# Recent Progress on the VOYAGER System

**Victor Zue, James Glass, David Goodine, Hong Leung, Michael McCandless, Michael Phillips, Joseph Polifroni, and Stephanie Seneff**

Room NE43-601
Spoken Language Systems Group
Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, MA 02139

## Introduction

The VOYAGER speech recognition system, which was described in some detail at the last DARPA meeting [9], is an urban exploration system which provides the user with help in locating various sites in the area of Cambridge, Massachusetts. The system has a limited database of objects such as banks, restaurants, and post offices and can provide information about these objects (e.g., phone numbers, type of cuisine served) as well as providing navigational assistance between them. VOYAGER accepts both spoken and typed input and responds in the form of text, graphics, and synthesized speech. Since the last meeting, we have made developments to VOYAGER that have had an impact on the usability of the system.

In this paper, we will describe these developments and report on evaluation results after these changes were incorporated into the system. Two key developments to VOYAGER are a tighter integration of the speech and natural language components and a pipelined hardware implementation leading to a speed-up in processing time from approximately 12 times real time to approximately 5 times real time. We also discuss here a number of incremental improvements in the word-pair grammar, pronunciation networks, and the back-end capabilities.

## SR/NL Integration

In our initial implementation of VOYAGER, the integration of speech and natural language components was accomplished by obtaining the best word sequence from the recognizer and passing that word sequence to the natural language system. Modifying the speech recognition component to produce a list of the top scoring word sequences provides a convenient means for increasing the level of integration of the speech recognition and natural language components [2]. In this way, the natural language system can be run successively on each of the word sequences to find the highest scoring sequence that passes the natural language constraints.

### Two-stage N-Best search

Previously, to produce the top scoring word sequence, our speech recognition system used Viterbi search [4,10]. This algorithm provides an efficient search for the top word sequence but does not directly provide the top N word sequences. Others have chosen to modify this search by keeping track of the top N word sequences at each point in the search [2]. We also use a modification of Viterbi search to produce the top N word sequences. In our algorithm, we first use Viterbi search to compute the best partial paths both arriving and leaving each lexical node at each point in time. The algorithm then successively extracts the next best complete path by searching through the precomputed matrix of partial paths to find the highest scoring path that has not yet been extracted.

To extract the N highest scoring paths from the precomputed matrix of partial paths, this two-stage N-Best search utilizes the fact that each new path must either contain a new node-pair (a given lexical node at a given point in time) or must be some combination of portions of the paths found so far. So, the search must keep track of the best path passing through each node-pair (which is the sum of the scores of the best arriving and leaving paths computed by the Viterbi search) and must also keep track of all combinations of the complete paths found so far. The next highest scoring path can be found by taking the highest scoring path either through a new node-pair or from some combination of previous paths.

The computation of the partial paths either arriving or leaving each lexical node at each point in time is the same as needed for the forward Viterbi search for the top scoring word sequence. Therefore, the total computation needed for this algorithm is two times the Viterbi search plus the amount of computation we need to extract the paths from the precomputed matrix. We have measured the computation time and memory use of our implementation of this algorithm as a function of the number of sentence hypotheses. This resource use is plotted as the open symbols in Figure 1. This experiment was performed on 495 utterances with a test set word-pair perplexity of 73 and a vocabulary size of 350 words.

This algorithm is somewhat different from the frame- synchronous algorithm described previously [2], and has a number of advantages and disadvantages. An important advantage for VOYAGER is that we do not have to choose N before performing the search. In the system, we are able to check each word string as it is produced by the recognizer and tell the system to quit as soon as one of the sentences passes the natural language constraints. Also, at least in our segment based system, this algorithm is quite efficient. This efficiency advantage may not hold for frame-based systems. As described above, it is necessary to keep track of pointers for the partial paths for the entire node-pair matrix. This is not a

large problem in our system, since the nodes are at a segment level rather than at a frame level. Furthermore, we needed to keep track of these pointers for the forward pass in the Viterbi search anyway, and so the memory requirements only increase by a factor of two. A disadvantage of this approach, at least when implemented on a per utterance-basis as described, is that more than two-thirds of the search cannot be started until the end of the utterance is reached. Therefore, this part of the processing cannot be pipelined with the incoming speech.

