Typically, these constraints are represented in the form of probabilistic or semantic networks that remain constant, while different utterances are parsed. These grammars are effective when the input to the recognizer is syntactically and semantically accurate, as in read speech, and the average constraint at each decision point, measured as perplexity, is small (30 to 60). Under such favorable conditions, recognizers have achieved word accuracy rates in excess of 95%. However, as search space and complexity increase, static, low-perplexity grammars become inadequate.

One central component of our new method for controlling large and complex search spaces is pragmatic, ‘‘dialog-level’’ knowledge. We derived our pragmatic knowledge constraints from research in natural-language processing, which demonstrated that communication forms, such as dialog and discourse, are structured into goals and subgoals. Knowledge of the speaker’s goals and the dialog domain, or context, helps us understand implied and unexpected information. In addition, pragmatic knowledge about problem-solving communication enables us to anticipate what the speaker will say next.

As we began our project, pragmatic knowledge was used at only two research sites for processing the recognizer’s output, specifically for correcting errors, disambiguating the recognized utterance, or performing inferences required for understanding. We, on the other hand, decided to insert dialog knowledge early in the recognition process and to modify the grammar dynamically for different utterances. By reducing the search space at the word level we intended to limit recognition errors that are typically caused by ambiguity and confusion among words.

To demonstrate the effectiveness of our approach, we built an interactive dialog system called MINDS. In addition to spoken input, MINDS allows keyboard and pointer input for selecting menus, making corrections, and providing clarification. Our system has four knowledge databases:

- A hierarchically organized, tree-structured knowledge base of possible abstract goals and plans a user may have in a dialog or problem-solving task
- A frame-based network database of specific domain knowledge
- A collection of general, domain-independent knowledge, such as user focus and anaphoric references, represented as procedures and rules
- A model of an individual user’s domain knowledge.

To construct the domain-independent dialog-knowledge database we collected transcripts of human/computer dialogs and analyzed them for goals, plans, and concepts. We then prepared scenario scripts that describe all possible subdialog sequences. Based on these scripts we built a goal tree that defines goals, subgoals, and domain concepts and, in addition, all available traversal options.

MINDS essentially processes the input in a loop. Based on the current state of the dialog, our system anticipates the scope of plausible utterances. These ‘‘predictions’’ are sets of concepts that we represent in a goal tree. We expand predictions into a semantic grammar and restrict the active lexicon so that it contains only words, phrases, and sentences of possible, prospective user actions. The restricted grammar and lexicon are then applied as the next utterance is parsed. Finally, the system displays the recognized utterance to the user for verification and generates an appropriate command/query to the database application.

We tested our initial system on input from an experimental version of the Angel front end. The test sets consisted of three dialogs with a 205-word vocabulary. By restricting vocabulary and grammar according to system-generated predictions, MINDS effectively reduced the number of candidates in the recognition process. We quantified this search space reduction in terms of *average perplexity* of the dynamically generated grammars that MINDS uses to parse incoming utterances. On the average, our system reduced perplexity by a factor of three, as compared to a system that uses only static grammatical constraints.

We found two shortcomings in our pilot system. First, all predictions were based on built-in assumptions, a strategy that caused the system to fail when the assumptions were not met by what the user subsequently said. In addition, we observed that larger task domains increased the number of possible goals and plans and weakened the effect of dialog-based constraints.

To provide recovery from failed predictions, we modified our system so that it generates ‘‘layered’’ predictions for each utterance [Young et al. 89a], [Young et al. 89b]. For the most specific predictions all possible constraints are used. The constraints for the most general predictions are those provided by the task grammar. When the recognizer fails to parse an utterance at a particular constraint level, the next layer of more general predictions is invoked. The system then reparses the utterance and progressively relaxes constraints until the parse corresponds to a legal phrase or sentence described by the dynamically recompiled grammar. To prevent generating multiple active goals and plans, we designed an algorithm that selects a ‘‘most probable goal state.’’ This state describes goals that are complete, most likely to follow the previously activated goal, and consistent with goals at higher levels.

For a preliminary evaluation of our improved system, we used the independent TONE (task-oriented, naturally elicited) speech test set. Results show that MINDS reduced the test-set grammar’s perplexity by a factor of 15. In addition, we tested our system on a database query task. This task consists of problem-solving scenarios based on transcripts of naval personnel solving problems caused by disabled vessels (a domain similar to the Resource Managements task). In this evaluation, MINDS correctly formulated 100% of the database queries.

The final enhancement to our system under this contract was to implement it on distributed communication facilities, providing a separate processor for each knowledge source. This improvement decreased MINDS' response time and made testing and debugging easier.

5. Accelerating stochastically-based searches

While general-purpose machines provide the necessary computational power for processing the speech signal, they are insufficient for handling the recognizer's word search on the 1000-word Resource Management task in real time. On a Sun-4/260 and a 15-processor Encore Multimax this search requires about 7× and 5× real time, respectively. Our work on accelerator architectures for Sphinx continues research introduced in Section 2.3.

Sphinx uses a Viterbi search that applies heuristic rules to discard unpromising alternatives. The remaining ''beam'' of nearly optimal interpretations is examined in parallel and each of its states updated every 10ms, the duration of one speech frame. This beam search reduces the number of states that need updating at every frame from roughly 40,000 to about 4000. Still, the beam's size can vary dramatically and unpredictably from frame to frame, making the search extremely memory-demanding.

