

A LARGE VOCABULARY REAL TIME CONTINUOUS SPEECH RECOGNITION SYSTEM

J. Rabaey, R. Brodersen,
A. Stoelzle, D. Chen, S. Narayanaswamy, R. Yu, P. Schrupp,
H. Murveit (†), A. Santos (‡)
University of California, Berkeley, CA 94720
† SRI, Menlo Park
‡ ETSI Telecommunicacion, 28040 Madrid, Spain

1. INTRODUCTION

In the last couple of years, the Hidden Markov Model (HMM) [8],[3],[14],[6],[9],[10],[11] has proven to be a popular and effective technique for the implementation of speech recognition systems. We have developed a system architecture to implement real-time speech recognition using HMM algorithms with very large vocabularies and statistical grammars. This architecture will run (in real time) a class of such HMM-based systems including expanded versions of the BYBLOS system being developed at Bolt Beranek and Newman, the SPHINX system being developed at Carnegie-Melon University, and an HMM-based speech recognition system being developed at SRI International.

Machines based on this architecture can run HMM-speech recognition algorithms that update over 5,000 word-models and over 100K grammar transitions every 10 ms. This computation rate will allow realization of real-time recognition of 10,000- to 20,000-word vocabularies using certain types of bigram and trigram language models (see the Appendix), and is one or two orders of magnitude better than what is achievable with dedicated state-of-the-art general-purpose processors. The architecture meets these ambitious specifications because it uses special-purpose data paths in conjunction with parallelism and pipelining in the processor designs.

We are currently implementing the first prototype of this architecture. As it is being implemented with a 5Mhz cycle time and a non-pruned recognition search (to expedite the design and implementation process) it will be capable of recognizing continuous speech with a 3000 word vocabulary and a bigram language model.

After an overview of some alternative approaches, an overall view on the speech recognition architecture will be given. The paper will then expand in more detail on the architectural issues, involved in the realization of the application specific hardware. An overview of the speech model will be given in an appendix.

2. OTHER APPROACHES

There are several examples of real-time speech-recognition systems that have recently been developed. Although these systems represent significant advances, there is still much room for further advancement (and therefore a need for the kind of architectures proposed here that can cope with computationally more advanced systems as they evolve).

A 20,000-word speaker-dependent isolated-word recognition system has been put together at IBM [1]. It runs in real time with the aid of some specially designed hardware based on programmable digital signal processors. This system uses a sophisticated trigram grammar to improve recognition results by emphasizing likely three-word sequences. One important constraint that enables this hardware to recognize such a large vocabulary using a sophisticated grammar is that it is restricted to isolated-word speech. Isolated-word speech allows grammar processing to be done only once per word (about once a second) instead of once per spectral sample (about 100 times per second). Because the grammar-processing requirement for trigram grammars can be a large percentage of the system's total computational load for continuous-speech recognition, isolated speech represents a

significant simplification. Isolated speech also significantly reduces the number of word candidates to choose among in a given position, because words cannot overlap each other pruning is more effective. Due to these differences between isolated and continuous speech, the IBM hardware approach would have difficulty with a continuous-speech recognition system.

Carnegie Mellon University has constructed special hardware for 1000-word continuous-speech recognition based on several parallel microprocessors and digital signal processors. This hardware approaches real-time performance when the system uses a constraining bigram grammar. The CMU system can process in real time approximately 8,000 candidates (in this case, states in a HMM speech-recognition system) per 10-ms frame. This may be sufficient for a 1000-word vocabulary using a simple bigram grammar, but with a 20,000-word vocabulary (or a more complex, trigram grammar), as many as 100,000 to 300,000 such candidates might have to be processed per frame. Thus, a technology based on general-purpose processors may require large amounts of replication to run this task.

3. SYSTEM ARCHITECTURE

3.1. Overall Properties

The very high computation rates needed for recognition of continuous speech with flexible syntax can be achieved through the use of application-specific hardware, and, when appropriate, the design of special-purpose integrated circuits.

We have designed a new architecture that can achieve real-time performance for the kind of computationally demanding speech recognition algorithms described the Appendix. This architecture has the following specifications:

- Continuous-speech
- Large vocabulary (3,000 words in the initial prototype, 10,000 to 20,000 words in future versions with pruning and 10Mhz cycle times).
- Hidden-Markov-model recognition algorithm
- Up to four parallel discrete output densities
- Probabilistic finite-state syntactic constraints
- Breadth-first pruned recognition search (This will not be implemented in the first version of the system).

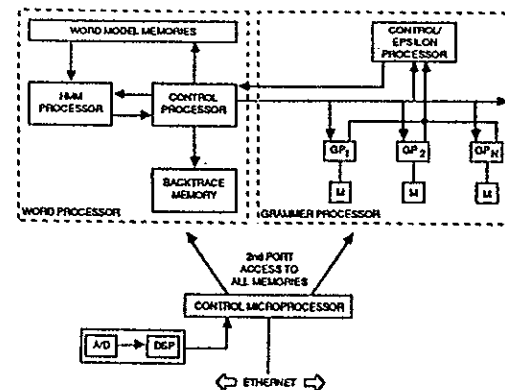


Figure 1: Overall System Architecture

significant simplification. Isolated speech also significantly reduces the number of word candidates to choose among in a given position, because words cannot overlap each other pruning is more effective. Due to these differences between isolated and continuous speech, the IBM hardware approach would have difficulty with a continuous-speech recognition system.