## A* search

Passing the top N word sequences to the natural language system is an improvement over passing only the single best scoring sequence, but our goal is to make better use of the natural language constraints at an early stage of the search. The A* search algorithm can provide a flexible mechanism for making use of natural language constraints because it keeps a stack of partial paths that are extended based on an evaluation function. Non-probabilistic natural language constraints can be used to prune partial hypotheses either before they are put on the stack or before they are extended. Prediction capability of the natural language system can be used to propose ways of extending partial paths. Finally, probabilities of partial paths provided by the natural language system can be incorporated into the evaluation function.

The A* search evaluation function is defined as

$$f^*(p) = g(p) + h^*(p),$$

where $f^*(p)$ is the estimated score of the best path containing partial path $p$, $g(p)$ is the score for the match from the beginning of the utterance to the end of the partial path $p$, and $h^*(p)$ is an estimate of the best scoring extension of the partial path $p$ to the end of the utterance [1]. This search is admissible if $h^*(p)$ is an upper bound on the actual best scoring extension of partial path $p$ to the end.

To efficiently apply A* search to spoken language systems, it is important to have as tight a bound as possible for $h^*(p)$ since a looser bound results in increased computation. We can use Viterbi search to compute this upper bound by searching back from the end of the utterance to find the best score to the end for each lexical node at each point in time. If the constraints we use in the Viterbi search to compute the best score to the end are a subset of the full natural language constraints, this estimate of the best score to the end is guaranteed to be an upper bound on best score to the end given the full constraints.

The A* search allows a large amount of flexibility in when to apply the natural language constraints. For example, we can wait until we have entire sentence hypotheses before applying the full natural language constraints. This turns the A* search into an N-best algorithm [3] and allows us to compare it directly to the other N-best algorithms. We computed processing time and memory use for our implementation of this algorithm and plotted it in Figure 1. For the top 1 word sequence, this algorithm requires about the same amount of resources as our implementation of Viterbi search and the amount of resources increases approximately linearly with N at least for small N.

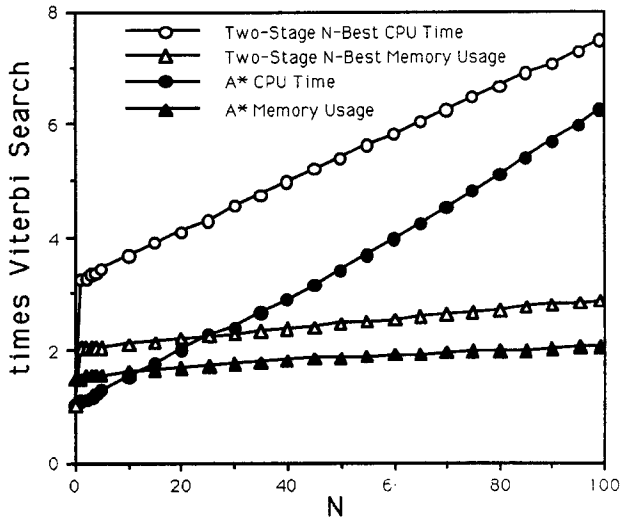


**Figure 1:** This figure compares the CPU and Memory usage of the A* N-Best search with the Two-Stage N-Best algorithm as a function of N. All quantities are relative to the resource use of our implementation of Viterbi search for the top scoring word sequence.

We have begun to perform experiments to determine which natural language constraints to apply at an earlier stage of the A* search. There is a tradeoff between the cost of applying the constraint and the amount of other computation that is saved by the application of the constraint. Since we are able to apply word-pair constraints at a very small cost (by precompiling them into the lexical network), we have been applying word-pair constraints at the lowest levels in all of these experiments.

## Word pair constraints

In our initial implementation of VOYAGER, the search was constrained by a word-pair language model obtained directly from the training utterances. This word-pair language model had a perplexity of 22 and a coverage of 65%. However, this word-pair language model was obtained without consideration of the constraints from TINA and, therefore, did not match the capabilities of the full system. Utterances that TINA could accept as well-formed were sometimes rejected by the word-pair language model.

Now that we are moving towards tighter integration of the speech and natural language components, we are not so dependent on the constraints of a simple language model. However, if it is possible to automatically extract the local constraints of the natural language system, we can save computation by making use of them. Even in a tightly integrated speech and natural language system, it is possible to compile these constraints directly into a lexical network. The overall accuracy will not suffer as long as we can guarantee that the

constraints of the local language model are a subset of the full constraints.

