A Tree-Trellis Based Fast Search for Finding the N Best Sentence Hypotheses in Continuous Speech Recognition

Frank K. Soong
*Eng-Fong Huang**
AT&T Bell Laboratories
Murray Hill, New Jersey 07974

ABSTRACT

In this paper a new, tree-trellis based fast search for finding the N best sentence hypotheses in continuous speech recognition is proposed. The search consists of two parts: a forward, time-synchronous, trellis search and a backward, time asynchronous, tree search. In the first module the well known Viterbi algorithm is used for finding the best hypothesis and for preparing a map of all partial paths scores time synchronously. In the second module a tree search is used to grow partial paths backward and time asynchronously. Each partial path in the backward tree search is rank ordered in a stack by the corresponding full path score, which is computed by adding the partial path score with the best possible score of the remaining path obtained from the trellis path map. In each path growing cycle, the current best partial path, which is at the top of the stack, is extended by one arc (word). The new tree-trellis search is different from the traditional time synchronous Viterbi search in its ability for finding not just the best but the N-best paths of different word content. The new search is also different from the A^* algorithm, or the stack algorithm, in its capability for providing an exact, full path score estimate of any given partial (i.e., incomplete) path before its completion. When compared with the best candidate Viterbi search, the search complexities for finding the N-best strings are rather low, i.e., only a fraction more computation is needed.

I. Introduction

In a spoken language understanding (recognition) system, the search space of possible sentence hypotheses are determined by many factors such as the size of recognition vocabulary, the rigidity of grammar, the system dependency on specific speakers, etc. When these constraining factors are relaxed, the space can be very large and the effort for finding a global optimal hypothesis may become expensive or even prohibitive. Sometimes, even for a smaller scale problem, certain language constraints can not be easily incorporated into a low level, acoustic search. For example, a check-sum grammar used for detecting error in a digit string, due to its nonlinear and modulo arithmetic nature, can not be built into a continuous digit recognizer except for some trivial cases.

To reduce the search effort, a spoken language system is in general divided into two stages: a continuous speech recognition system followed by higher level language processing modules. First, the frame level acoustic information is processed by a continuous speech recognizer. The output of a continuous speech recognizer, strings of symbols or words, are then fed into higher level language modules such as a sentence parser, semantic analysis modules, etc. for further processing. Unfortunately, such a division was usually done at a price of sacrificing the optimality of solutions and the final output, in most cases, is only suboptimal or not optimal at all. This compromise is not really necessary and global optimality can be obtained if the following two conditions are satisfied: first, the reduction of the search space in the first stage should not too greedy to cause any hard errors so that the optimal solution are dropped from being considered for further processing in the first stage; and second, the output of all processing modules should be rank ordered according to some universal optimality criteria like the likelihood scores.

To fulfill the above two conditions, it is important to to devise an efficient search for the N-best sentence hypotheses in the first stage where N should be adaptively adjustable. By doing so, we can preserve the optimality of final results while reducing the search space to a manageable size, i.e., an N element subset of all possible hypotheses. Existing efforts for finding N-best sentence hypotheses were devised in the level building algorithm by Meyer and Rabiner [1], a frame synchronous network search by Lee and Rabiner [2] and recently a sentence hypothesis search by Steinbiss [3]. The resultant N-best hypotheses, however, are only optimal under constrained conditions: lower ranked sentence candidates are derived

* On leave from Telecommunication Labs, Chung-li, Taiwan.

from the sentence segmentation of higher ranked candidates. Due to the constraints, the top-N candidates thus derived are not exact.

Recently an exact, top-N candidate search was proposed by Chow and Schwartz [4]. The top-N string hypotheses are obtained in a Viterbi-like, breadth-first search set-up in the first stage and they are then processed by knowledge sources in the second stage. Another approach, proposed by Paul [5], is to use stack-based decoding algorithms [6,7,8] as common tools for a continuous speech recognizer and natural language processing modules. Likelihood scores are used as a common metric shared by both the recognizer and language modules. In the same paper, a natural interface was proposed to link a continuous speech recognizer and higher level language modules.

