IMPLEMENTATION ASPECTS OF LARGE VOCABULARY RECOGNITION BASED ON INTRAWORD AND INTERWORD PHONETIC UNITS

R. Pieraccini, C. H. Lee, E. Giachin† , L. R. Rabiner

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

ABSTRACT

Most large vocabulary speech recognition systems essentially consist of a training algorithm and a recognition structure which is essentially a search for the best path through a rather large decoding network. Although the performance of the recognizer is crucially tied to the details of the training procedure, it is absolutely essential that the recognition structure be efficient in terms of computation and memory, and accurate in terms of actually determining the best path through the lattice, so that a wide range of training (sub-word unit creation) strategies can be efficiently evaluated in a reasonable time period. We have considered an architecture in which we incorporate several well known procedures (beam search, compiled network, etc.) with some new ideas (stacks of active network nodes, likelihood computation on demand, guided search, etc.) to implement a search procedure which maintains the accuracy of the full search but which can decode a single sentence in about one minute of computing time (about 20 times real time) on a vectorized, concurrent processor. The ways in which we have realized this significant computational reduction are described in this paper.

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. We have especially found this to be the case when we use context dependent interword speech units where the fan-out at the end of a word and the fan-in at the beginning of the next word are both very high, and the bookkeeping to keep track of all possible phonetic contexts in which a word occurs can dominate the computation (to the extent that it can make even the likelihood computation with mixture density HMM's be negligible compared to the bookkeeping overhead), since at *every frame*, all possible connections between every valid pair of words in the grammar must be examined. 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, 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, including different versions of stack decoding[1] [2], and A^* algorithms[3] [4].

Perhaps the most widely used search algorithm for large vocabulary speech recognition is the so called *beam search* algorithm introduced first by Bruce Lowerre in the Harpy system[5]. In its original form the algorithm carries out the search by exploring, at every step, only a fixed number, N, of best scoring paths (beam). The

† Now with CSELT, Torino, Italy.

assumption behind this heuristic is that the globally optimal path is very unlikely to lie outside of the N best paths at each stage during the search. The implementation of this form of beam search presents some computational problems in finding the N best paths, since all explored paths must be sorted according to their likelihoods. An approximation to the original beam search consists in finding the current best scoring path among the active ones and extending only those paths for which the difference between their score and the best score is less than a given threshold (the so called beam width). In this case the number of preserved (active) paths is variable during the search and depends on the distribution of the likelihoods among the paths themselves. Again, a sufficiently large value of the threshold can make the probability of missing the globally optimal path sufficiently small. An advantage of the beam search strategy, over other strategies, is that being a breadth first search it can be conveniently implemented in a real time fashion, in the sense that the processing can proceed frame synchronously. Also, since all active paths at every step of the search have the same length, the scores of all active paths can be compared on an equal basis.

For the reasons described above, a beam strategy is generally the one used in large vocabulary continuous speech recognition systems. 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.

The resulting recognizer structure has been designed to take advantage of the capabilities of a vectorized concurrent processor (an Alliant FX/80) which consists of a cluster of up to 8 computing elements (CE's) that can execute code in vector concurrent mode. The iterations of a loop within a program can execute concurrently provided that there is no data dependency within the loop itself, i.e. operations in one iteration don't use and are independent of results obtained in other iterations. Moreover, since every CE works as an array processor that pipelines operations performed on data vectors, vector operations, instead of single scalar operations, are actually distributed across processors for concurrent execution. 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.

Based on the above discussion it should be clear that a single HMM state constitutes a data structure that is simple enough simple 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 (with the exception of word initial or word final 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 network. 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.

In our implementation of the DARPA resource management task 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, 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. Since the word models used in the recognition system have a number of possible different beginning and end segments (heads and tails) represented by individual interword context dependent phones, and since the language may be represented by a finite state network with word pair constraints, the connection scheme between the boundary states may be fairly complex. The check of all the connection conditions (syntactic and phonological) for every word beginning and word ending state is thus 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 uttered 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. This occurs often during long content words, and rarely during short function words. Overall it leads to a useful reduction in computation.

2. REVIEW OF THE RECOGNIZER STRUCTURE

A more detailed description of the large vocabulary continuous speech recognition system implemented by the algorithms described in this paper can be found in Refs[6] [7]. Three state, continuous density, HMM's are used to represent each of the units (except silence which is represented by a single state HMM). There are only two transitions leaving from a state, a self loop transition and a transition to the next state. Transition probabilities are not used during the likelihood computation, since it has been experimentally shown that they don't affect the recognition performance.