A useful facility for deriving inexpensive recognizer constraints from a natural language system would be a mechanism to extract an exhaustive word-pair language model automatically from the parent grammar. To this end, we explored a number of procedures to discover all legitimate two word sequences allowed by TINA. We assessed the resulting language models by measuring coverage and perplexity on our designated development set of about 500 sentences.

The simplest approach is to exhaustively generate all terminal -pairs directly from the context-free rules, without applying any other semantic or syntactic constraints. We tried this approach, and, as expected, it gave 100% coverage on the test set, but with a very high perplexity ($\approx$ 200). In an attempt to reduce the perplexity, we tried some permutations of this method. We first discarded any rules that did not show up in our set of 3000 training sentences. This resulted in a loss of coverage on 10% of the test sentences, so this idea was abandoned. A second, more conservative, idea was to allow the disappearance of trace nodes only within those rule contexts that showed up in the training set. This resulted in a slight reduction in perplexity to 190, and the coverage remained at 100%.

The other approach we tried was to make use of TINA's generation capability to generate sentences at random, and then use the resulting terminal pairs to update the word-pair language model. This approach has the disadvantage that it can never be guaranteed that TINA's language model is exhaustively covered. However, it permits the incorporation of local syntactic and semantic constraints. We decided to discard semantic match requirements in the trace mechanism, so that a sentence such as "(What restaurant)$_i$ is it (t$_i$) from MIT to Harvard Square?" would be accepted. We did away with the trace mechanism in generation since these long distance constraints are generally invisible to the word-pair language model. This was necessary because, when semantic matches are required, generation usually picks the wrong path and aborts on constraint failure. As a consequence, paths with traces are rarely visited by the generator and may not show up in our word-pair language model.

This method was quite successful. TINA can generate 100,000 sentences in an overnight run, and the resulting word-pair language model had a perplexity of only 73 with a single missed word-pair in the test set. We therefore decided to incorporate this word-pair language model into the recognizer.

## Increased Coverage

As we have described previously [9], the command generation component translates the natural language parse to a functional form that is evaluated by the system. This component has been made more flexible, in part due to our experience with developing an ATIS system [6]. We have extended the capabilities of the back-end functions to handle more complex manipulations. Some of these changes were motivated by an examination of our training data. In other cases, we were interested in knowing if our framework could handle manipulations commonly used in other database query systems. For this reason we included conjunction and negation, even though they are rarely used by subjects (except by those with a natural language processing background!). As a result of these modifications, the system is now capable of handling queries such as "Show me the Chinese *or* Japanese restaurants that are *not* in Central Square," or "Do you know of any *other* restaurants near the main library?"

## Pronunciation Networks

Pronunciation networks and their expansion rules were modified as a result of the increased amount of training data. An effort was made to modify both the networks and the rules as consistently and minimally as possible. The VOYAGER dictionary was periodically reviewed to insure that pronunciations were consistent in terms of both segmentals and the marking of stressed and unstressed syllables. When phonetically labelling the VOYAGER corpus, unusual or new pronunciations were noted by the labelers, who conferred on phonetic transcriptions. New pronunciations were entered into the dictionary or added to the lexical rules when it was felt that the phenomena they represented were sufficiently generalizable to the corpus as a whole. Aberrant pronunciations or mispronunciations were not included.

## Current Implementation

In the initial implementation of VOYAGER, the system ran on a Sun 4/280 using a Macintosh II with four DSP32Cs as a front-end. That system was not pipelined and took approximately 12 times real time before the top-choice utterance appeared. Since that time we have developed a pipelined implementation of VOYAGER on a new set of hardware as illustrated in Figure 2. We are using four signal processing boards made by Valley Enterprises, each of which has four DSP32C's. Each processor has 128Kbytes of memory and operates independently of the others (in the board configuration that we have been using). Communication with the host is through the VME bus of the host. The host may read from any location of any of the DSP32C's memory while the DSP processor is running. The host may simultaneously write to any combination of the four DSP32C's memories. For speech input and playback, we are using an A/D D/A made by Atlanta Signal Processing Inc. This has a high speed serial interface which connects to the serial port of one of the DSP32Cs. We are currently using a Sun4/330 with 24Mbytes of memory as a host. We are running the natural language and response generation components on a separate Sparcstation. These parts of the system are written in Lisp; they have fairly large memory requirements and would slow down the processing if run simultaneously on the same host as the speech recognition system. Also, our Sun4/330 has no display. The entire system could easily run on a single host with more memory plus a display.