In this paper, we present a newly proposed, fast tree-trellis based N-best search [9] for finding the top-N sentence hypotheses in a continuous speech recognizer. The search uses an A^* algorithm for finding the top N sentence hypotheses. However, different from other A^* or the stack algorithm where heuristic ways are used to evaluate path scores before completion and optimalities of the solutions are compromised, the new algorithm is an exact N-best search and only exact, not heuristic, path scores are used. The new algorithm generates the N-best hypotheses sequentially. Furthermore, the number of hypotheses need not to be preset and the algorithm can terminate at any time whenever a string hypothesis is accepted as a valid sentence. The search is also computationally efficient due to its tree-trellis search nature. The new algorithm only needs a fraction more computation than a normal Viterbi search for finding the top N sentence hypotheses.

The rest of the paper is organized as follows. In the next section, we present the fast tree-trellis algorithm. In Section III we discuss the optimality of the algorithm. In Section IV we present results obtained from testing the algorithms on two different applications: connected digit recognition using check-sum rules and the DARPA Resource Management task using the word-pair grammar and the original finite state automata of DARPA sentences. In Section V we confirm the efficiency of the tree-trellis search by presenting a CPU time breakdown of different computation modules.

II. The Tree-Trellis N-Best Search

The proposed algorithm, as its name indicates, is a fast search by combining a tree search based A^* [10] (or "stack") algorithm [6,7,8] with a trellis search based modified Viterbi algorithm. A block diagram of the algorithm is shown in Fig. 1.

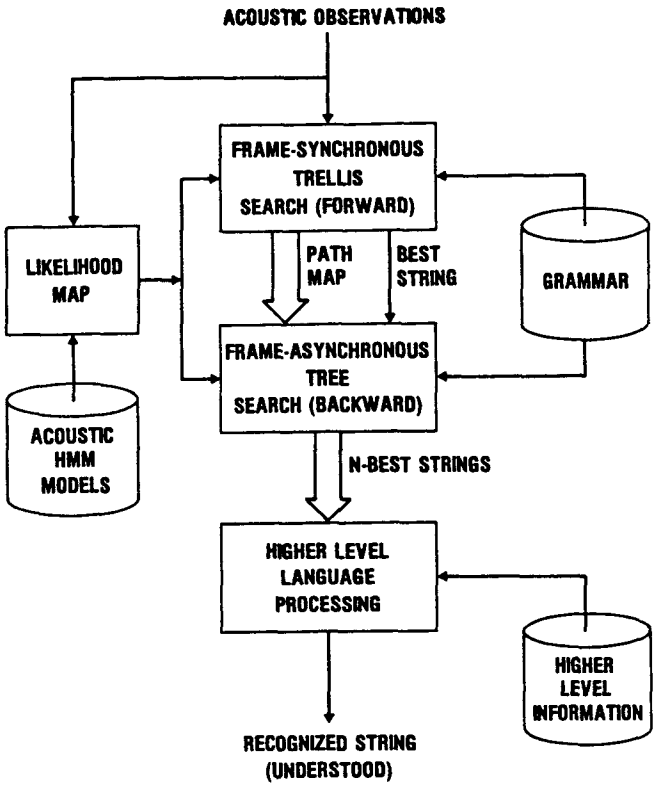

Fig. 1. Block diagram of the tree-trellis search

As shown in the diagram, first, input acoustic observations are compared with HMM models and a corresponding log likelihood map is generated. The trellis search is then performed time (frame) synchronously in the forward (left-to-right) direction and the search is guided by a given grammar and/or any knowledge source which is incorporated into the Viterbi search. In addition to output the best sentence hypothesis, a path map of all partial paths is registered in the Viterbi search. The partial path map contains scores of all partial paths that lead to any grammar node at every time instant.

At the end of the trellis search, a new, tree search for finding the best N sentence hypotheses is initiated. The search is performed backward in time and frame asynchronously. It is a best-first search implemented by using an A^* search or the stack algorithm. The N-best candidates are found one at a time and each candidate is then fed sequentially to higher level processing modules. The final result of the whole system is a recognized (understood) string. In the following subsections we present individual modules of the tree-trellis search in detail.