Within each state S_i^k of each model, the spectral observation probability density is represented by a weighted mixture of M_i^k multivariate normal density functions. In addition, each state has associated with it a log energy histogram $E_i^k(\varepsilon)$ representing the logarithm of the probability of observing a frame with log-energy ε while in state S_i^k.

3. STRUCTURE OF THE LEXICON

When using intraword and interword units, every word in the lexicon is represented by three distinct segments, namely a *head* a *body* and a *tail*. The head and the tail take into account all the possible types of coarticulation with adjacent words, according to the unit set chosen for representing the vocabulary. Hence the head and tail of a particular word consist of a collection of all the possible conjuction units at the beginning and at the end of a word. The body of each word is a simple linear sequence of units and is assumed to be independent of the neighboring words. 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 possible heads merge with all the possible tails.

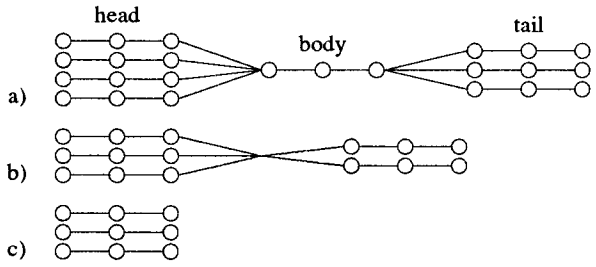

Figure 1. Examples of word models.

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 inter-word unit. Hence a one phone word, in all its possible contexts, is represented by a collection of all the inter-word units whose central phone corresponds to the word itself. Fig.1 shows an example of a word model for three cases; namely for a word consisting of more than 2 phones (a), for a 2 phone word (b), and for a single phone word (c). (the self transition of HMM states has not been drawn in the figure). Silence at the beginning and at the end of a sentence is represented as a single phone word.

The connections between the tails of a word and the heads of the following word are based on a precomputed, syntax dependent 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. The matrix $CONN(ph_i,ph_j)$ may be set according to phonological rules[8]

4. THE GRAMMAR COMPONENT

The sequence of words defining legal sentences within the task grammar are expressed through a finite state network (FSN) and a list of permitted word pairs. In the FSN used in the experiments, the entire vocabulary is divided into four independent sets, with no overlap.

These sets consist of the following: $B\bar{E}$ which includes the words that can be spoken at the beginning of a sentence but not at the end; $\bar{B}E$ which includes the words that can be spoken at the end of a sentence but not at the beginning; BE which includes the words that can be spoken either at the beginning or end of a sentence; and $\bar{B}\bar{E}$, which includes the words that cannot be spoken either at the beginning or at the end of a sentence. Language constraints, in the form of word occurrence statistics such as bigrams or word pairs, can be used in the recognition algorithm. The word pair grammar, specifying the words that are allowed to follow a given word, has a perplexity of about 60 for the resource management task. The language perplexity is equal to the number of different words (i.e. 991) when no word constraints are used.

5. STATIC INFORMATION USED IN THE DECODER

The decoding (recognition) algorithm uses a network (decoding network) that integrates acoustic, phonological, lexical, and syntactical models, in order to find the sequence of words that gives the best interpretation of the input sentence in terms of likelihood. The decoding network is obtained by substituting the corresponding lexical models along every arc of the FSN that represents the language. Of course, if interword units are used, the connections between words at the FSN nodes must be made according to the phonological rules defined for those units. The use of a word pair grammar greatly increases the complexity of the decoding network since new nodes must be added to the original FSN in order to allow only the valid sequences of word pairs. Thus, even a simple FSN can become very large when interword units and word pairs are taken into account.

Generally speaking, one can always trade off memory for computing time. In this particular case the network does not have to be completely compiled (i.e. a detailed list made of all nodes and arcs) but, for instance, word pair constraints can be taken into account by saving the run time decoding information associated with every arc of the network which joins at a given node of the original FSN at a given frame, and using this information to check the word pair constraints before extending new paths out of the same node. This procedure, that we call *interpreted decoding* as opposed to *compiled decoding*, has been used in preliminary versions of the algorithm. The interpreted decoding leads to a highly inefficient implementation in terms of computing time, to decode a given string. This is because the basic operation for decoding, namely checking matches for word pair constraints, cannot generally be coded in a form that can be efficiently parallelized and vectorized. Thus, as stated in the introduction of this paper, a vectorized compiled representation of the decoding networks is necessary for taking advantage of the parallel and vectorized architecture of the computer used for the recognition experiments. An even better structure for the algorithm is a network which is represented by two different levels of information, namely *lexical* information, encoded by the sequence of HMM states that form word bodies, heads and tails, and the *connection* information that is needed for propagating the score among bodies, heads and tails during the decoding procedure. The lexical information is encoded into a *state vector*. Every element of the state vector corresponds to a particular HMM state 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 ($UNIT(i)$) and the state number ($STATE(i)$). 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 solely used 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 connection information. Hence the last state of every piece of word model has $A(i)=-\infty$ while all other states have $A(i)=0$. The location of items within the vector is stored in auxiliary (directory) arrays. Thus all beginning and end states of every segment may be directly accessed.