It has been straightforward to divide the processing for VOYAGER's front-end [9] into subsets which can each be performed in real-time by a single DSP32C and which do not require excessive amounts of intercommunication. The auditory model can be broken up by frequency channel and

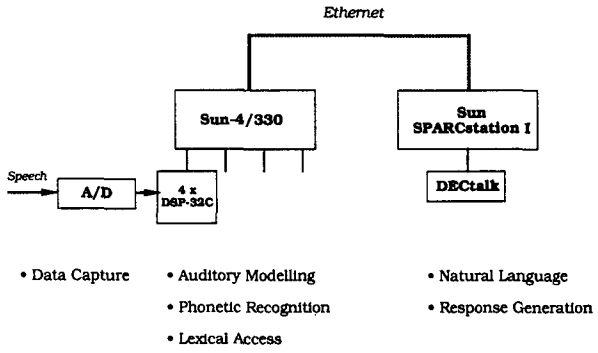


**Figure 2:** This figure shows the current hardware configuration of the VOYAGER system.

therefore the current representation could be run on up to 40 different processors. The dendrogram computation is difficult to divide among processors, but fortunately it runs in under real time on a single DSP32C. The computation of acoustic measurements and phonetic classification is done on a segmental basis and could be broken up by segment if necessary.

We have implemented each processor-sized subset of the computation for the DSP32C with a circular input and output buffer. Each of these processes monitors the input and output buffers, and runs as long as the input buffer is not empty and the output buffer is not full. The host keeps larger circular buffers for each of the intermediate representations and fills the input buffers and empties the output buffers of the DSP processors as the data become available. We have used the same general mechanism for each part of the system, allowing us to easily change the various parts of the system as new algorithms are developed. All parts of the system before natural language processing are written in C with the exception of a small number of hand-optimized DSP32C functions.

The lexical access component is using a reversed version of the A* N-Best search as described above and in [3]. So, rather than using Viterbi search to compute the best completion of partial paths and A* search to search forward, we use Viterbi search to find the best path from the beginning of any partial path and use A* search to find the best path from the end. This allows us to pipeline the Viterbi search with the incoming speech.

We are still in the process of optimizing the code on the DSP32C's, so we are not sure what the final configuration will be, but we are currently using one processor for data capture, one processor for input normalization, eight processors for the auditory model, two processors for some additional representations, one processor for the dendrogram, one processor for acoustic measurements, and two processors for phonetic classification. The current implementation computes these parts of the system in 2.3 times real time. When we combine lexical access on the same host the total processing time for VOYAGER is 5 times real time to completion.

With further optimization of DSP code, we believe that the processing through phonetic classification will run in real time in the present hardware configuration. When combined with lexical access, the entire system will run in approximately 3 times real time on a Sun4/330 and in approximately 2 times real time on a Sun 4/490.

## Evaluations

At the October 1989 DARPA meeting, we presented a number of evaluations of our initial version of VOYAGER [8] and we have used the same test set to measure the effects of the changes made since that time. To measure the effects of multiple sentence hypotheses, we allowed the system evaluated in [8] to produce the top N word sequences rather than the highest scoring word sequence. Its performance is plotted as a function of N in Figure 3. For each utterance, we