II.1 Modified Viterbi Algorithm (MVA)

The Viterbi algorithm (trellis search) is modified in the new tree-trellis search to generate a partial path map. The map is needed by the A^* tree search. The modified

Viterbi algorithm is given as follows.

Trellis Search (Modified Viterbi) Algorithm

INITIALIZE: (1) path scores; (2) arc ranking indices; (3) backpointers (optional)

LOOP I: loop over time indices from left to right

LOOP II: loop over grammar nodes

LOOP III: loop over arcs of a grammar node

LOOP IV: loop over states of an arc (word)

Evaluate dynamic programming recursion

— Update accumulated likelihood arrays
— Update backpointer arrays (optional)

LOOP IV control

For every grammar node,

— sort accumulated likelihood path scores
— register arc ranking index arrays
— register "from frame" arrays (optional)

LOOP III control

LOOP II control

LOOP I control

After all bookkeeping arrays are initiated, four nested loops are performed. The dynamic programming starts first from the outermost loop, a loop over the time indices from left to right frame synchronously, over a loop of all grammar nodes, then over all arcs (words) of a grammar node and finally, over the innermost loop of all states associated with an arc (word). Since the best path will be obtained in the backward tree search as the first sentence hypothesis output, it is not necessary to register any backpointer arrays and all backtracking operations are only optional. The arc (word) ranking index arrays are recorded only when the number of possible arcs (words) at a node exceeds N, the number of sentences hypotheses to be found. In addition to the best partial path, all the other partial paths that lead to a grammar node, are recorded in the modified Viterbi algorithm.

II.2 Tree Search A^* Algorithm for Finding the N-best Strings

At the end of the modified Viterbi search, a backward tree search is initiated from a terminal node and the search is performed time asynchronously in a backward (right-to-left) direction. The tree search is implemented using an A^* search or the stack algorithm. However, different from a typical A^* search where the incomplete portion of a partial path is estimated (or predicted) using some heuristics, the tree search here uses the partial path map prepared in the first stage trellis search and the score of the incomplete portion of a path in the search tree is then exactly known. The backward partial paths are rank ordered in the stack based upon the exact scores of their corresponding full but yet incomplete paths. The tree-trellis search algorithm has other advantages over a breadth-first N-best search in its ability to output sequentially the N-best hypotheses, one at a time, according to descending likelihood scores. The backward tree search can be best illustrated by a conceptual diagram depicted in Fig. 2.

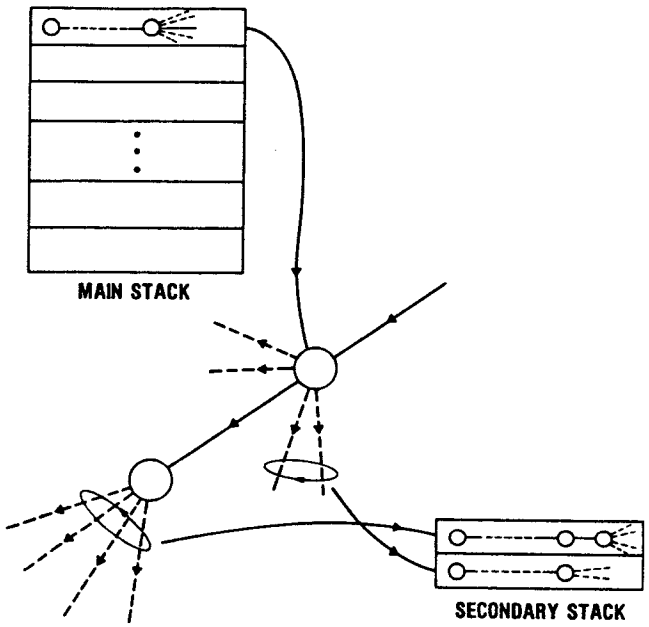

Fig. 2. Conceptual diagram of path growing in a tree

Two stacks, a main stack and a secondary stack, are two list data structures for storing partial hypotheses. In the main stack, all partial paths, are rank ordered according to their likelihood scores. The best partial path at the top of the main stack gets extended in every path growing cycle. As shown in the figure, the top entry in the main stack is first split into two parts, the best one word (arc) extension and the set of remaining one word (arc) extensions. These two extensions are stored temporarily in the secondary stack and then reinserted back into the main stack so that all main stack entries are still rank ordered. The modified A^* algorithm used to grow a tree is summarized as follows:

N-Best Tree Search (A^*) Algorithm

INITIALIZE: put the root node in a rank
ordered list (main stack) to form a null partial path

LOOP: best first path growing loop

Take the top entry (the best partial path) off the main stack

IF the top entry is a single path (i.e., not a group of partial paths),
THEN

IF the best partial path is complete (i.e., leads to a
terminal node), THEN

output the path and increment the output
hypothesis counter by one

IF output counter equals N, THEN

stop

ENDIF

ELSE

— Split the partial path into two sets: the best one-arc extension and the remaining one-arc extensions.

— Use the partial path map provided by the Viterbi algorithm in evaluating the one-arc extensions.

— Store the two sets temporarily into the secondary stack and then reinsert them back into the main stack such that the main stack is still rank ordered. Ranking is based upon complete path scores.

ENDIF

ELSE

— Split the set of partial paths into two sets: the best partial path and the remaining partial paths in the set.

— Store them temporarily into the secondary stack and then reinsert them back into the main stack such that the main stack is still rank ordered.

ENDIF

LOOP CONTROL

II.3 Partial Path Merging at a Grammar Node

The search of the N-best paths in continuous speech recognition is somewhat more complicated than a typical graph search problem due to the time varying nature of the graph, i.e., the cost of a path varies with time. In other words, a single path in a graph is actually a set of paths of different time signatures (trajectories). Since we consider only paths of different word content, paths of the same word content but with different trajectories have to be compared first and only the best path is retained and it is then compared with other best paths of different word content. In search of the N-best sentence hypotheses, the best paths between the start node and the terminal node can pass any given node in-between at any time instants should be compared. These paths can be further divided into two sets of partial paths: backward partial paths grown in the tree search with forward partial paths grown in the trellis search at a specific grammar node as illustrated in Fig. 3.

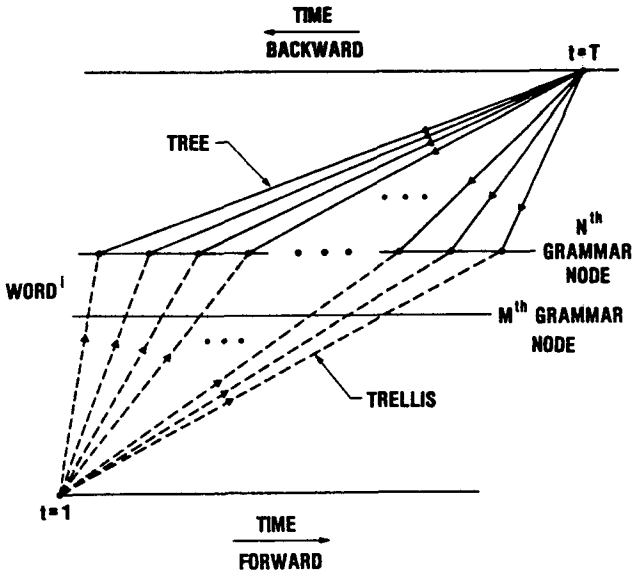

Fig. 3. Merging forward and backward partial paths

As shown in the figure, solid lines represent partial paths grown by the backward tree search while broken lines represent partial paths grown by the forward trellis search. To be specific, partial paths leading to a grammar node, say the N^{th} node, from a terminal node and partial paths stemming from the root node, passing a grammar node which is a predecessor of the N-th node, say the M-th node, along the arc of word ^{i} are merged at the N-th node at matched time instants. The best path is the one with the maximum summed likelihood scores.

III. Optimality of the Tree-Trellis Search

The optimality of the A^* search has been proven before, e.g., [10]. It is stated as the admissibility of the A^* algorithm. That is, if there is a path from the root node to a terminal node, A^* terminates by finding an optimal path. There is also an interesting property that associates with any node n chosen for path growing. That is, the path score, a summation of the computed partial path score from the root node to the node n and the estimated incomplete partial path score from node n to the terminal node, is equal to or better than the optimal path score between the root node and the terminal node. The equality holds in our modified A^* algorithm because the exact rather than an estimated score of the incomplete path between n and the terminal node is precomputed in the first stage Viterbi search and is readily available. This equality maximizes the A^* search efficiency and minimizes the main stack size to N, the number of candidates. The search efficiency is maximized because the exact score for the incomplete partial path is used instead of an estimated upperbound.