Developing a computationally-efficient, multiprocessor-based search engine required fine-grain algorithm decomposition and extensive synchronization. We also had to solve the load-balancing problem caused by the changing beam size. The primary challenge, however, lay in overcoming the memory-bandwidth bottleneck caused by an average data flow of 40 Mbytes/s to execute the 1000-word Resource Management task in real time. Not only was the bandwidth of the general-purpose multiprocessors (typically 10 Mbytes/s) quickly exhausted, but a careful analysis showed that even additional processors would not eliminate the bottleneck, since about 50% of the processing time was consumed by synchronization operations.

5.1 Beam search accelerator

Our efforts in resolving these challenges resulted in the ''Beam Machine'' [Bisiani et al. 89a], a customized architecture of general-purpose hardware that is more powerful than the sum of its components. We connected three Weitek 8032 processors through optimized software that enhances their combined performance. To increase memory bandwidth, we added two banks of shared memory to the local storage. Each bank connects to its host through a separate VME bus, thus two processors can access shared data simultaneously. We augmented the shared memory with one-bit-per-word flags, which are managed directly by the hardware and visible at the instruction level. The mutual exclusion provided by these flags effectively reduces synchronization overhead and prevents load imbalances.

Three people built the Beam accelerator in about five months, which demonstrates that incorporating general-purpose technology in a custom architecture is many times faster and cheaper than developing general-purpose or fully-customized systems. The Sphinx recognizer executed five times faster on the Beam architecture than on the 15-processor Encore Multimax. At a demonstration in June, 1988, Sphinx achieved 1.1× real time on the 997-word Resource Management task with a perplexity 20 grammar and 1.38× real time with a perplexity 60 grammar. Since that time the speech group has used the Beam accelerator almost daily.

5.2 A programmable accelerator

Despite its effectiveness, the Beam architecture remains limited by its bus bandwidth, the major drawback of all bus-based systems. This drawback becomes particularly significant and causes poor performance on fast processors, large numbers of processors, or algorithms that have a poor cache-hit ratio. Furthermore, Beam is not scalable to more demanding searches, such as those for processing spontaneously spoken language. Finally, the complexity of Beam's host interface discourages application developers from optimizing the search implementation. In late 1988, we embarked on designing a scalable multiprocessor accelerator that can maintain reasonable memory performance with a large number of fast processors [Bisiani and Ravishankar 90a]. To achieve this performance we decided to provide a physically distributed, global memory that is shared logically by each of the general-purpose processor nodes and connected through a scalable communication mechanism. Our Plus architecture has no bus but supports, at the instruction set level, a shared-memory programming model on a message-passing machine.

The main difficulty we had to overcome in implementing this distributed-shared-memory system was the communication latency caused by remote-memory-access operations. While caching reduces latency considerably, it is traditionally performed on demand and requires a substantial number of fast and expensive processors. To avoid this cost, Plus uses software-controlled, *nondemand* caching of data among multiple memories. However, the speedup gained through caching does not solve the increase in latency caused by synchronization operations. We minimized this bottleneck by providing special hardware-supported synchronization mechanisms for programming the Plus architecture.

Nondemand caching

Each node's local memory is used both as main memory and as cache for data that reside in another node's memory. Data in a processor's local memory may also reside in its cache. A processor's local memory and cache are kept coherent with a local snooping protocol, while local memories of different nodes are updated automatically by Plus's hardware-based coherence manager.

In a bus-based system, write operations can immediately be made visible to all processors either by announcing them on the bus (with a write-through protocol) or by announcing them when necessary (with copy-back protocols). These sequentially-consistent and strongly-ordered systems let programmers use ordinary read/write memory operations to implement synchronization operations. In architectures with multiple physical memory units that are not connected by a global communication medium, enforcing strong ordering requires time-consuming protocols. Fortunately, weak ordering of one processor's actions, which means that changes are not immediately made visible to all other processors, is usually sufficient.

We implemented Plus as a weak-ordering memory system, leaving it to the programmer to enforce strong ordering of shared-data access when concurrent computations require it. In Plus, data may be replicated and stored in more than one node's local memory at the request of software. These *replicated data* (to differentiate them from *cached data*) are usually pages whose size is dictated by the memory management of the general-purpose CPU. To avoid having write operations block a processor, Plus provides dedicated hardware that write-updates replicated pages. This allows several write operations, issued by a single processor, to proceed concurrently and even complete out of order.

Hardware-supported synchronization mechanisms

Plus provides three hardware-supported mechanisms that allow the programmer to improve synchronization efficiency.

The first of these mechanisms, an *explicit fence* operation, supports implementing, when necessary, strong ordering of synchronization operations. This operation causes the coherence manager to block any subsequent writes issued by the processor until all earlier ones have completed. Secondly, Plus separates the initiation of a read, modify, or write operation from the production of its result by issuing an identifier the program can use to retrieve the operation's result. Plus allows these *delayed operations* to overlap regular processing and several to be in progress at the same time on one processor, thereby considerably reducing the average synchronization latency of these operations. Finally, Plus improves synchronization efficiency by supporting *complex* synchronization operations that allow implementing many common synchronization data structures, such as semaphores, queues, barriers, and high-contention locks. While complex operations are not easy to use, they are more efficient for distributed memory systems than the currently favored and simpler RISC schemes that require repeated network transits between an invoking process and a memory location.