Carnegie Mellon University has constructed special hardware for 1000-word continuous-speech recognition based on several parallel microprocessors and digital signal processors. This hardware approaches real-time performance when the system uses a constraining bigram grammar. The CMU system can process in real time approximately 8,000 candidates (in this case, states in a HMM speech-recognition system) per 10-ms frame. This may be sufficient for a 1000-word vocabulary using a simple bigram grammar, but with a 20,000-word vocabulary (or a more complex, trigram grammar), as many as 100,000 to 300,000 such candidates might have to be processed per frame. Thus, a technology based on general-purpose processors may require large amounts of replication to run this task.

3. SYSTEM ARCHITECTURE

3.1. Overall Properties

The very high computation rates needed for recognition of continuous speech with flexible syntax can be achieved through the use of application-specific hardware, and, when appropriate, the design of special-purpose integrated circuits.

We have designed a new architecture that can achieve real-time performance for the kind of computationally demanding speech recognition algorithms described in the Appendix. This architecture has the following specifications:

- Continuous-speech
- Large vocabulary (3,000 words in the initial prototype, 10,000 to 20,000 words in future versions with pruning and 10Mhz cycle times).
- Hidden-Markov-model recognition algorithm
- Up to four parallel discrete output densities
- Probabilistic finite-state syntactic constraints
- Breadth-first pruned recognition search (This will not be implemented in the first version of the system).

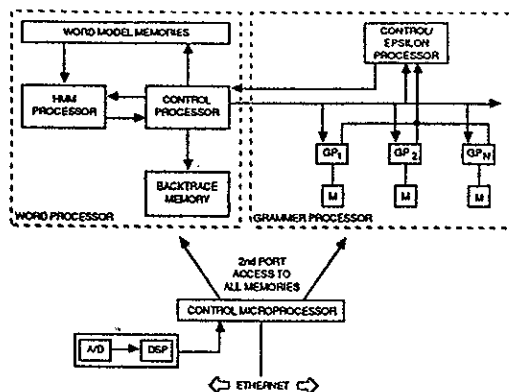


Figure 1: Overall System Architecture

These specifications were chosen because they apply to several relatively stable research speech-recognition systems [3], [6], and [9]; with some modification, they would apply as well to several others: [12], [10].

The overall system architecture is shown in Figure 1. The major processing is achieved with three separate modules and is controlled by a microcomputer:

- The front-end module samples the speech and extracts basic features.
- The word-processor module performs a Viterbi search and finds the probabilities that certain words in the vocabulary match certain features extracted by the front end.
- The grammar module controls the word processor by telling it what words to process next based on syntactic constraints. In initial system versions without pruning, it simply tells the word processor what the probability is that each word in the vocabulary will begin at the current time frame.

The rationale for partitioning the system in this way is both logical and computational. Logically, future changes in one module need not affect another. For instance, new types of grammars need not affect the design of the word processor. Computationally, the system is partitioned to meet the different processing needs of the different modules. For instance, the front-end processing is efficiently done with a few digital signal processor ICs (like the Texas Instruments TMS320 family [7]). The grammar-processing algorithms require many simple computations that can be implemented efficiently with a set of simple special-purpose integrated circuits that operate in parallel. The word-processor algorithms require access to a large amount of memory at a high rate and cannot be implemented by parallel processors as efficiently (unless several large memories are replicated); it is more effectively implemented with a more complex pipelined architecture that has several memory inputs operating concurrently.

3.2. Capabilities of this Architecture

With the above architecture, we expect to implement a speech-recognition system that is far beyond today's real-time capabilities. The reason is the special-purpose design, which allows the processors to achieve the specifications noted below:

- *Word Processor Raw Speed*—The inner loop of the recognition routine [Eqs. (A3) and (A4) in the Appendix] is performed every clock cycle (5,000,000 times per second, using our initial 5Mhz memory access rates).
- *Grammar Processor Raw Speed*—One grammatical transition [the inner loop of the MAX function for Eqs. (A9) and (A10)] is also processed every clock cycle. This rate increases linearly with the clock rate and number of grammar processors used. In the first version we will use 4 processors.

The above specifications have the following practical implications. For our initial system with 5Mhz memory access rates and no hypothesis pruning, and assuming a typical word has 18 states,¹ then Eqs. (A3) and (A4) would be run 18 times for each word; hence, $50,000/18 = 2777$ active words can be searched in 10 ms. This is the possible real-time vocabulary size for unpruned 5Mhz systems. When we incorporate pruning and a 10Mhz clock, then about 5,500 hypotheses can be searched. Then, because not all words in the vocabulary will be active at the same time (because of the hypothesis-pruning algorithm), the actual vocabulary size can be much larger than the number of active words. A single grammar module can process 100,000 grammar transitions for each (parallel) grammar processor that is used (at 10Mhz). Assuming 5,500 active words, an average of 18 next-word hypotheses could thus be processed in real time with a single processor system, or about 72 next-words per hypothesis for the expected four-processor system. We expect these rates to be appropriate for a speech-recognition system with a vocabulary of 20,000 words using a type of bigram and possibly trigram grammars.

