COMPLEXITY REDUCTION IN A LARGE VOCABULARY SPEECH RECOGNIZER

Roberto Pieraccini, Chin-Hui Lee, Egidio Giachin†, Lawrence R. Rabiner

Speech Research Department
AT&T Bell Laboratories
Murray Hill, NJ 07974

ABSTRACT

This paper deals with the complexity issue of a large vocabulary high performance speech recognizer based on continuous density hidden Markov models. It has been demonstrated that large vocabulary continuous speech recognizers, based on hidden Markov models of phone-like units, achieve high word accuracy when highly detailed speech units are used for characterizing words in the vocabulary. In particular, when context dependent units are used to represent both intraword and interword phones, a word accuracy greater than 95% has been achieved on speaker-independent recognition of the standard DARPA Resource Management task when a word-pair grammar of perplexity 60 is used to constrain the search. At this level of algorithm complexity, every single detail of the implementation can have a major impact on the computation requirements. One way to reduce the complexity of the recognizer is to compile the whole network in such a way that the minimum number of memory accesses is required. Another substantial reduction of the complexity is obtained by using robust heuristics in the search, such as beam search pruning. Along with the beam search strategy we used a search strategy, called *guided search*, which speeds up performance assessment of the recognizer in the case where the test string identity is known. Particular care has been taken in making the algorithmic architecture and the data structures suitable for vectorization and concurrency.

1. Introduction

Most large vocabulary speech recognition systems are implemented as network searches for the best path through a large, but finite grid. The best path generally corresponds to the most likely sequence of words as constrained by a finite state network which implements the grammar or syntactic component of the system. When the number of basic speech (sub-word) units is small (i.e., on the order of 50-200 units), the details of implementation of the search strategy don't have a major impact on the overall complexity of the recognition task and experimental tuning and assessment of different training strategies and overall performance of the recognizer is relatively straightforward. However, when highly detailed (context dependent) speech units are used, including both intraword and interword context dependent units, the complexity of the overall implementation often increases quadratically with the number of basic units, and correspondingly the details of how the network search is implemented become of major importance in accessing the suitability of various network structures. When this is the case, a full search implementation of the speech recognition algorithm is totally impractical, if not impossible. For a task like the DARPA Resource Management Task (RMT), the number of grid points to be examined is on the order of tens of millions while the number of connections between grid points ranges in the hundreds of millions. An effective solution to this problem consists of performing an intelligent search through the grid and using reasonable but effective heuristics to eliminate unlikely path candidates from consideration. There are several ways for reducing the computational cost of a search for the most likely path through a finite but large grid of points. For the size of the task we refer to in this paper (1000 word vocabulary) a beam search [1] strategy is generally the one used. Although the algorithm is simple and straightforward to implement in theory, in practice there are a number of factors which strongly affect the overall efficiency of the implementation. It is the purpose of this paper to discuss these factors and show how proper design lead to an efficient and accurate implementation of a large vocabulary recognition system.

† Now with CSELT, Torino, Italy

The resulting recognizer structure has been designed to take advantage of the capabilities of a vectorized concurrent processor (an Alliant FX/2800) which consists of a cluster of up to 28 computing elements (CE's) that can execute code in vector concurrent mode. Of course, for taking advantage of the vector capabilities of each processor, the code must be structured so that the heart of the computation is performed in such a way that it can be distributed to the concurrent processors and can readily be performed in vector mode. Thus the algorithms must be implemented as a sequence of simple vector operations which are iteratively applied to a set of computing elements sharing the same data structure. Furthermore the size of the data structure, required by each CE for computation, must be small enough to fit within a local CE cache so that the inherent speed of the processor is not compromised by excessive memory faults outside the cache. In our implementation of the speech recognizer all the states are sequentially allocated in the same order as they appear in the HMM chains that represent the words. Given the simple HMM structure used to represent the units (i.e. three state left-to-right models without state jumps), the local path optimization in the Viterbi decoding has to be performed between states that are stored in consecutive locations of the state vector, except for those states that are at the boundaries of word segments. In this way, the decoding problem is split into two sub-problems, namely processing of internal word segment states and processing of word boundary states. The method of path extension, during the Viterbi decoding, involving boundary states that coincide with the beginning and end of a word model, is driven by the syntactic constraints used in the recognition system as well as by the phonological constraints imposed at the word junction level. The check of all the connection conditions (syntactic and phonological) for every word beginning and word ending state is very expensive computationally. Hence we use a compiled form of the list of possible connections between boundary states. This gives a significantly more efficient implementation of the recognition algorithm.