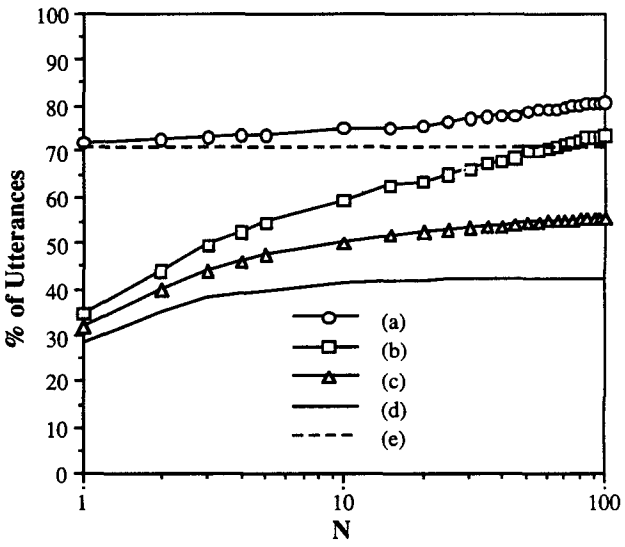


**Figure 3:** This figure shows the overall performance on the test set as a function of the number of word strings produced by the speech recognition component. Curve (d) shows the percentage of utterances where the correct word string is found. Curve (c) shows the percentage where the correct response is generated (see text for definition of "correct"). Curve (b) shows the percentage of utterances where VOYAGER produces any response. The horizontal line (e) shows the percentage of utterances where a response would have been produced if the correct word string had been found by the speech recognition component. Finally, curve (a) shows the percentage of utterances where *either* a response was produced from the top N word sequences from the recognition, *or* a response would have been produced given the correct word string.

took the highest scoring word string accepted by the natural language component of VOYAGER. The lower curve shows the percentage of these strings that are identical (after expanding contractions such as "what's") to the orthographic

transcription of the utterance. The next curve shows the percentage that produce the same action in VOYAGER as the action produced by the correct word string; these are the utterances that are "correct" at a functional level. The next curve shows the percentage of utterances that produced any response from VOYAGER. The difference between curve (c) and curve (b) indicates the number of incorrect responses (with "incorrect" meaning that the utterance produces a different response from the one that would have been produced with the correct word string). The remaining utterances, indicated by the area above curve (b), produce an "I'm sorry, I didn't understand you" response from VOYAGER. Of these remaining utterances, we found the number that would have produced a response if the system was given the correct word string. This is plotted as the difference between curves (b) and (a). The horizontal line (e) shows the percentage of utterances that produce an action given the correct word string. The difference between curves (a) and the horizontal line is the percentage of utterances that produce a response from VOYAGER when given the speech input but do not produce a response given the correct word string. These responses were judged either correct or incorrect by the system designers.

There are a number of things to learn from this figure. If we search deeper (either by increasing N or by incorporating the natural language constraints earlier in the search), we still increase the number of utterances that produce a correct response but at the expense of producing more incorrect responses. The difference between curves (a) and (b) shows the number of utterances that will produce a response if we can only find the correct word string with the search. So, this difference is the most that we can hope to gain by increasing the depth of the search (although this is not quite true since it is possible to find a word string that parses and produces the correct response even if the correct word string does not parse).

The previous results were computed using the perplexity 22 word pair grammar. As discussed previously, we have produced a word pair grammar with perplexity 73 that better matches the constraints of the natural language system. A comparison of these two sets of constraints can be seen in Figure 4. In this figure, we have plotted the upper three curves of Figure 3 for both the perplexity 22 grammar and the perplexity 73 grammar. It can be seen that while the perplexity 73 grammar has slightly lower performance, this degradation decreases as N increases above 10. We would hope that even with less constraint in the speech recognition component, the performance will be better than the tighter constraints as we search deeper. This should be true since the constraints match the natural language constraints much better.

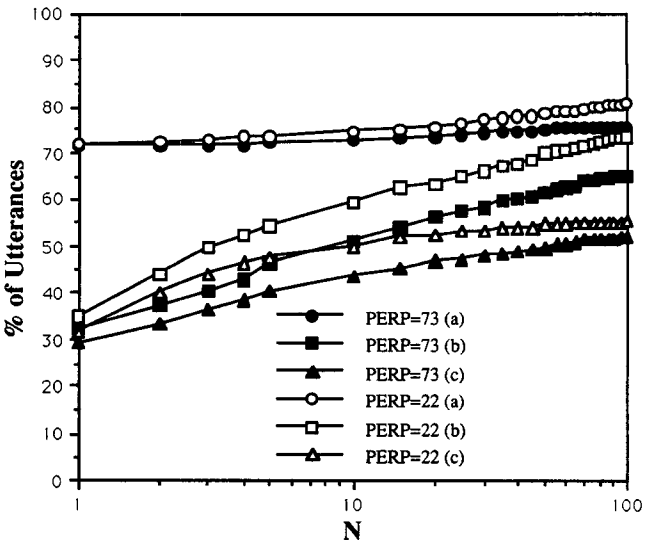


**Figure 4:** This figure shows the difference in performance for two different sets of speech recognition constraints. The curves are the same as the upper three curves in Figure 3 for perplexity=22 and perplexity=73.