Programming environment

Plus hardware uses a multithreaded, shared-memory model and, to simplify programming, completely supports the same multiple-thread-of-control abstractions that the Mach operating system uses for concurrent execution. A Plus program is identical to a Mach task with multiple threads and can be executed and debugged on any Mach host.

Implementation

We have implemented Plus on a general-purpose Motorola 88000 processor (25 MHz) with 32 Kbytes of cache and 8 or 32 Mbytes of main memory at each node. The memory is organized in two interleaved banks to sustain the burst bandwidth needed for cache accesses. An interface connects memory and processors to a communication network that uses mesh routers designed at Caltech. Each Plus node can be connected to host computers and audio peripherals through an SCSI-to-network adapter.

This one-node Plus prototype has been running since November 1989. For a preliminary evaluation, we ported Sphinx's signal processing program to a single Plus node and achieved approximately 0.7× realtime performance.

6. Speaker adaptation and environmental robustness

In this project, we aimed at enhancing automatic speech recognition in two respects: by making recognizers robust to noise, reverberation, and other changes in the acoustical ambience, and by improving recognition accuracy of systems that are designed to be speaker independent but degrade as vocabulary size and number of speakers increase.

The algorithms we implemented to provide environmental robustness operate on the raw speech waveform and are readily adaptable to knowledge- and statistically based technologies. On the other hand, the speaker-adaptation techniques we developed for Angel are embedded in the acoustic/phonetic classification procedure and thus much more system dependent. We could not transfer these algorithms to Sphinx and had to change our speaker adaptation approach. Toward the end of this contract, we began developing a parameter-estimation procedure that is heavily influenced by the adaptive-filtering scheme.

6.1 Speaker adaptation for a knowledge-based system

Although many modern recognizers are designed to be speaker independent, their tolerance to acoustic variability decreases with large task vocabularies and spontaneously spoken input. Acoustic variability is caused by the differences in individual speakers' vocal tracts, dynamic range, pitch, and speech characteristics. Dynamically adapting the recognizer to the acoustical characteristics of new speakers remains an important means of improving recognition performance.

To adapt Angel to new speakers, we implemented a dynamic parameter-adaptation algorithm derived from the method used in its predecessor, Feature, which used the same acoustic recognition approach. This procedure had decreased Feature's error rate from 11% to 6.5%, and we expected it would be effective for Angel as well. The main goal of dynamic parameter adaptation is to modify, during the recognition process, a new speaker's feature vectors with respect to the statistically optimal feature-vector combination used in training the system. Since this procedure was designed to improve recognition of isolated English letters, numerous extensions were required to implement it in Angel. The main difficulty was to include knowledge about contextual effects caused by continuous speech.

Contrary to expectation, this algorithm improved the acoustic/phonetic subsystem's accuracy only marginally. Manual analyses of feature vectors in misrecognized utterances revealed that Angel's acoustical features showed greater variability in different phonetic contexts than they did from speaker to speaker, and this contextual variability hampered speaker adaptation.

6.2 Environmental robustness

Recognition accuracy degrades dramatically when recognizers are trained and evaluated in different acoustical environments. Our group's objective is to develop techniques for improving the recognizer's robustness with respect to background noise and acoustical distortions. Specifically, we have endeavored to raise recognition performance in office environments with desktop microphones to the level achieved in noise-controlled settings with close-talking microphones.

Acoustical distortions that degrade speech-recognition performance can be clustered in two complementary classes:

- Additive background noise caused by fans or other office machinery, by people talking, slamming doors, or rustling paper.
- Collective linear filtering effects caused by such acoustic systems as microphones, speakers' vocal tracts, and typical rooms.

Various methods to compensate for acoustical distortions are available. During this contract we concentrated on enhancing the speech signal before it enters the recognizer. Two conventional techniques are used in acoustical preprocessing: *spectral subtraction* to eliminate additive noise and *spectral normalization* to compensate for linear filtering effects.

6.2.1 Robustness for the knowledge-based Angel recognizer

The initial approach we used in the Angel system to compensate for both types of acoustical distortions was to enhance the speech signal before it enters the recognizer.

The spectral subtraction procedure estimates the amount of background noise present during nonspeech intervals and subtracts the noise spectrum from that of the incoming signal. This approach proved quite effective for Angel's acoustic/phonetic component. It enabled extracting better features, which resulted in more accurate phoneme hypotheses. We also developed a fixed-point-arithmetic version of this algorithm that is more efficient and degrades accuracy only slightly.

Spectral normalization —to compensate for linear-filtering effects— first estimates a static compensation vector. This ‘‘linear filter’’ is based on the average sound spectrum (when speech is present) of a given system's training data. The filter is then applied to speech and nonspeech frames of the incoming signal. While this simple scheme provided some compensation for linear filtering effects and improved the acoustic/phonetic scores, it was less effective than spectral subtraction.