A final issue in the recognizer implementation concerns the use of a *guided search* algorithm which is used only for evaluation and assessment of different recognition and training strategies. When the spoken sentence is known, as is the case during the phase of development and performance evaluation of a speech recognition system, the beam search threshold can be based on the best path obtained through a forced alignment of the input speech with the HMM representation of the true sentence. This allows a further reduction of the search space, since the beam search threshold will be greatly reduced whenever the correct path is the same as the best path found by the forced alignment.

2. The Decoding Algorithm

The class of algorithms we are discussing in this paper is based on the assumption that the language used in the application may be expressed as a regular grammar. Although the algorithm may be modified in order to account for context-free representation of the language [2] we assume the regularity of the language in the rest of the paper. The grammar, or syntactic component of the system, defines all the possible ways of combining words in order to form a sentence. If the grammar is regular, it can be represented as a finite state network (FSN), hence in terms of arcs and nodes. Each arc corresponds to a word of the vocabulary. The vocabulary for the DARPA

CH2977-7/91/0000-0729 $1.00 © 1991 IEEE

RMT consists of 991 different words. A simple FSN that represents the so-called *no grammar* case consists of a network that allows any of the 991 words to follow any other word in the vocabulary, with optional pauses at the beginning and end of the sentence and between words. Silence, for the purpose of decoding, is treated as a vocabulary word represented by a single state HMM. A more constraining grammar (perplexity 60) is the one known as the *word pair grammar*, and consists of a finite list of words which can follow each word of the vocabulary. Of course, the word pair grammar is represented by a far more complex FSN than that used for the no grammar case. In this case the basic number of arcs representing words (i.e. 991) is still the same, but each word ending is connected to the set of possible word beginnings through a number of null arcs (referred to as *connections* in the rest of the paper) that is approximately given by the product of the perplexity and the number of word arcs ($60 \times 991 = 59{,}460$). Moreover, interword pauses must be taken into account, increasing the number of connections. An additional complication arises when interword units are used. Explicit modeling of interword coarticulation phenomena lead to a substantial increase in the accuracy of the speech recognizer [3,4,5,6] The assumption is made that coarticulation between two consecutive words affects only the boundary phones. The central portion of words, that is not supposed to be affected by the coarticulation (body) is represented by a linear sequence of context dependent phone like units (PLU's). The first (head) and the last (tail) phones of words are replaced by a list of possible context dependent PLU's that account for all the possible phonetic contexts at the beginning and at the end of the word. It should be noted that words composed of two phones and words composed of one phone are special cases of this kind of model. A two phone word does not have a body; hence all the elements of its head merge with all the elements of its tail. The concept of word head and tail cannot be extended to one phone words. Depending on the neighboring words, a single phone word consists of a particular context dependent interword unit. The connections between the tail of a word and the head of a possibly following word are based on a precomputed connection matrix CONN(ph_i,ph_j) whose generic element assumes the logic value true if unit ph_j may follow unit ph_i in the conjunction of two consecutive words. Fig.1 shows an example of the connections between two consecutive words, where a phone x in the left context y and right context z is represented as y_x_z and the symbol \$ refers to a generic context (i.e. no context dependency).