¹ Using the BYBLOS [3] speech-recognition system as an example, 18 states translates to three states per phoneme times six phonemes per word, which is a reasonable approximation for large vocabularies.

3.3. Detailed Description of the Modules

The first-generation system, configured as shown in Figure 1, is organized around a control microcomputer that communicates with the outside world through an Ethernet, accepts speech features from the front-end board, and communicates with the speech-recognition system (which incorporates the word processor and the grammar processors).

The microcomputer board consists of a 68020 microprocessor with memory, control logic and logic for an Ethernet interface. A number of slave units are organized around this processor, communicating with it over the VME bus. The following chapters describe in detail the functionality and the architecture of the front-end and speech recognition sub-systems.

3.3.1. Front-End Processing

The front-end board, which is being designed at U.C. Berkeley comprises A/D and D/A converters, control logic, and two TMS32025 digital signal processors [7]. It performs the cepstral analysis on the incoming speech (after A/D conversion) followed by a vector quantization. The resulting data stream (800 bits/sec) forms the input to the HMM-recognizer, which is implemented on another set of boards.

3.3.2. Processing at the Word Level

Requirements

The architecture described in this section is capable of performing the Viterbi Algorithm [13] in real time to find the single best state sequence in the HMM word graph. Equation (1) formulates the inner loop of the Viterbi Algorithm:

$$P_t(j) = \max_{i \in \text{pred.}} [P_{t-1}(i) a_{ij}] \times b_j(O_t) \quad (1)$$

In order to update the probability of a given state j at time t the probabilities of the predecessor states of j , $P_{t-1}(i)$, computed at time $t-1$, have to be multiplied with their individual transition probabilities a_{ij} to state j . The maximum of these values then is multiplied with the output probability $b_j(O_t)$, which is the probability that state j matches the observation O at time t (Figure 2). If α is the number of predecessors for state j , this equation, which has to be performed for every state in every timeframe, involves $\alpha + 1$ multiplications and one maximum operation. Representing the probabilities as logarithmic quantities transforms multiplications into additions, which results in a substantial hardware saving.

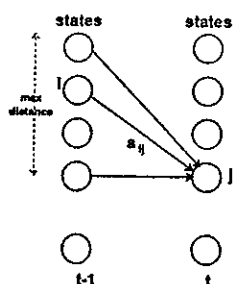
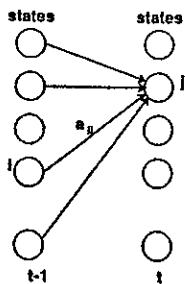


Figure 2: Viterbi Algorithm Figure 3: Viterbi Algorithm for Left-To-Right HMMs

Looking at the number of operations which have to be performed it is obvious that a custom implementation with multiple parallel units is necessary. In this way, it becomes feasible to update one state (or one evaluation of eqn (1)) per processor cycle.

The major bottleneck however is the amount of data needed to perform (1). In order to determine a predecessor for a given state a topology memory describing the graphs of all the given Markov Models has to be read. Then $P_{t-1}(i)$ and a_{ij} have to be accessed so that the probability of one possible transition of the ideal path can be computed. To finally compute $P_t(j)$ the output probability $b_j(O_t)$ must be read and the result written to memory. Accumulating all the data needed and considering the real time aspect, the resulting bitrate is 61×10^6 Bits per 10msec or 6×10^9 bits per second. However, the bandwidth between memory and a single chip is limited by the number of pins per chip (< 257) and the speed of the data transfer. To keep up with the data rate described above using high density DRAMs (cycle time = 200nsec) 1200 datapins would be required. Using fast (and expensive) static RAMS is not a reasonable solution to this bandwidth problem since the amount of memory required to store the word models and the intermediate results is large (8.5 MBytes).

Besides these algorithmic specific requirements, it is desirable to pursue the following architectural features, which may help to simplify the implementation in a drastic way.

- If possible, memory accesses should be organized sequentially. This avoids the use of expensive address arithmetic and simplifies the DRAM memory refreshing process.
- Control flow should be kept straightforward: In order to obtain the required arithmetic throughput, a heavy use has to be made of deep pipelining. This makes the usage of conditional operations much harder and less efficient.