We also explored the feasibility of improving Angel's robustness in office environments through an adaptive noise-canceling method that employs two microphones. One microphone provides a reference channel assumed to contain only noise components. The adaptive filtering procedure subtracts the reference microphone's output from that of the other microphone, leaving only the speech signal. Evaluations showed that, contrary to expectation, this noise-cancellation technique produced virtually no improvement. Further analysis indicated that most conventional adaptive filters are not designed to cope with reverberation encountered in typical rooms.

6.2.2 Robustness for a stochastic recognizer

Sphinx' high accuracy makes it possible to evaluate the effectiveness of environmental-robustness algorithms in the context of the entire system's performance, while the Angel system allowed us to evaluate our enhancement methods only on the output produced by the acoustic/phonetic subsystem. An additional advantage of the stochastic approach is that it uses the cepstral domain, rather than the raw spectrum, for characterizing the signal's frequency content. The cepstral domain requires fewer parameters than the spectral domain, which simplifies our enhancement procedures.

Specific experiments showed that, in a continuous-speech, alphanumeric task, Sphinx' recognition accuracy decreased from 85.3% to 18.6% when a closetalking microphone was used for training and an omnidirectional desktop microphone for testing.

To enhance Sphinx' robustness in the presence of background noise and linear filtering effects, we first implemented the classical spectral normalization and subtraction algorithms independently, using the cepstral domain as the basis for computations [Acero and Stern 90a]. With spectral normalization Sphinx' word accuracy increased to 38.3% on the same alphanumeric task and under the same conditions that previously caused performance to drop to 18.6%. With spectral subtraction Sphinx' word accuracy rose to as much as 64.8%.

To combine the advantages of spectral subtraction and normalization we developed two novel algorithms that additively correct the incoming feature vectors in the cepstral domain [Acero and Stern 90b]. Of these, simple *SNR-dependent cepstral normalization* (SDCN) raises accuracy from 18.6% under the above described conditions to 67.2%. One drawback of this algorithm is that it requires microphone-specific training. Our *codeword-dependent cepstral normalization* (CDCN) algorithm is computationally more intensive but achieves 74.9% accuracy and doesn't require specific training for new environments. This improvement due to CDCN enhancement is significant since it approaches Sphinx' 76.5% accuracy without the normalization algorithm, when the recognizer is both trained and tested on the same omnidirectional microphone. With the CDCN algorithm we are able to compensate almost completely for the accuracy lost when the system is trained on one microphone and tested on another.

I. Bibliography

[Acero 90] Acero, Alejandro.
Acoustical and environmental robustness in automatic speech recognition.
PhD thesis, Carnegie Mellon University, Department of Electrical and Computer Engineering, September, 1990.

[Acero and Stern 90a]
Acero, A. and R.M. Stern.
Towards environment-independent speech recognition.
In *Proceedings of the DARPA Speech and Natural Language Workshop*. DARPA, June, 1990.

[Acero and Stern 90b]
Acero, A. and R.M. Stern.
Environmental robustness in automatic speech recognition.
In *Proceedings of the ICASSP*. IEEE, April, 1990.

[Adams and Bisiani 86]
Adams, D. and R. Bisiani.
The Carnegie Mellon University distributed speech recognition system.
Speech Technology Mar/Apr(2):14-23, 1986.

[Alleva and Lee 89]
Alleva, F. and K.F. Lee.
Automatic new word acquisition: spelling from acoustics.
In *Proceedings of the DARPA Speech and Natural Language Workshop*. DARPA, October, 1989.

[Alleva and Thayer 88]
Alleva, F.A., and E.H. Thayer.
The Carnegie Mellon portable speech library.
In *Proceedings of the 116th Meeting of the Acoustical Society of America*. Acoustical Society of America, November, 1988.

[Anantharaman 87]
Anantharaman, T.S.
Algorithms for parallel execution of programs with memory aliasing.
Technical Report CMU-CS-87-109, Carnegie Mellon University, Computer Science Department, March, 1987.

[Anantharaman 89]
Anantharaman, T.S.
A delay insensitive regular expression recognizer.
Technical Report CMU-CS-89-109, Carnegie Mellon University, School of Computer Science, January, 1989.

[Anantharaman and Bisiani 86]
Anantharaman, T.S. and R. Bisiani.
A hardware accelerator for speech recognition algorithms.
In *Proceedings of the 13th International Symposium on Computer Architecture*. ACM and IEEE, June, 1986.

[Anantharaman and Bisiani 87]
Anantharaman, T.S. and R. Bisiani.
Computer architectures for speech recognition.
Scientific Computation Series. The Characteristics of Parallel Algorithms.
The MIT Press, 1987.

[Anikst et al. 90] Anikst, M.T., W.S. Meisel, M.C. Soares, and K.F. Lee.
Experiments with tree-structured MMI encoders on the RM task.
In *Proceedings of the DARPA Speech and Natural Language Workshop*. DARPA, June, 1990.

[Bisiani 85] Bisiani, R.
Agora, an environment for building problem solvers on distributed computer systems.
In *Proceedings of the 1985 Distributed Artificial Intelligence Workshop*. December, 1985.

[Bisiani 87a] Bisiani, R.
Beam search.
Encyclopedia of Artificial Intelligence 1:56-58, 1987.