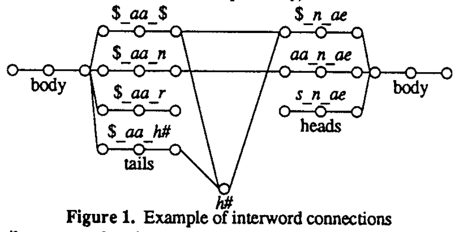

Figure 1. Example of interword connections

Since silence must be always put as an option between two consecutive words, a silence state is included in the connections. It should be noted that there may be multiple connections between two words, i.e. there is not a unique tail-head connection between two consecutive words and, moreover, a word tail may be connected to many different word heads and viceversa. For instance, in the example of Fig.1, silence is connected with two units: one of them (\$_*aa*_\$) can be followed by silence, while the other (\$_*aa*_*h#*) must be followed by silence (resented by the symbol *h#*). Hence it is clear how the introduction of an explicit model accounting for interword coarticulation greatly increases the complexity of the speech recognizer by increasing the number of connections between the words. The decoding (Viterbi decoding) is based on the explicit representation, through a graph (*decoding graph*), of all the the possible temporal evolutions of the acoustic patterns in terms of HMM states. The states of the decoding graph are obtained by substituting every arc of the FSN that represents the language with the corresponding word model in terms of subword unit HMM's. There are several kinds of state connections. There are connections (*structural connections*) inside word model segments (i.e. a bodies, tails, and heads), that result from the structure of the HMM's. In our case each state S_j at time t is connected only with the same state S_j at time t+1 and with the following state S_{j+1} at time t+1. Internal states (i.e. states that are not at the boundary of a word segment) are connected only through structural connections. The other two kinds of connections deal only with boundary states. There are connections among word model segments, i.e. between the heads of a word and its body, and between the body and the tails of a word (*inner connections*), that result from the structure of the word models, and there are connection among the tails of a word and the head of a possibly following word (*outer connections*). While the number of structural and inner connections is relatively small, the number of outer connections is generally very large and increases with the complexity of the language and with the complexity of the interword connections. Viterbi decoding is used for detecting, in the decoding graphs, the sequence of states that gives the maximum likelihood for the observation of the sequence of input patterns (the system we refer to in this paper uses a 38 dimensional input pattern, consisting of the cepstrum, delta cepstrum, delta-delta cepstrum, energy and delta energy, computed every 10 msec). The decoding equation for a graph node representing state S_i at time t can be written as:

$$G(i,t) = \max_{j=1,N_s} G(j,t-1) + l(\bar{x}_t \mid S_i)$$

where N_s is the overall number of states in the decoding graph, $G(i,t)$ is the accumulated score of the best path reaching state S_i at time t, $l(\bar{x}_t \mid S_i)$ is the local likelihood for state S_i given the observation vector $\bar{x}_t$ at time t (note that there are no transition probabilities in the equation since we assume equiprobable transitions in the HMM's). There are clearly two factors affecting the complexity of the recognizer. The first one is the number of combinatorics or the bookkeeping factor and it depends on the number of connections between states in the decoding graph. The second factor is the computation of the local likelihood. The amount of computation needed for computing the local likelihood depends on the kind of model we use and on the number of *different* states we have in the decoding graph. In the rest of the paper we refer to mixture density HMM's; hence the local likelihood is computed as:

$$l(\bar{x}_t \mid S_i) = \ln \sum_{m=1}^{M_i} w_{im}\, \mathrm{N}(\bar{x}_t, \bar{\mu}_{im}, \mathrm{C}_{im})$$

where $\mathrm{N}(\bar{x}_t, \bar{\mu}_{im}, \mathrm{C}_{im})$ represents a multivariate Gaussian density with mean vector $\bar{\mu}_{im}$ and diagonal covariance matrix C_{im}, and w_{im} are the mixture weights for state S_i (the maximum number, M_i, of mixture components is 16 in the experiments described in the rest of the paper).

The maximization operation shown in the decoding equation must be done for all the states in the decoding graph that may precede the current state S_i. Of course, if the state is an internal state, the regularity of the structural connections makes the bookkeeping very simple. For inner connections, the maximization is also quite straightforward, since, for every word model, we build a list of the boundary states of the body, of the heads and of the tails. The bookkeeping becomes more complicated for outer connections as both the linguistic constraints (e.g. word pairs) and coarticulation constraints (interword units) must be taken into account. Bookkeeping can be done by interpreting, at each time, the linguistic and coarticulation information. For instance, taking into account word pairs, for each of the 991 words we have to check whether each one of the possibly preceding 991 words is in the word pair list, resulting in 982,081 operations for every frame of input speech. Moreover, for each pair of words in the word pair list, all the possible tail-head connections must be checked using the CONN matrix. Interpreted decoding generally requires an enormous number of memory accesses, and therefore this leads to a poorly performing recognizer. A solution to this problem consists of compiling, in a single representation, both linguistic and coarticulation constraints, and generating, for each word boundary state, a list of all the word boundary states it is connected to by outer connections.