Architecture

The architecture of the sub-system working on the word level is based on the fact that the word models are left-to-right Hidden Markov models. That means that the word graph is always traversed from left to right (as shown in Figure 3), so that the predecessors of a given state are situated to the left of that state. The number of possible predecessors is thus reduced to a very small number compared to the number of states. The adopted word model has the restriction that a state j must never have predecessors that are more than 16 states behind (or left of) j . If the word processor has the capability to store 16 successive predecessor probabilities on chip, then all the predecessor probabilities necessary to compute (1) are readily available. In the word processor implementation, the predecessor probabilities $P_{t-1}(j)$ are sequentially transferred from external memory into an on-chip buffer. This read operation is synchronized with the evaluation of (1) in such a way that $P_t(i)$ is being computed, while $P_{t-1}(i)$ is being read. This technique makes it possible to parallelize the computation of the inner loop of (1),

$$P_{t-1}(i) \times a_{ij} \quad (2)$$

on one single chip without increasing chip i/o for reading the different predecessor probabilities $P_{t-1}(i)$.

Since most states in the Models don't have more than three predecessors, three identical concurrent data paths have been provided (Figure 4). The data paths are heavily pipelined (11 stages) in order to achieve the necessary arithmetic throughput. Each data path reads the predecessor probabilities from a private copy of the on chip memory so that memory contention is avoided. To increase the throughput, these buffers are designed such that a read and a write operation can be performed within one clock cycle. Reading the external memory as well as writing these bidirectional memories (modulo 16) is done in a sequential fashion thereby overwriting the probability of the state which is 16 locations left of the current state. The (random) read addresses to access the different $P_{t-1}(i)$'s are generated from offsets relative to the current write address.

The above architecture makes it possible to use one single chip to evaluate the Viterbi equation (1) in real time at a rate of one state/cycle. All the memories in the system involved in solving this equation are sequentially addressed and share one single address bus. If the number of predecessors for a particular state exceeds three, more processor cycles are needed to update the state. However, the memory describing the Markov Models is organized in such a way that the counter generating the memory addresses is not interrupted. This simplifies the control of the word processor and the entire

subsystem by avoiding conditional branches. The word processor acts as a slave device of the grammar processor, which will be described next.

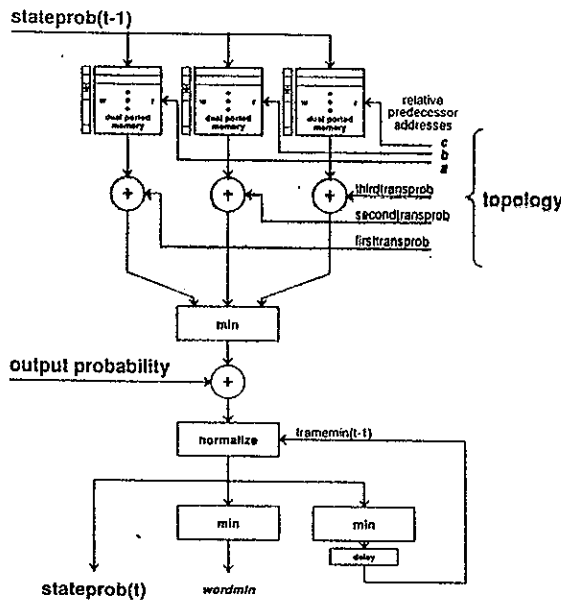


Figure 4: Word Processor Architecture

3.3.3. Processing at the Grammar Level

The major task of the grammar processor is to evaluate once per time frame the grammar node probabilities of all the words in the grammar. In addition, for each word a pointer has to be stored, defining the most probable path leading to that particular word at that point in time. At the end of the sentence, those pointers will be used to reconstruct the optimal word sequence (back tracing).

This task requires the following operations in the grammar processor :

- Sequentially fire up every word in the grammar with the input grammar node probability, derived during the previous time frame. The word and its probability are passed to the word processor for evaluation of the internal state. Eventually, it is possible to reduce the computational complexity by pruning words with a probability lower than a certain threshold. This feature is however not implemented in the current system.

- Determine the input grammar node probabilities for all the words, based on the output grammar node probabilities returned by the word processor. In contrast to the structure of the word model, the grammar model does not form a left-right graph, since every word can (in principle) follow every other word. Instead of using the predecessor technique of the word processor, we have adopted a successor based evaluation of the grammar graph (Figure 5). For each word i returned from the word processor with output grammar node probability PGO, the input grammar node probability PGI of the successor words j can be updated using (3) :

$$PGI_{t+1}(j) = \max(PGO_t(i) \times c_{ij}, PGI_{t+1}(j)) \quad (3)$$

where c_{ij} represents the transition probability from word i to word j . Once again, multiplications can be replaced by additions by using a logarithmic number representation. The successor nodes are stored in the order of descending transition probability, so that part of evaluation can be pruned when the result of (3) drops below a variable threshold.

- A statistical grammar requires a complete graph, where every node connects to every other node, but this would require excessive storage. Therefore we have adopted a simplified model (called the ϵ -model) for arcs with low probabilities. In this model, the low probabilities c_{ij} are approximated by a simplified probability ϵ_{ij} , (with $\epsilon_{ij} = \epsilon_i \times \epsilon_j$). This reduces the storage requirements (for a vocabulary of size N) from N^2 to $2N$. It is the task of the grammar processor to evaluate those ϵ -probabilities and to update the input grammar node probabilities when that computed ϵ -probability becomes dominant. The amount of computation involved here is only minor and does not require any special architectural optimization.