There are two different types of connection information, namely inner connections, i.e. the ones among heads, body and tails of the same word, and outer connections, between tails and heads of temporally consecutive word hypotheses. Outer connection information, conveying phonological (i.e. the matrix $CONN(ph_i,ph_j)$) and linguistic (i.e. the word pair grammar) constraints, can be stored either in an interpretable form or in a compiled form. It is obvious that compiled connection information leads to a far more efficient version of the decoder than one running on interpretable connections. In the interpretable version the decoder has to check, at every frame and for every word head, all the arcs corresponding to legal preceding words (i.e. according to the word pair grammar), and for each one of them find, among all the tails, those that are connectable to the head, each time checking the $CONN(ph_i,ph_j)$ matrix. In a compiled version all the pointers to the legal tail ends are pre-stored in an array (*connection list*) addressable by the head identifier. Table 1 shows statistics on the number of heads and tails, and the overall number of states and outer connections in two different cases, namely using 1172 and 1769 context dependent units.

The state vector has been purged from all head and tail segments that have no outer connections (not all connections among the words are possible due to the word pair constraints). The number of heads and tails in the table is computed after the purge operation.

5.1 Inter word silence implementation

There are two distinct sets of units that may be followed by interword silence when they are used as word tails. The first set are those units that must always be followed by silence, i.e. they are constrained to have silence as the right context. In order to avoid problems when using a word pair grammar, these units are realized by appending the silence model (consisting of a one state HMM) to the specific unit model and considering the sequence of the two models as a single model.

Units	States	Heads	Tails	Connections
1172	41636	6860	2785	240024
1769	47641	8082	3486	274863

TABLE 1. Statistics on static information with two different unit sets

The second set includes all those units that are followed by an optional silence. Again, for ease of implementation, these units are duplicated when they appear in the tail of a word and a silence model is appended to one of the two instantiations.

6. DYNAMIC INFORMATION IN THE DECODER

The 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 in implementing a partial backtracking strategy[9], [10]. 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, we didn't implement the partial backtracking strategy.

For a given frame and a given arc, the backtracking information has to be recorded when a unique interpretation of the back-pointers is available for that arc. Since arcs have head segments joining at the first state of the body segment and each head segment may have a different back pointer at a given stage of the decoding, the only place where the information relative to the previous arc is unique is along the body segment. Thus the backpointers of the first state of the body segment of each arc are recorded at each time frame. For two phone words, that don't have a body segment, the recorded backpointers are those relative to the best path among those exiting from the last state of every head segment.

7. THE FINAL DECODING ALGORITHM

The diagram in Fig.2 shows the breakdown of the decoding process for one frame of input speech into functional blocks. In the remainder of this section we analyze the main implementation characteristics of each block.

7.1 Internal states

This block performs the dynamic programming optimization for all the active states in the state vector. The pointers to active states are kept in a list ($LIST(i)$, $i=1,N_{active}$) that is updated at the end of the processing of each frame. A state is active if its score, at the previous frame, is better than the beam-search pruning threshold defined for that frame. Since the HMM structure we use in our system has only two possible transitions from each state, namely the self loop and the forward transition, and since we don't use transition probabilities in the likelihood computation, the basic dynamic programming optimization consists of the comparison of the score of each state with the score of

the previous state. In order to inhibit the propagation of the score between consecutively stored word segments, the constant $A(i)$ is added to the score of state i before comparison with state $i+1$. The backpointers $BPO(i,NEW)$ and the time markers $BEG(i,NEW)$ are updated according to the result of the maximization operation. The operation performed in this block is completely parallelized and vectorized.