In terms of the storage, a main stack of size N, is sufficient to maintain the N-best hypotheses within the search procedure because any partial path on the stack does not change its optimal (complete) path score throughout the search.

IV. Applications to Check-Sum Based Connected Digit Recognition

The development of the fast tree-trellis search for finding the N-best candidate sentence strings was originally motivated by a continuous digit recognition application. The American Express (AMEX) credit card company initiated a project recently to automate its procedure in verifying the merchant I.D., the credit card number and authorizing the dollar purchase amount via telephone calls. Currently all verifications and authorizations are carried out by human operators and the final goal of the project is to replace as many human operators as possible by automatic speech recognizers while maintain comparable recognition performance. Both credit card numbers and merchant I.D.'s are fixed length digit strings, i.e., 10 digits for a merchant I.D. and 15 digits for a credit card number.

The last digit of each digit string (merchant I.D. or a credit card, is a check-sum digit, which is a nonlinear, modulo, combination of the previous digits. The check-sum digit is installed for security reasons and the exact formulas for generating the check-sum digit are not given here. The check-sum rules, despite their simplicity, can not be incorporated directly into a continuous speech recognizer. Because except by using an exhaustive, hence prohibitive, finite state network, the check sum formulas can not be tested before all digits in a string are available (recognized). Consequently, the search for the correct digit string is thus divided into two stages: a continuous digit recognition and a check-sum test. The fast tree-trellis search for finding the top-N candidates is then ideal for this two-stage processing. The sentence hypotheses are found sequentially and they are tested against the check-sum rules. If at any time, a string passes the check-sum rules, search stops. Otherwise, the next digit string with a lower likelihood score is then fetched and tested.

Two digits strings (credit card number and merchant I.D.) from each speaker were recorded over each local or long distance telephone call to the American Express processing facilities in Phoenix, Arizona. 1,800 digit strings recorded by 900 speakers constitute the training

data base. Separate 100 strings of merchant I.D.'s and 114 strings of credit card numbers recorded by a different set of speakers form the test data base. Various "real world" conditions are present in the recordings, including: speakers with a strong foreign accent, music or human conversations in the background, tone noise, cross-channel interference from different telephone lines, etc. The SNR of the recordings is around 25 to 30 dB.

In constructing word-based HMMs, 13 words were chosen to form the vocabulary, which consists of the ten digits, i.e., {"0" to "9"}, "oh", silence, and extraneous speech. Two HMM models were built for each word in the vocabulary. For each state in a word models, a 64 mixture component continuous Gaussian mixture probability density function was trained. Both the best string Viterbi search and the top-*N*, tree-trellis search were used to recognized the strings in the test data base.

The recognition results are tabulated in Table I by string accuracies.

Type	Top 1	Top 10 + check sum
Credit Card	84	98
Merchant I.D.	82	97

Table I. Recognition String Accuracy (%) for the AMEX Trials

The string accuracies of the best word hypothesis obtained from an unmodified Viterbi decoding are 84% and 82% for the credit card and merchant I.D. recognition, respectively. However, when we used the check-sum rules to check the top-10 candidate obtained from the tree-trellis search the string accuracy was improved from 84% to 98% for the credit card number recognition and from 82% to 97% for the merchant I.D. recognition. This high level of recognition performance has been reported before e.g., [11], but it was achieved with a clean microphone data base recorded in a sound booth. The results of this experiment demonstrate that high performance connected digit recognition in a real world environment is achievable when simple error detection rules are used in conjunction with the new tree-trellis search algorithm. As an example, the top-10 candidates of a merchant I.D. recognition trial is listed according to their correspond decreasing likelihood scores.