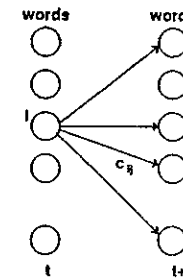


Figure 5: Grammar Graph Evaluation

The bottleneck of the grammar processor is clearly located in the evaluation of equation (3). For a system with 3000 words with an average of 100 successors per word, 300,000 evaluations are required in a worse case scenario. A custom processor, requiring 1 clock cycle per evaluation, could handle only 50,000 evaluations (for a clock frequency of 5 Mhz). Therefore, parallel processing is a necessity. The partitioning of the task over a set of parallel processors is however not straightfor-

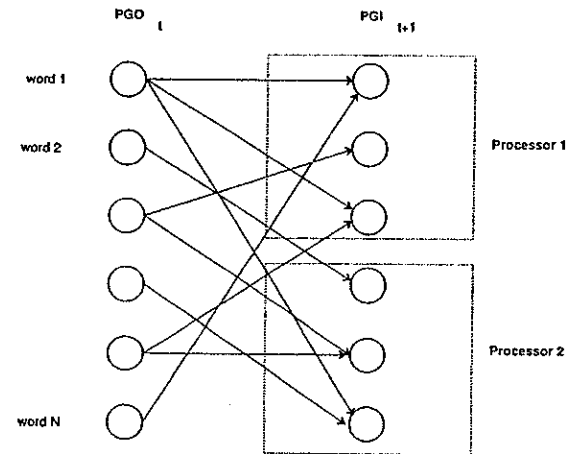


Figure 6: Memory Partitioning

ward. In fact, all those processors have to read and write from the PGI_{i+1} memory, so that contention with its necessary slow down cannot be avoided. One possibility is to provide multiple copies of that memory and to combine the information from the different memories upon transmission in the next frame. Due to the read-write requirement of equation (3), fast static memories with 25 nsec access time have to be used for the PGI-memory. Replication of that memory is therefore expensive. A better solution is to partition the memory into different sections and have one processor supporting each section (Figure 6). A word, returned from the word processor is passed to the different concurrent processors after which each processor picks only those successors, which are part of its own memory space. In order to balance the load between the different processors, buffering FIFO's have been provided between the word-processor and the concurrent grammar processors. A total view on the grammar processor is given in Figure 7. The concurrent processors have been designed in such a fashion that an arbitrary number of processors can be used. In this way, this design can easily be extended to system with larger grammars.

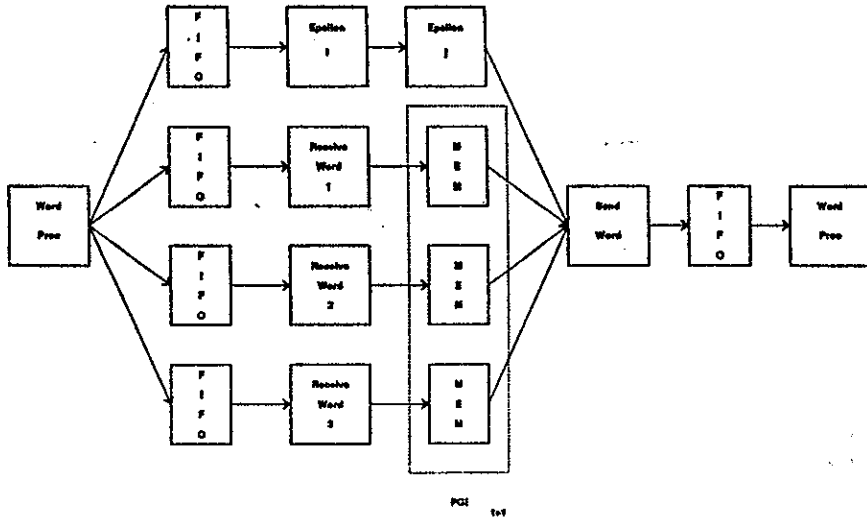


Figure 7: Architecture of the Grammar Processor

4. CONCLUSIONS

A large vocabulary real time continuous speech recognition system has been described. It has been shown that the largest bottleneck in such a system is located in the access of the memories. The presented architecture exploits a variety of techniques, such as partitioning and replication in order to cope with this memory bottleneck. The required throughput is achieved with the aid of extensive pipelining (up to eleven levels deep) and concurrency. The architecture allows for the extension to larger vocabularies by just adding more parallel units. Pin count considerations have resulted in the definition of five custom integrated circuits, which are currently under development.

Appendix

SPEECH-RECOGNITION ALGORITHMS

This appendix is provided as a brief description of the basic speech-recognition approach we plan to use on this project, namely, hidden Markov model recognition algorithms used in conjunction with finite-state statistical grammars. These algorithms were chosen because they seem to underlie the most successful speech-recognition systems to date, and show continued promise for improvement. IBM has demonstrated a 20,000-word speaker-dependent isolated-speech recognition system based on this technology [1]; BBN has demonstrated a system with high performance on a 1000-word continuous-speech speaker-dependent task [3], and CMU and SRI have demonstrated systems with good performance on speaker-independent continuous-word tasks: [6] and [9].