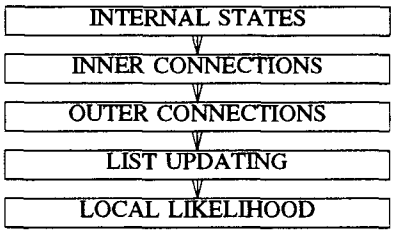

Figure 2. Steps in the decoding of one frame

7.2 Inner connections

Inner connections are those among the heads, the body and the tails of the same word. Only words with a number of phones greater than or equal to 2 have inner connections. In order to keep efficiency of the implementation high, arcs have been ordered according to the number of phones of corresponding words. Hence, in our implementation, the arcs whose order number goes from 1 to $N_{arcsgt2}$ correspond to words with more than two phones; the arcs from $N_{arcsgt2}+1$ to N_{arcs2} correspond to words with two phones and the arcs from $N_{arcs2}+1$ to N_{arcs1} correspond to words composed of only one phone. This block performs three basic functions:

1. For arcs 1 to $N_{arcsgt2}$. Finds the best scoring end state among all the possible heads of the word and propagates the corresponding path to the first state of the body if it has a greater score. This is a vectorized and parallelized operation.
2. For arcs $N_{arcsgt2}+1$ to N_{arcs2}. Finds the best scoring end state among all the possible heads of the word and propagates the corresponding path to the first state of each tail, if it has a greater score. This is a vectorized and parallelized operation.
3. For arcs 1 to $N_{arcsgt2}$. Propagates the path corresponding to the last state of the body to the first state of each tail of the word, if it has a greater score. This is a parallelized operation.

7.3 Outer connections

Outer connections are those among the tails of a word and the heads of another word. This operation is performed only for active arcs. An arc is considered active when at least one tail end state is active. The pointers to active arcs are kept in a list $LST\_C(i)$, $i=1,N_{active\_c}$ that is updated at the end of the processing of each frame. In the most efficient implementation the connections are compiled into a vector. The elements of this vector are the locations, within the state vector, of the states that are connected to a given word tail.

7.4 List updating

The beam search pruning of states is performed in this block. In the standard implementation, at the $i-th$ frame, a pruning threshold Θ_i is set as:

$$\Theta_i = \Lambda^{max_i} - \Delta$$

where Λ^{max_i} is the maximum likelihood among the active states at the $i-th$ frame and Δ is a fixed constant. Hence the likelihood of all states that are active at the $i-th$ frame is compared with Θ_i. A state is then included in the new list if its likelihood is greater than the threshold Θ_i. The list updating operation is computationally demanding due to the sequential nature of the operations to be performed. There are five sets of states that are potentially active at the $i-th$ frame. They are all the states that were active at the previous frame, all the successors of the states that were active at the previous frame, all body initial states, all head initial states, and all tail initial states. For computing the maximum state likelihood Λ^{max_i}, and for the subsequent state pruning and list updating, it is necessary to check all five sets of states. This produces a computational overhead for the intersection of the different sets of states (e.g. a state in the first set may also be in the second set; hence the checking operation is performed twice for that state) which leads to an inefficient implementation. A solution is to keep an additional vector $LIVE(i)$ whose generic $j-th$ element is set to true any time the $j-th$ state in the state vector is set to a new likelihood value during the decoding (all the elements of $LIVE(i)$ are set to false before the decoding of each frame). To further improve the implementation of this block, the list updating operation is performed in a concurrent mode, first generating partial lists of active states, and then merging the partial lists into the final list. Moreover, whenever a state is put into the active state list, the corresponding phonetic unit is marked as active setting to true the corresponding location of a vector $USE(i)$, $i=1,...,N_{units}$. This is done in order to restrict the local likelihood computation only to active units.

An additional operation is performed by this block and consists in updating the list of active arcs. An arc is considered to be active, for the purpose of propagating its score through outer connections, if at least a tail final state is active. Again, in the parallel implementation, partial active arc lists are computed first, and then merged into the final active arc list.

7.5 Local likelihood

While local likelihood computation in discrete density HMM's is a simple table lookup, with mixture density HMM's it becomes one of the major computational loads of the entire decoding procedure. A particularly optimized version of the state local likelihood computation has been implemented, enhancing the vectorized structure of the computation. Also, the local likelihood is computed only for active units. i.e. when the value of $USE(i)$ is set to true.

7.6 Guided search

We have also 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 that has to be recognized is known a priori for every uttered sentence. The forced alignment of the test speech, with the network representing the actually uttered sentence, produces a path whose frame-by-frame score (self score) may be used to further reduce the size of the search space. The concept behind the guided search is that the overall best path will have a final score that cannot be inferior to the final score of the forced alignment procedure. This is not true for the local score along the overall best path and the forced alignment path. It may happen that the overall best path drops below the score of the forced alignment path at a certain point in the search, eventually attaining a better score later in the search. Moreover, if the guided search is performed in a frame synchronous fashion, i.e. the forced alignment is carried out frame synchronously with the recognition, we actually don't know which is the best path in the forced alignment. A non frame synchronous version that first performs the forced alignment, then performs a backtracking along the best path and saves all the local score values, would require too much memory to store the backtracking information during the alignment phase. Thus, only the score of the locally (not globally) best path is available during the frame synchronous alignment. The threshold is computed by decrementing the score of the locally best path by a fixed amount in order to take into account the above mentioned sources of error.