## Summary/Future Plans

The evaluations show that compared to passing only the top scoring word string to the natural language system, the performance of the overall system is much improved by increasing the degree of integration of the speech recognition and natural language systems. However, the evaluations also show that there is not much to be gained in our system by increasing the depth of the search (either by increasing N in an N-Best search or by integrating the natural language constraints at an earlier stage of the search) since this will increase the number of incorrect responses faster than increasing the number of correct responses. What is needed are new sources of information for the search. Fortunately, our natural language system is capable of providing probabilities that we have not yet utilized. These probabilities have been shown to reduce the perplexity by at least a factor of three [9] and therefore should allow an increase in the depth of the search with a smaller number of incorrect responses.

We may also gain some performance by incorporating some form of explicit rejection criterion. Currently we reject an utterance based on the number of word strings that fail to produce a response (by choosing an upper bound on N in the N-Best search). If we used a more explicit rejection criterion (by taking into account the scores of the top N word strings for example) we may be able to decrease the ratio of incorrect response to correct responses.

There have been a number of developments in the speech recognition components that we intend to incorporate into the VOYAGER system. These are discussed in more detail in [7].

We would like to begin exploring dynamic adaptation of the natural language constraints. For example, we would like to increase the objects in VOYAGER's database to a much more complete set. In our current implementation, this would increase the perplexity of the speech recognition and result in poor performance. However, if we limit the vocabulary based on the discourse history, it is likely that we can make large increases in the size of the VOYAGER domain without

significant increases in perplexity.

Since we are interested in improving performance in the interactive use of the system, we have implemented a mechanism for automatically generating tasks for the user to solve with the help of the system [5]. This has allowed us to begin testing the system in a goal-directed mode and compare results obtained in such a mode to results obtained on data collected in a simulation mode.

## Acknowledgements

We would like to thank Dave Goddeau and Kirk Johnson for their help with the modifications made to VOYAGER described above.

## References

[1] Barr, A., E. Feigenbaum, and P. Cohen, *The Handbook of Artificial Intelligence*, 3 vols., William Kaufman Publishers, Los Altos, CA, 1981.

[2] Chow, Y, and R. Schwartz, "The N-Best Algorithm: An Efficient Procedure for Finding Top N Sentence Hypotheses", *Proc. DARPA Speech and Natural Language Workshop*, pp. 199-202, October, 1989.

[3] Soong, F., and E. Huang, "A Tree-Trellis Based Fast Search for Finding the N-best Sentence Hypotheses in Continuous Speech Recognition", these proceedings.

[4] Viterbi, A., "Error Bounds for Convolutional Codes and an Asymptotically Optimal Decoding Algorithm", *IEEE Trans. Inform. Theory Vol. IT-13*, pp. 260-269, April, 1967.

[5] Whitney, D. *Building a Paradigm to Elicit a Dialog with a Spoken Language System*, Bachelor Thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, 1990.

[6] Zue, V., J. Glass, D. Goodine, H. Leung, M. Phillips, J. Polifroni, and S. Seneff, "Preliminary ATIS Development at MIT", these proceedings.

[7] Zue, V., J. Glass, D. Goodine, H. Leung, M. Phillips, J. Polifroni, and S. Seneff, "Recent Progress on the SUMMIT System", these proceedings.

[8] Zue, V., N. Daly, J. Glass, D. Goodine, H. Leung, M. Phillips, J. Polifroni, S. Seneff, and M. Soclof, "The Collection and Preliminary Analysis of a Spontaneous Speech Database", *Proc. DARPA Speech and Natural Language Workshop*, pp. 126-134, October, 1989.

[9] Zue, V., J. Glass, D. Goodine, H. Leung, M. Phillips, J. Polifroni, and S. Seneff, "The VOYAGER Speech Understanding System: A Progress Report", *Proc. DARPA Speech and Natural Language Workshop*, pp. 51-59, October, 1989.

[10] Zue, V., J. Glass, M. Phillips, and S. Seneff, "The MIT SUMMIT Speech Recognition System: A Progress Report," *Proc. of DARPA Speech and Natural Language Workshop,* February, 1989.