Hidden Markov Model Speech-Recognition Algorithms

In Hidden Markov Model (HMM) speech-recognition systems, speech is modeled as being generated by a Hidden Markov process; i.e., a process that has a finite number of states, state-to-state transition probabilities that depend only on the two states, and a probability density function for outputting some speech component. Because speech can be modeled as a sequence of features, the output speech component is a feature or a vector of features generated according to the state-dependent probability density function.

One approach to applying hidden Markov modeling to continuous-speech recognition systems is shown below: We first collect models M_i corresponding to each word in the recognition vocabulary V . A composite model M comprising all M_i that could generate any sequence of words in V is created by adding transitions from each model's final states to each model's initial states. Now, given an output sequence O of speech features, the word sequence corresponding to the state sequence S in M that maximizes $P(O, S | M)$ is recognized as the most likely sequence of words to generate O and would be the recognized sentence.

The following technique can be used to find the state sequence that maximizes $P(O, S | M)$. Let $M(\pi, A, B)$ be a (composite) Markov model with $\pi(s)$ being the initial state probability distribution, $A(s, t)$ being the probability that state s transitions to state t , $B_s(o)$ being the probability that state s outputs o , $O = \{o_i\}$ being a sequence of N outputs, and $S = \{s_i\}$ being a sequence of N states. The probability of the state sequence and the outputs given the model is

$$P(O, S | M) = \pi(s_1) \cdot B_{s_1}(o_1) \cdot \prod_{i=2}^N [A(s_{i-1}, s_i) \cdot B_{s_i}(o_i)] \quad (A1)$$

Let us define $P(O_i, s | M)$ as the probability of the most probable state sequence that ends with state s and generates the i outputs $O_i = \{o_1, o_2, \dots, o_i\}$. Let $\hat{P}(O_i, s | M)$ be the corresponding state sequence. We will use these to find the most likely sequence for generating the whole output sequence by means of dynamic programming. Note that

$$P(O_1, s | M) = \pi(s) \cdot B_s(o_1) \quad (A2)$$

$$P(O_i, s | M) = B_s(o_i) \cdot \text{MAX}_p [P(O_{i-1}, p | M) \cdot A(p, s)] \quad (A3)$$

and

$$\hat{P}(O_i, s | M) = \hat{P}(O_{i-1}, p | M) \cdot s \quad (A4)$$

where the p in Eq. (A4) is the state p that maximizes Eq. (A3), and \cdot is the concatenation operator.

Given these equations, $P(O_i, s | M)$ can be computed for all states for O_1 , then all states for O_2 , and so on until probabilities for all states for O_N are computed. Then the final state s that maximizes $\text{MAX}_s P(O_N, s | M)$ is solved for and $\hat{P}(O_N, s | M)$ is the most likely state sequence whose proba-

bility is $P(O_N, S | M)$.

Finite-State Grammatical Constraints

The preceding section describes a continuous-speech recognition system configured by allowing all words to follow each other. The problem can be constrained more—thereby improving recognition performance—by only allowing sensible word sequences. Two approaches to achieving this goal use finite-state networks. The first approach using finite-state network to constrain the problem is to connect the word models of Section 1 of this appendix according to a finite-state grammar [3]. This approach is shown in Figure A-1, where the ellipses are word networks. Note that word networks may have to be replicated if they appear more than once in the grammar.

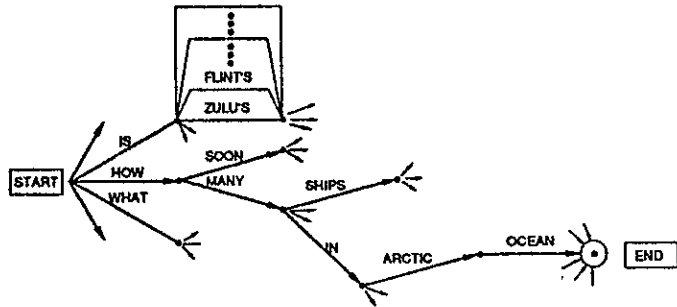


FIGURE A-1 FINITE-STATE GRAMMATICAL CONSTRAINTS

Another approach to constraining the problem based on finite-state networks is the use of N -gram statistical models [4]. Because the probability of an M -word sequence $W = \{W_i\}$ can be represented as the product of the probabilities of each successive word, given the previous words,

$$P(W) = P(W_1) \cdot P(W_2 | W_1) \cdot P(W_3 | W_2, W_1) \cdots P(W_M | W_{M-1}, W_{M-2}, \dots, W_1) \quad (A5)$$