7.7 Timing experiments

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 1172 sub-word units. The maximum number of mixture components per state was 16, while the dimension of the observation vector was 24. The guided search strategy was used in all the experiments.

Table 3 shows the average time (in seconds) per sentence (TPS), and average time per decoded frame (TPF), in 3 different versions of the recognizer. In REC1 all the connections are explored at every frame, in REC2 only connections coming from active arcs are explored, and REC3 has the same features of REC2, but uses a compiled version of the connection list.

RECOGNIZER	TPS	TPF
REC1	555	1.8
REC2	326	1.1
REC3	65	0.2

TABLE 2. Average time (in seconds) per sentence (TPS) and per frame (TPF) in three different implementations of the recognizer

Table 4 shows the time breakdown for the five modules of Fig.2 when REC3 is used. The numbers shown are the percentages 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 3. 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 score to outer connections takes only 5% of the whole computation. In fact, even though the number of potential connections is very large (240024 in the experiment), only a small fraction of them are actually used at each frame.

Finally Fig. 3 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.

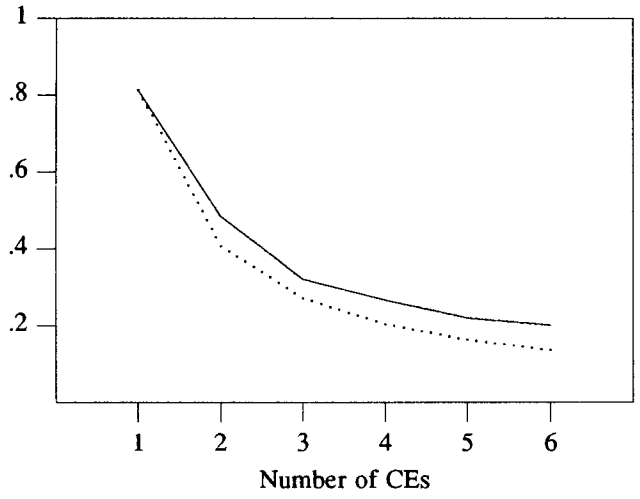

Figure 3. CPU time per frame (seconds) versus number of CEs in REC3 (solid line) and in the theoretical case (dotted line).

8. CONCLUSIONS

This paper provided a detailed presentation of all aspects of the implementation of a large vocabulary speaker independent continuous speech recognizer to be 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, has led to a speed up of the algorithm of about one order of magnitude. An average recognition time of about one minute 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 performance recognition on complex tasks such as the DARPA resource management.

9. ACKNOWLEDGEMENTS

The authors gratefully acknowledge the helpful advice and consultation provided by Fil Alleva of Carnegie Mellon University on the implementation details of the SPHINX system and Douglas B. Paul of Lincoln Laboratories, MIT, for the guided search idea.

REFERENCES

1. Jelinek, F. (1969). A fast sequential decoding algorithm using a stack. *IBM J. Res. Develop.*, vol. 13, pp 675-685, Nov. 1969

2. Schwartz, R. and Chow, Y. L. (1990). The N-best algorithm: an efficient and exact procedure for finding the N most likely sentence hypotheses. *Proc. ICASSP 90*, pp. 81-94, Albuquerque, NM, April 1990.

3. Nilsson, N. J. (1980). Principles of artificial intelligence. Tioga Publishing Co., Palo Alto, CA.

4. Huang, E. F., Soong, F. K. (1990). A fast tree-trellis search for finding the N-best sentence hypotheses in continuous speech recognition. *J. Acoust. Soc. Am.* suppl. 1, vol. 87, S105, Spring, 1990, also in *Proc. DARPA Speech and Natural Language Workshop*, Somerset, PA, June 1990.

5. Lowerre, B. and Reddy, D. R. (1980) The HARPY speech understanding system. In *Trends in Speech Recognition* (Lee, W. ed.), 340-346. Prentice-Hall Inc., New York.

6. Lee, C. H., Rabiner, L. R., Pieraccini, R., Wilpon, J. G. (1990). Acoustic modeling for large vocabulary speech recognition. *Computer Speech and Language*, 4, pp. 127-165, 1990

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

8. Giachin, E. P., Rosenberg, A. E., Lee, C. H. (1990). Word juncture modeling using phonological rules for HMM-based continuous speech recognition. *Proc. ICASSP 90*, pp. 737-740, Albuquerque, NM, April 1990.

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

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