[Bisiani 87b] Bisiani, R.
Some considerations on computer architectures for artificial intelligence.
Fundamentals in Computer Understanding: Speech and Vision.
In Haton, J.P.,
Cambridge University Press, 1987.

[Bisiani 90] Bisiani, R.
Computer system architecture and hardware technology for speech processing.
In *Proceedings of VERBA*. January, 1990.

[Bisiani and Forin 87]
Bisiani, R. and A. Forin.
Multilanguage parallel programming.
In *1987 International Conference on Parallel Processing*. IEEE, August, 1987.

[Bisiani and Forin 88]
Bisiani, R. and A. Forin.
Architectural support for multilanguage parallel programming on heterogeneous systems.
IEEE Transactions on Computers 37(8):930-946, 1988.
Also in Proceedings of the Second International Conference on Architectural Support for Programming Languages and Operating Systems, IEEE, October 1987.

[Bisiani and Ravishankar 90a]
Bisiani, R. and M. Ravishankar.
PLUS: a distributed shared-memory system.
In *Proceedings of the 17th International Symposium on Computer Architecture*. IEEE, May, 1990.

[Bisiani and Ravishankar 90b]
Bisiani, R. and M. Ravishankar.
Programming the PLUS distributed memory system.
In *Proceedings of the Distributed Memory Computing Conference*. IEEE, April, 1990.

[Bisiani et al. 86a] Bisiani, R., F. Alleva, A. Forin, and R. Lerner.
A software and hardware environment for developing AI applications on parallel processors.
In *Proceedings of the 4th National Conference on AI*. American Association of Artificial Intelligence, August, 1986.

[Bisiani et al. 86b] Bisiani, R., F. Alleva, A. Forin, and R. Lerner.
Agora: Closing the gap between general-purpose systems and applications.
In *Proceeding of AAAI-86*, pages 742-747. American Association of Artificial Intelligence, August, 1986.

[Bisiani et al. 87a] Bisiani, R., F. Alleva, F. Correrini, A. Forin, F. Lecouat, and R. Lerner.
Heterogeneous parallel processing: The **Agora** *shared memory*.
Technical Report CMU-CS-87-112, Carnegie Mellon University, Computer Science Department, March, 1987.

[Bisiani et al. 87b] Bisiani, R., F. Alleva, F. Correrini, A. Forin, F. Lecouat, and R. Lerner.
Heterogeneous parallel processing: The **Agora** *programming environment*.
Technical Report CMU-CS-87-113, Carnegie Mellon University, Computer Science Department, March, 1987.

[Bisiani et al. 88a] Bisiani, R., F. Lecouat, and V. Ambriola.
A tool to coordinate tools.
IEEE Software 5(6):17-26, 1988.

[Bisiani et al. 88b] Bisiani, R., A. Nowatzyk, and M. Ravishankar.
Coherent shared memory on a message passing machine.
Technical Report CMU-CS-88-204, Carnegie Mellon University, School of Computer Science, December, 1988.

[Bisiani et al. 89a] Bisiani, R., T.S. Anantharaman, and L. Butcher.
BEAM: An accelerator for speech recognition.
In *Proceedings of the ICASSP*. IEEE, May, 1989.
Also available as technical report CMU-CS-89-102.

[Bisiani et al. 89b] Bisiani, R., A. Nowatzyk, and M. Ravishankar .
Coherent shared memory on a distributed memory machine.
In *18th International Conference on Parallel Processing*. IEEE, August, 1989.

[Chigier 88] Chigier, B. & R.A. Brennan.
Broad class network generation using a combination of rules and statistics for speaker-independent, continuous speech.
In *Proceedings of the ICASSP*. IEEE, April, 1988.

[Cole 86] Cole, R.A.
Phonetic classification in new generation speech recognition systems.
In *Proceedings of Speech Tech 1986*, pages 43-46. Speech Technology , April, 1986.

[Forin et al. 87] Forin, A, R. Bisiani, and F. Correrini.
Parallel processing with Agora.
Technical Report CMU-CS-87-183, Carnegie Mellon University, Computer Science Department, December, 1987.

[Franzini et al. 89a]
Franzini, M., M. Witbrock, and K.F. Lee.
A connectionist approach to continuous speech recognition.
In *Proceedings of the ICASSP*. IEEE, May, 1989.

[Franzini et al. 89b]
Franzini, M.A., K.F. Lee, and A.H. Waibel.
Combining hidden Markov models and connectionist networks for continuous speech recognition.
In *Proceedings of the Neural Information Processing Society*. Neural Information Processing Society, November, 1989.

[Franzini et al. 89c]
Franzini, M.A., M.J. Witbrock, and K.F. Lee.
Speaker-independent recognition of connected utterances using recurrent and non-recurrent neural networks.
In *Proceedings of the International Joint Conference on Neural Networks*. June, 1989.

[Franzini et al. 89d]
Franzini, M.A., M.J. Witbrock, and K.F. Lee.
Connected digit recognition using recurrent neural networks.
In *Proceedings of Speech Tech '89*. Speech Technology, May, 1989.

[Franzini et al. 90] Franzini, M.A., K.F. Lee, A.H. Waibel.
Connectionist Viterbi training: a new hybrid method for continuous speech recognition.
In *Proceedings of the ICASSP*. IEEE, April, 1990.