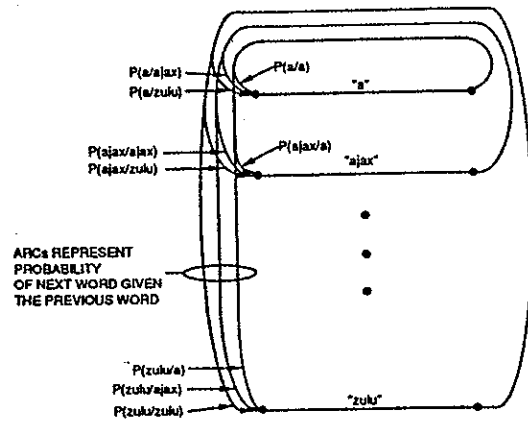


FIGURE A-2 BIGRAM LANGUAGE MODEL

To simplify this representation [and because the large-order probabilities in Eq. (A5) cannot reasonably be estimated], we use an N -gram model that approximates the conditional probability of one word given all the previous words as the probability of one word given the past $N - 1$ words:

$$P(W_i | W_{i-1}, W_{i-2}, \dots, W_1) = P(W_i | W_{i-1}, W_{i-2}, \dots, W_{i-N}) \quad (A6)$$

A simple model, the bigram model ($N = 2$), represents

$$P(W_i | W_{i-1}, W_{i-2}, \dots, W_1) \approx P(W_i | W_{i-1}) \quad (A7)$$

This can also be represented in a finite-state network, as shown in Figure A-2. In this figure, each node corresponds to a probability distribution for what the next word is given the previous $N - 1$ words (encoded by the state). One might expect that for trigram grammars ($N = 3$) with a V -word vocabulary, V^2 nodes would be required to encode all the possible previous two words. However, typically far fewer than V^2 distributions can be reasonably estimated, and thus many of the two-word histories share distributions and states. Currently, trigram models are the largest order models that can be practically estimated.

Although it is often the case that probabilities of the form $P(W_i | W_{i-1}, W_{i-2})$ —or even $P(W_i | W_{i-1})$ —are not available, it is possible to estimate these unavailable probabilities as functions of the proposed word and the history. That is,

$$P(W_i | W_{i-1}, W_{i-2}) = C_1(W_i) \cdot C_2(W_{i-1}, W_{i-2}) \quad (A8)$$

or

$$P(W_i | W_{i-1}) = C_1(W_i) \cdot C_2(W_{i-1})$$

A Complete Speech-Recognition System

There are several ways that hidden Markov models for words and finite-state word-grammars can be combined into speech-recognition systems. The speech-recognition architecture described in this paper uses the following approach: Speech is processed by a front-end processor; an output vector from this processor is passed to the HMM processor every time frame (typically every 10 ms). The grammar processor keeps a data structure called the active words, which is initialized before frame 1 to the list of words that can begin a sentence. At each frame t , the grammar processor instructs the HMM processor to compute Eqs. (A2), (A3), and (A4) for all states in all the words of the active word list. For each word in the active word list, the HMM processor returns to the grammar processor

- The probability of the most probable state in that word, $P(w_i, t)$
- The probability of the last-state of that word, $P_{last}(w_i, t)$
- A tag, $TAG_{last}(w_i, t)$, which represents the sequence of words that led to $P_{last}(w_i, t)$.

The grammar processor then computes the probabilities that new words can begin the next frame continuing from these last states as the product of the last state's probability and the grammar transition's probability.

Thus, the probability² that word w_j can start at frame $t + 1$ using a bigram grammar is

$$P_{start}(w_j, t+1) = \text{MAX}_{w_i} \left[P_{last}(w_i, t) \cdot P(w_j | w_i) \right] \quad (A9)$$

and the tag or sequence of words corresponding to this hypothesis is

$$TAG_{start}(w_j, t+1) = TAG_{last}(w_i, t) * w_j \quad (A10)$$

where w_i in Eq. (A10) is the w_i that maximized Eq. (A9). At the end of frame t , the grammar processor places all words w_j in the active list for frame $t + 1$ if either $P_{start}(w_j, t+1)$ or $P(w_j, t)$ is

² Actually the true probability would use a sum instead of a max in Eq. (A9). Using the max results in the probability of the best hypothesis as opposed to the probability of all possible hypotheses.

greater than a pruning threshold. This continues until frame = N , where the probability of the best word sequence is

$$\text{MAX}_{w_i} P_{\text{last}}(w_i, N) \quad (\text{A11})$$

and the recognized word sequence is $\text{TAG}_{\text{last}}(w_i, N)$ for the w_i that maximized Eq. (A11).