2.1 Allocation of Static and Dynamic Information

A vectorized compiled representation of the decoding graph is necessary for taking advantage of the parallel and vectorized architecture of the computer used for the recognition experiments. A single HMM state constitutes a data structure that is simple enough to be used as a basic element for vectorized processing. Since there is no difference in the processing of HMM states with regard to their position within the model (unless they are boundary states), the operations performed on single states can be easily expressed in vectorized form. Hence a vector is the ideal structure for storing the information related to the states in the decoding graph. Depending on the size of the task, the algorithm can be implemented with static or dynamic state allocation. In the static memory case, when the memory needed to allocate all the states of the decoding network is sufficiently small, each state is assigned an address within the state vector at the beginning of the program, and the address remains constant. When the number of states of the decoding network is very large, hence the amount of memory needed for static allocation is too big for practical implementation, a different solution is required to avoid memory faults within individual processors. The solution to this problem is to allocate memory only for those states that are active at any particular time, i.e. stack the "alive" nodes within a small memory stack. The address of a state within the state vector is therefore not predictable a priori. Hence a more sophisticated addressing scheme is needed to perform the decoding. This scheme, unfortunately, is not well suited to a parallel and vectorized implementation. The amount of memory needed in the DARPA resource management task, both in the no grammar case and with the word pair grammar, permits a static state allocation. However for more complex tasks (e.g. vocabularies of the order of 10,000 words) a dynamic allocation of states should be implemented. All the states of the decoding graph are stored sequentially in a *state vector* the elements of which correspond to HMM states in the word representation. Hence every element of the vector has to be identified as a state of a particular HMM. The information needed for this identification is the unit number (UNIT(i),$i=1,N_s$) and the state number (STATE(i),$i=1,N_s$) within the unit model it belongs to. An additional vector, called A(i), is used to control the transition between consecutive states. Since we do not use transition probabilities in the likelihood computation, A(i) can be either 0 or $-\infty$, depending on whether the transition from that state to the next state in the vector is allowed or not. $A(i)$ is used just to prevent the propagation of the score from the last state of a piece of a word model (head, tail or body) to the first state of the following part of the word model in the vector, since the score propagation among different segments must be fully controlled by the inner and outer connection information. The use of the $A(i)$ vector eliminates the need for checking at each decoding step whether the state is an internal or boundary state, allowing the decoding to be performed as an uninterrupted flow of vectorial operations. More information must be stored for handling both inner and outer connections, like the number of heads and tails of each word, the address, within the state vector, of the boundary states of each body, head, and tail, and the list of outer connections for each tail of each word. The dynamic information related to active states in the state vector must be stored at each step of the decoding algorithm. This information includes: the current score (SCORE), the pointer to the previous lexical item (BPO) on the best path reaching that state at the current time, and a time marker (BEG) indicating when the current best path entered into the current lexical item. Due to the Markovian property of the models, the decoding process needs only the score and the pointers relative to the last processed frame. Hence the three arrays are doubled in size, the OLD version of each array is relative to the previously processed frame, while the NEW version is relative to the current frame. At the end of the processing for the current frame, the pointers NEW and OLD are flipped. In order to be able to backtrack the best path from the last frame to the beginning of the sentence and decode the recognized sequence of words, we have to store the back pointers and the time markers along the whole decoding process. The amount of memory needed for keeping this information is not negligible as this information must be recorded for every arc of the FSN and for every frame of the decoded sentence. A possible solution to reduce the amount of memory for the backtracking information consists of implementing a partial backtracking strategy [7,8] In the partial backtracking, the backpointers are checked during the decoding in order to find some past node that is the only *ancestor* of all the currently active nodes *(immortal node)*. Hence a partial section of the global optimal path can be tracked back from the current immortal node to a previously detected immortal node, and all the backtracking information in the time segment between the two immortal nodes can be deleted, making memory available for new data. The partial backtracking strategy is advisable for a real time, continuously running, implementation of the decoding algorithm, where we do not know in advance the maximum duration of sentences. Since in the version of the system used for speech recognizer performance evaluation we know the maximum duration of any sentence and the memory needed for the backtracking information is within the capability of the computers we use, the partial backtracking strategy was not used in this version of the recognizer.