[Hauptmann and Rudnicky 90]
Hauptmann, A.G. and A.I. Rudnicky.
A comparison of speech and typed input.
In *Proceedings of the DARPA Speech and Natural Language Workshop*. DARPA, June, 1990.

[Hauptmann et al. 88]
Hauptmann, A.G. and A.I. Rudnicky.
Talking to computers: An empirical investigation.
International Journal Man-Machine Studies (28):583-604, 1988.
Also available as technical report CMU-CS-87-186.

[Hauptmann, Young, and Ward 88]
Hauptmann, A.G., S.R. Young, and W.H. Ward.
Using dialog level knowledge sources to improve speech recognition.
In *Proceedings AAAI-87, Seventh National Conference on Artificial Intelligence*. AAAI, August, 1988.

[Hayamizu et al. 90]
Hayamizu,S., K.F. Lee, and H.W. Hon.
Description of acoustic variations by hidden Markov models with tree structure.
Technical Report CMU-CS-90-116, Carnegie Mellon University, School of Computer Science, March, 1990.

[Hayes et al. 86] Hayes, P.J., A.G. Hauptmann, J.G. Carbonell, and M. Tomita.
Parsing spoken language: A semantic caseframe approach.
In *Proceedings of Coling '86*. International Conference on Computational Linguistics, August, 1986.

[Hon and Lee 90] Hon, H.W. and K.F. Lee.
On vocabulary-independent speech modeling.
In *Proceedings of the ICASSP*. IEEE, April, 1990.
Appeared under the title *Towards speech recognition without vocabulary-specific training* in Proceedings of the DARPA Speech and Natural Language Workshop, October 1989.

[Huang et al. 89a] Huang, X.D., H.W. Hon, and K.F. Lee.
Multiple codebook semi-continuous hidden Markov models for speaker-independent continuous speech recognition.
Technical Report CMU-CS-89-136, Carnegie Mellon University, School of Computer Science, April, 1989.

[Huang et al. 89b] Huang, X.D., H.W. Hon, and K.F. Lee.
Large-vocabulary speaker-independent continuous speech recognition with semi-continuous hidden Markov models.
In *Proceedings of the DARPA Speech and Natural Language Workshop*. DARPA, October, 1989.

[Huang et al. 89c] Huang, X.D., H.W. Hon, and K.F. Lee.
Speaker-independent continuous speech recognition with continuous and semi-continuous hidden Markov models.
In *Proceedings of Eurospeech*. September, 1989.

[Huang et al. 90a] Huang, X., F. Alleva, S. Hayamizu, H.W. Hon, M.Y. Hwang, and K.F. Lee.
Improved hidden Markov modeling for speaker-independent continuous speech recognition.
In *Proceedings of the DARPA Speech and Natural Language Workshop*. DARPA, June, 1990.

[Huang et al. 90b] Huang, X.D., K.F. Lee, and H.W. Hon.
On semi-continuous hidden Markov modeling.
In *Proceedings of the ICASSP*. IEEE, April, 1990.

[Hwang et al. 89a] Hwang, M.Y., H.W. Hon, and K.F. Lee.
Between-word coarticulation modeling for continuous speech recognition.
In *Proceedings of Eurospeech*, pages 5-8. , September, 1989.
Also available as technical report CMU-CS-89-141.

[Hwang et al. 89b] Hwang, M.Y., H.W. Hon, and K.F. Lee.
Interword coarticulation modeling for continuous speech recognition.
In *Proceedings of the 117th Meeting of the Acoustical Society of America*. Acoustical Society of America, May, 1989.

[Lee 85] Lee, K.F.
Incremental Network Generation in Template-Based Word Recognition.
Technical Report CMU-CS-85-181, Carnegie Mellon University, Computer Science Department, December, 1985.

[Lee 88a] Lee, K.F.
Large-vocabulary speaker-independent continuous speech recognition: the Sphinx system.
PhD thesis, Carnegie Mellon University, Computer Science Department, May, 1988.
Also available as technical report CMU-CS-88-148.

[Lee 88b] Lee, K.F.
On the use of triphone models for continuous speech recognition.
In *Proceedings of the 116th Meeting of the Acoustical Society of America*. Acoustical Society of America, November, 1988.

[Lee 88c] Lee, K.F.
Speech recognition research at Carnegie Mellon.
In *Proceedings of the Second Symposium on Advanced Man-Machine Interface through Spoken Language*. November, 1988.

[Lee 88d] Lee, K.F.
On large-vocabulary speaker-independent continuous speech recognition.
Speech Communication, Journal of the European Association for Signal Processing 7:375 - 379, 1988.

[Lee 89a] Lee, K.F.
Hidden Markov models: past, present, and future.
In *Proceedings of Eurospeech*. September, 1989.

[Lee 89b] Lee, K.F.
Automatic Speech Recognition: the Development of the Sphinx System.
Kluwer Academic Publishers, Boston, 1989.

[Lee 90] Lee, K.F.
Context-dependent phonetic hidden Markov models for speaker-independent continuous speech recognition.
IEEE Transactions on Acoustics, Speech, and Signal Processing 38(4):599 - 609, 1990.