The above procedure is modified so that the approximations shown in Eqs. (A8) can be used. Thus, using bigram grammars as an example, each word has a C_1 value and a C_2 value [see Eq. (A8)] in addition to its list of transition probabilities $P(W_i | W_j)$. If we define

$$\epsilon - \text{transition} = \text{MAX}_{w_i} \left[P_{\text{last}}(w_i, t) \cdot C_1(w_i) \right] \quad (\text{A12})$$

and

$$\text{TAG}_{\epsilon} = \text{TAG}_{\text{last}}(w_i, t) \quad (\text{A13})$$

where w_i in Eq. (A13) is the w_i in Eq. (A12) that was maximum, then Eqs. (A9) and (A10) can be rewritten as

$$P_{\text{start}}(w_j, t+1) = \text{MAX} \left\{ \text{MAX}_{w_i} \left[P_{\text{last}}(w_i, t) \cdot P(w_j | w_i) \right], \epsilon - \text{transition} \cdot C_2(w_j) \right\} \quad (\text{A14})$$

$$\text{TAG}_{\text{start}}(w_j, t+1) = \text{OR}_{\epsilon} \left[\text{TAG}_{\text{last}}(w_i, t), \text{TAG}_{\epsilon} \right] \cdot w_j \quad (\text{A15})$$

where OR_{ϵ} uses TAG_{ϵ} if the $\epsilon - \text{transition}$ was chosen in Eq. (A14), and uses the TAG_{last} value otherwise.

REFERENCES

- [1] A. Averbuch, L. Bahl, R. Bakis, P. Brown, A. Cole, G. Daggett, S. Das, K. Davies, S. De Gennaro, P. de Souza, E. Epstein, D. Fragleigh, F. Jelinek, S. Katz, B. Lewis, R. Mercer, A. Madas, D. Nahamoo, J. Picheny, G. Shichman, and P. Spinelli: "An IBM-PC Based Large-Vocabulary Isolated-Utterance Speech Recognizer," *Proc. ICASSP 86: 1986 International Conference on Acoustics Speech and Signal Processing, Tokyo Japan* (Institute of Electrical and Electronic Engineers, Piscataway, New Jersey), pp. 53-56 April 1986
- [2] J.K. Baker, "Stochastic Modeling for Automatic Speech Understanding", in *Speech Understanding*, D.R. Reddy Ed., New York, Academic Press, pp.521-542, 1975.
- [3] Y.L. Chow, M.D. Dunham, O.A. Kimball, M.A. Krasner, G.F. Kubala, J. Makhoul, P.J. Price, S. Roucos, and R.M. Schwartz, "BYBLOS: The BBN Continuous Speech Recognition System," *Proc. ICASSP 87: 1987 International Conference on Acoustics Speech and Signal Processing, Dallas, Texas* (Institute of Electrical and Electronic Engineers, Piscataway, New Jersey), pp. 89-92 April 1987
- [4] F. Jelinek, "The Development of an Experimental Discrete Dictation Recognizer", *IEEE Proceedings*, Vol. 73, No 11, pp. 1616-1624, Nov. 1985.
- [5] R. Kavaler, M. Lowy, H. Murveit and R. Brodersen, "A Dynamic-Time-Warp Integrated Circuit for a 1000-Word Speech Recognition System", *IEEE Journ. Solid State Circuits*, Vol. SC-22, No 1, pp. 3-14, February 1987.
- [6] K. Lee and H. Hsiao-Wuen, "Large Vocabulary Speaker-Independent Continuous-Speech Recognition Using HMM," *Proc. ICASSP 88: 1988 International Conference on Acoustics Speech and Signal Processing, New York, New York* (Institute of Electrical and Electronic Engineers, Piscataway, New Jersey), pp. 123-126 April 1988
- [7] K. Lin, G. Frantz and R. Simar, "The TMS320 Family of Digital Signal Processors", *IEEE Proceedings*, Vol 75, No 9, pp. 1143-1159, September 1987.
- [8] A.A. Markov, "An example of Statistical Investigation in the text of Eugene Onyegin Illustrating Coupling of Tests in chains", in *Proc. Acad. Sci. St. Petersburg*, vol. 7, pp. 153-162, 1913.
- [9] H. Murveit and M. Weintraub, "1000-Word Speaker-Independent Continuous Speech Recognition Using Hidden Markov Models," *Proc. ICASSP 88: 1988 International Conference on Acoustics Speech and Signal Processing, New York, New York* (Institute of Electrical and Electronic Engineers, Piscataway, New Jersey), pp. 115-118 April 1988
- [10] D.B. Paul, "A Speaker-Stress Resistant Isolated-Speech Recognizer," *Proc. ICASSP 87: 1987 International Conference on Acoustics Speech and Signal Processing, Dallas, Texas* (Institute of Electrical and Electronic Engineers, Piscataway, New Jersey), pp. 713-716 April 1987
- [11] L. Rabiner and B. Huang, "An Introduction to Hidden Markov Models", *IEEE ASSP Magazine*, pp. 4-16, Jan. 1986.
- [12] L.R. Rabiner, J.G. Wilpon, and B.H. Juang, "Performance Evaluation of a Connected Digit Recognizer," *Proc. ICASSP 87: 1987 International Conference on Acoustics Speech and Signal Processing, Dallas, Texas* (Institute of Electrical and Electronic Engineers, Piscataway, New Jersey), pp. 101-104 April 1987
- [13] A. Viterbi, "Error Bounds for convolutional codes and an asymptotically optimal decoding algorithm", *IEEE Trans. Inform. Theory*, vol IT-13, pp. 260-269, 1967.