2.2 Organization of the Beam Search

As stated in the introduction, beam search is a powerful heuristic that allows the search to focus around the locally best path and to disregard unlikely solutions. With a proper setting of the beam search threshold, the probability of losing the globally best path can be made small enough so that the performance with a finite beam approaches the performance of a full search algorithm. For a beam search to be efficient, the bookkeeping must be performed in such a way to avoid computation on nodes that have been pruned in previous stages of the search. This generally implies a reorganization of the search based on a list of currently active nodes LST_A(i), $i=1,N_{act}$ that must be updated at the end of the decoding of each single frame of input speech. In this way, for a decoding step, only N_{act} nodes are explored. As one of the most computationally expensive parts of the decoding involves the propagation of score through outer connections, an additional reorganization of the boundary states was performed by updating, at each time frame, a list of active words LST_W(i), $i=1,N_{wact}$, i.e. those words that have at least one tail end-state active, among their tail segments. Hence only outer connections originating from active words are used in the score propagation. For enhancing the parallel structure of the algorithm, the entire decoding process was broken down into 5 sequential steps. A first module performs the dynamic programming optimization for all the active states in the state vector. A second module performs path expansion at inner connections while a third module takes care of the outer connections. A fourth module performs the beam search pruning and updates both LST_A and LST_W. Finally, the last module, computes the local likelihood for each active state and adds it to the corresponding score. The code for the five modules has been optimized both for vectorization and concurrency. The only stage that cannot be fully executed in concurrency is the list updating, due to the sequential nature of the list structures. In the current version of the recognizer, partial lists are concurrently updated at each processor, and eventually combined into a single list.

3. Guided Search

We developed a particularly efficient version of the recognizer suited only for experimental assessment of speech recognition accuracy. When assessing performance on a test database, the correct string of words (i.e. the word string actually spoken) is known a priori for every sentence. The forced alignment of the test speech, with the network representing the actually spoken sentence, produces a path (*correct path*) whose frame-by-frame score can be used to further reduce the size of the search space. The concept behind the guided search is that the globally best path will have a final score that cannot be inferior to the global score of the correct path. This condition, of course, holds only for the global scores and not for the partial scores along the globally best path and the correct path. It may happen that the globally best path drops below the score of the correct path at a certain point in the search (*path inversion*), eventually attaining a better score later in the search. However, the locally best score obtained during the forced alignment may be used to estimate a beam search threshold that reduces, to a minimum, the possibility of missing the globally best path during a path inversion. Although the parameters of the guided search were carefully tuned, estimation of the recognition performance is generally slightly biased towards a smaller error rate, as shown in the following section. However, guided search can still be used to speed up the experimental tuning of a system.

4. Performance of the Recognizer

Timing experiments have been performed during the development of the algorithm to assess the efficiency of the entire speech recognition system. All the performance scores reported in this section were obtained during the recognition of several sentences using a phone set of 1769 subword units with word pair constraints. The guided search strategy was used in all the experiments. The experiments were run on an Alliant FX/2800 using a cluster of 6 processors working in concurrency. Table 1 shows the average time (in seconds) per sentence, and average time per decoded frame, in 3 different versions of the recognizer. In REC1 all the outer connections are explored at every frame, in REC2 only connections originating from active words are explored, and REC3 has the same features as REC2, but uses a compiled version of the outer connections.

Recognizer	Time per sentence	Time per frame
REC1	222	0.7
REC2	130	0.4
REC3	25	0.08

TABLE 1. Average time (in seconds) per sentence (TPS) and per frame (TPF) in three different implementations of the recognizer

The difference in computing time between REC1 and REC3 is almost one order of magnitude. The complexity is almost halved by the introduction of the list of active words, while the compilation of the outer connections reduces by a factor of 5 the complexity. Table 2 shows the time breakdown for the five decoding modules described above when REC3 is used. The numbers shown are the percentage of time spent in each module during the decoding of one frame.

Operation	Time %
Internal states	14.8
Inner connections	11.8
Outer connections	5.0
List updating	13.2
Local likelihood	55.2

TABLE 2. Percentage of time spent in each module during the decoding of one frame

The table shows that the local likelihood computation accounts for more than 55% of the total decoding time and it is followed by the dynamic programming optimization on the active states, the list updating, and the propagation of scores for inner connections. The propagation of scores to outer connections takes only 5% of the entire computation. In fact, even though the number of potential connections is very large, only a small fraction of them are actually explored at each frame.