[Lee and Alleva 90]
Lee, K.F. and F.A. Alleva.
Continuous speech recognition.
Recent Progress in Speech Signal Processing.
In Furui, S. and M. Sondhi,
Marcel Dekker, Inc., 1990.

[Lee and Hon 89] Lee, K.F. and H.W. Hon.
Speaker-independent phone recognition using hidden Markov models.
IEEE Transactions on Acoustics, Speech, and Signal Processing 37(11):1641-1648, 1989.
Also published in *Proceedings of the 116th Meeting of the Acoustical Society of America*, November, 1988, and available as technical report CMU-CS-88-121.

[Lee and Mahajan 89]
Lee, K.F. and S. Mahajan.
Corrective and reinforcement learning for speaker-independent continuous speech recognition.
In *Proceedings of Eurospeech*. September, 1989.
Also available as technical report CMU-CS-89-100.

[Lee at al. 90] Lee, K.F., H.W. Hon, and R. Reddy.
An overview of the Sphinx speech recognition system.
IEEE Transactions on Acoustics, Speech and Signal Processing 38(1):35 - 45, 1990.

[Lee et al. 89a] Lee, K.F., H.W. Hon, M.Y. Hwang, S. Mahajan, and R. Reddy.
The Sphinx speech recognition system.
In *Proceedings of the ICASSP*. IEEE, May, 1989.

[Lee et al. 89b] Lee, K.F., H.W. Hon, and M.Y. Hwang.
Recent progress in the Sphinx speech recognition system.
In *Proceedings of the DARPA Speech and Natural Language Workshop*. DARPA, February, 1989.

[Lee et al. 89c] Lee, K.F., H.W. Hon, and M.Y. Hwang.
Recent advances in large-vocabulary speaker-independent continuous speech recognition.
In *Proceedings of Speech Tech '89*. Speech Technology, May, 1989.

[Lee et al. 90a] Lee, K.F., S. Hayamizu, H.W. Hon, C. Huang, J. Swartz, and R. Weide.
Allophone clustering for continuous speech recognition.
In *Proceedings of the ICASSP*. IEEE, April, 1990.

[Lee et al. 90b] Lee, K.F., H.W. Hon, M.Y. Hwang, and S. Mahajan.
Recent progress and future outlook of the Sphinx speech recognition system.
Computer Speech and Language (4):57 - 69, 1990.

[Lunati and Rudnicky 90]
Lunati, J.M. and A.I. Rudnicky.
The design of a spoken language interface.
In *Proceedings of the DARPA Speech and Natural Language Workshop*. DARPA, June, 1990.

[Nakagawa 86a] Nakagawa, S.
Speaker-Independent phoneme recognition in continuous speech by a statistical method and a stochastic dynamic time warping method.
Technical Report CMU-CS-86-102, Carnegie Mellon University, Computer Science Department, January, 1986.

[Nakagawa 86b] Nakagawa, S.
Speaker-Independent sentence recognition by phoneme-based word spotting and time-synchronous context-free parsing.
Technical Report CMU-CS-86-109, Carnegie Mellon University, Computer Science Department, February, 1986.

[Polifroni and Rudnicky 89]
J. H. Polifroni and A. I. Rudnicky.
Modeling lexical stress in read and spontaneous speech.
Journal of the Acoustical Society of America 86:S77(A), 1989.

[Rudnicky 89a] Rudnicky, A.I.
The design of voice-driven interfaces.
In *Proceedings of the DARPA Speech and Natural Language Workshop*. DARPA, February, 1989.

[Rudnicky 89b] Rudnicky, A.I.
Goal-directed speech in a spoken language system.
Journal of the Acoustical Society of America 86:S76(A), 1989.

[Rudnicky 90] Rudnicky, A. I.
The design of spoken language interfaces.
Technical Report CMU-CS-90-118, Carnegie Mellon University, School of Computer Science, March, 1990.

[Rudnicky and Hauptmann 89a]
Rudnicky, A.I. and A.G. Hauptmann.
Conversational interaction with speech systems.
Technical Report CMU-CS-89-203, Carnegie Mellon University, School of Computer Science, December, 1989.

[Rudnicky and Hauptmann 89b]
Rudnicky, A.I. and A.G. Hauptmann.
Errors, repetition, and contrastive emphasis in speech recognition.
In *Proceedings of the AAAI Symposium on Spoken Language Systems*. AAAI, March, 1989.

[Rudnicky and Li 88]
Rudnicky, A.I. and Z. Li.
Prosodic information in lexical access.
In *Proceedings of the DARPA Speech Workshop*. DARPA, June, 1988.

[Rudnicky and Sakamoto 89]
Rudnicky, A.I. and Sakamoto, M.H.
Transcription conventions and evaluation techniques for spoken language system research.
Technical Report CMU-CS-89-194, Carnegie Mellon University, School of Computer Science, November, 1989.

[Rudnicky and Stern 89]
Rudnicky, A.I. and R.M. Stern.
Spoken language research at Carnegie Mellon.
Speech Technology 4(4):38-43, 1989.

[Rudnicky et al. 87]
Rudnicky, A.I., L.K. Baumeister, K.H. DeGraaf, and Eric Lehmann.
The lexical access component of the CMU continuous speech recognition system.
In *Proceedings of the ICASSP*. IEEE, April, 1987.