Fig. 2 shows the efficiency of the whole system (REC3) in terms of concurrency. The figure shows the average decoding time per frame as a function of the number of computing elements used to execute the code. The performance shown by the solid line is that obtained with the recognizer REC3, while the dotted line is the theoretical curve $\frac{1}{N}$. The figure shows that the code performance is very close to that of fully concurrent code.

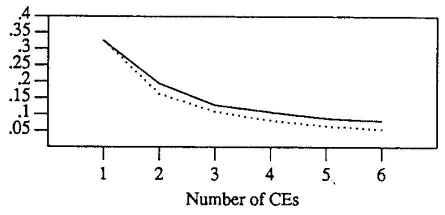

Figure 2. CPU time per frame (seconds) versus number of CEs in REC3 (solid line) and in the theoretical case (dotted line).

Finally Table 3 shows the word accuracy for different values of the beam search threshold and when using guided search (GS), along with the average percentage of search space that is explored (the search space is 49,210 states for one frame of speech), and the average CPU time for decoding one frame of speech.

Using a guided search the complexity is reduced more than three times, with a small bias in the word accuracy estimation. In a real recognition situation, when guided search cannot be used, the computing time, with this configuration of the recognizer, rises up to 0.25 sec per frame, corresponding to about 80 sec average per sentence and 25 times real time.

Beam Search Threshold	Word Accuracy	seconds per frame	percent of search space
GS	94.9	0.08	0.8
150	93.0	0.16	3.2
175	94.1	0.21	4.7
200	94.4	0.25	6.6
225	94.4	0.31	8.8
250	94.4	0.37	11.3
400	94.4	0.79	30.0

TABLE 3. Performance with different beam search thresholds and with guided search

5. Conclusions

This paper provides a detailed description of all aspects of the implementation of a large vocabulary speaker independent, continuous speech recognizer which is used as a tool for the development of recognition algorithms based on hidden Markov models and Viterbi decoding. The complexity of HMM recognizers is greatly increased by the introduction of detailed context dependent units for representing interword coarticulation. A vectorized representation of the data structures involved in the decoding process, along with compilation of the connection information among temporally consecutive words and an efficient implementation of the beam search pruning, has led to a speed up of the algorithm of about one order of magnitude. A guided search can be used during a tuning phase for obtaining a speed up of more than three times. An average recognition time of about 25 seconds per sentence (on the computer configuration used in the experiments), although far from real time, allows us to perform a series of training experiments and to tune the recognition system parameters in order to obtain high word accuracy on complex recognition tasks such as the DARPA resource management.

REFERENCES

[1] B. Lowerre, D. R. Reddy, "The HARPY speech understanding system," in *Trends in Speech Recognition* (Lea, W. ed.), 340-346. Prentice-Hall Inc., New York.

[2] H. Ney, "Dynamic Programming Speech Recognition Using a Context-free Grammar," *Proc, ICASSP 87*, pp. 69-72, Dallas, Texas, April 1987.

[3] C. H. Lee, E. Giachin, L. R. Rabiner, R. Pieraccini, and A. E. Rosenberg, "Improved acoustic modeling for continuous speech recognition," *Proc. DARPA Speech and Natural Language Workshop*, Somerset, PA, June 1990.

[4] X. Huang, F. Alleva, S. Hayamizu, H. W. Hon, M. Y. Hwang, K. F. Lee, "Improved Hidden Markov Modeling for Speaker-Independent Continuous Speech Recognition," *Proc. DARPA Speech and Natural Language Workshop*, Somerset, PA, June 1990.

[5] H. Murveit, M. Weintraub, M. Cohen, "Training Set Issues in SRI's DECIPHER Speech Recognition System," *Proc. DARPA Speech and Natural Language Workshop*, Somerset, PA, June 1990.

[6] D. B., Paul "The Lincoln Tied-Mixture HMM Continuous Speech Recognizer," *Proc. DARPA Speech and Natural Language Workshop*, Somerset, PA, June 1990.

[7] J. C. Spohrer, P. F. Brown, P. H. Hochschild, and J. K. Baker, "Partial traceback in continuous speech recognition," *Proc. IEEE Int Cong. Cybernetics and Society*, Boston (MA), 1980.

[8] M. Cravero, L. Fissore, R. Pieraccini, C. Scagliola, "Syntax driven recognition of connected words by Markov models," *Proc. of ICASSP 1984*, San Diego, (CA), 1984.