[Rudnicky et al. 88a]
Rudnicky, A., Z. Li, J. Polifroni, E. Thayer, and J. Gale.
An unanchored matching algorithm for lexical access.
In *Proceedings of the ICASSP*. IEEE, 1988.

[Rudnicky et al. 88b]
Rudnicky, A.I., R.A. Brennan, and J.H. Polifroni.
Interactive problem solving with speech.
In *Proceedings of the 116th Meeting of the Acoustical Society of America*. Acoustical Society of America, November, 1988.

[Rudnicky et al. 89]
Rudnicky, A.I., M. Sakamoto, and J.H. Polifroni.
Evaluating spoken language interaction.
In *Proceedings of the DARPA Speech and Natural Language Workshop*. DARPA, October, 1989.

[Rudnicky et al. 90a]
Rudnicky, A.I., M.H. Sakamoto, and J.H. Polifroni.
Spoken language interaction in a goal-directed task.
Proceedings of the ICASSP :45-48, April, 1990.

[Rudnicky et al. 90b]
A.I. Rudnicky, M.H. Sakamoto, and J.H. Polifroni.
Spoken language interaction in a spreadsheet task.
Human-Computer Interaction—INTERACT ’90.
In D. Diaper,
, 1990.

[Shikano 85] Shikano, K.
Text-independent speaker recognition experiments using codebooks in vector quantization.
Technical Report CMU-CS-86-101, Carnegie Mellon University, Computer Science Department, April, 1985.

[Shikano 86a] Shikano, K.
Evaluation of LPC spectral matching measures for phonetic unit recognition.
Technical Report CMU-CS-86-108, Carnegie Mellon University, Computer Science Department, February, 1986.

[Shikano 86b] Shikano, K., K.F. Lee, R. Reddy.
Speaker adaptation through vector quantization.
Technical Report CMU-CS-86-160, Carnegie Mellon University, Computer Science Department, December, 1986.

[Shikano 86c] Shikano, K.
Text-independent speaker recognition experiments using codebooks in vector quantization.
Technical Report CMU-CS-86-101, Carnegie Mellon University, Computer Science Department, , 1986.

[Stern and Acero 89] Stern, R.M. and A. Acero.
Acoustical pre-processing for automatic speech recognition.
In *Proceedings of the DARPA Speech and Natural Language Workshop*. DARPA, October, 1989.

[Stern and Lasry 87] Stern, R.M. and M.J. Lasry.
Dynamic speaker adaptation for feature-based isolated word recognition.
IEEE Transactions on Acoustics, Speech, and Signal Processing 35(6):751-763, 1987.

[Stern et al. 87] Stern, R.M., W.H. Ward, A.G. Hauptmann, and J. Leon.
Sentence parsing with weak grammatical constraints.
In *Proceedings of the ICASSP*. IEEE, April, 1987.

[Tomita 86] Tomita, M.
An efficient word lattice parsing algorithm for continuous speech recognition.
In *Proceedings of ICASSP*. IEEE, April, 1986.

[Ward 89a] Ward, W.H.
Understanding spontaneous speech.
In *Proceedings of the DARPA Speech and Natural Language Workshop*. DARPA, February, 1989.

[Ward 89b] Ward, W.H.
Modeling non-verbal sounds for speech recognition.
In *Proceedings of the DARPA Speech and Natural Language Workshop*. DARPA, October, 1989.

[Ward 90] Ward, W.H.
The CMU Air Travel Information Service: understanding spontaneous speech.
In *Proceedings of the DARPA Speech and Natural Language Workshop*. DARPA, June, 1990.

[Ward et al. 88] Ward, W.H., A.G. Hauptmann, R.M. Stern, and T. Chanak.
Parsing spoken phrases despite missing words.
In *Proceedings of the ICASSP*. IEEE, April, 1988.

[Young 89a] Young, S.R.
The MINDS system: Using context and dialog to enhance speech recognition.
In *Proceedings of the DARPA Speech and Natural Language Workshop*. DARPA, February, 1989.

[Young 89b] Young, S.R.
Use of pragmatics in speech recognition.
In *Proceedings of the AAAI Symposium on Spoken Language Systems*. AAAI, March, 1989.

[Young and Ward 88]
Young, S.R. and W.H. Ward.
Towards habitable systems: use of world knowledge to dynamically constrain speech recognition.
In *Proceedings of the Second Symposium on Advanced Man-Machine Interface through Spoken Language*. November, 1988.

[Young et al. 89a] Young, S.R., A.G. Hauptmann, W.H. Ward, E.T. Smith, and P. Werner.
High level knowledge sources in usable speech recognition systems.
Communications of the ACM 32(2):183-195, 1989.

[Young et al. 89b] Young, S.R., W.H. Ward, and A.G. Hauptmann.
Layering predictions: Flexible use of dialog expectation in speech recognition.
In *Proceedings of the International Joint Conference for Artificial Intelligence*. January, 1989.
Also available as technical report CMU-CS-89-105.

[Young, Hauptmann, and Ward 88]
Young, S.R., A.G. Hauptmann, and W.H. Ward.
An integrated speech and natural language dialog system: using dialog knowledge in speech recognition.
Technical Report CMU-CS-88-128, Carnegie Mellon University, Computer Science Department, April, 1988.

Table of Contents