1 4 2 8 4 11 2 7 11 5

time for likelihood map = 55.29

time for Viterbi = 37.06

1	19.25	1	4	2	8	4	11	2	7	11	5	**
2	19.19	1	4	2	7	4	11	2	7	0	5	
3	19.18	1	4	2	8	4	11	2	7	0	5	
4	19.15	1	4	2	9	4	11	2	7	11	5	
5	19.15	1	4	2	8	4	11	7	7	11	5	
6	19.14	1	4	2	8	4	11	2	7	11	4	
7	19.14	1	4	2	8	4	11	0	7	11	5	
8	19.13	1	4	2	7	4	11	2	7	0	5	
9	19.12	1	4	2	6	4	11	2	7	11	5	
10	19.12	1	4	2	2	4	11	2	7	11	5	

time for multi-candidate tree search = 5.89

In this example, while the best digit string happens to be the correct string, the rest 9 candidate string are different from the correct string only by one or two digits and all likelihood scores are very close. Also shown are the CPU time breakdown for computing the likelihood map, the trellis (Viterbi) search and the tree search. The time required in the tree search for finding the top-10 candidate strings is only about 15% of the amount needed in the forward trellis search. A different example of merchant I.D. recognition is depicted in Fig. 4, where the 10-best digit strings are displayed with corresponding word boundaries, word likelihood scores (average) and their rankings.

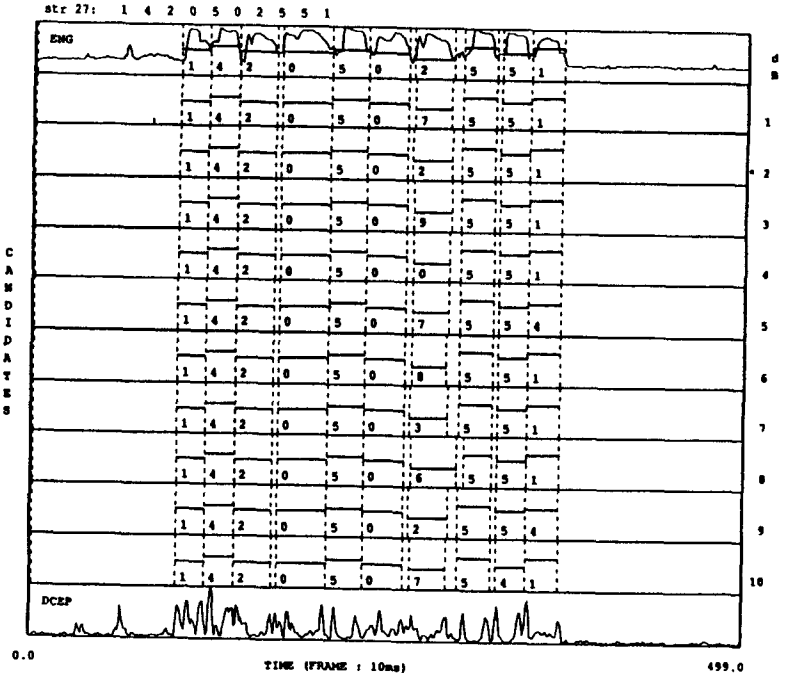

Fig. 4. Multi-candidates and their segmentations

The segmented but unmarked portions in the figure were recognized as either silence or extraneous speech. The correct string, or the second best recognized string, is different from the best recognized string by a single digit confusion ("2" with "7"). Similar competing word tokens can be collected for training discriminative HMM models [12-15].

V. Applications to the DARPA Resource Management Task

The new search procedure was also applied to the DARPA resource management task and some preliminary results are reported here. Forty seven context-independent HMM phone models were trained by using 3,200 sentences (40 sentences/talker). Each phone HMM has 3 states and the output probability density function (pdf) of each state is characterized by a 32-component, Gaussian mixture densities. A 150-sentence test set used by Lee [16] was used as a test set and recognition string accuracies are given in Table III.

Type	Top 1	Top 100 + FSN check
CMU150	38	75

Table II. String accuracy (%) for the DARPA task

When a beam-search based Viterbi search was used to decode an input string under a word-pair grammar constraint, a sentence string accuracy of 38% was obtained (perplexity 60). But when we used the tree-trellis search and incorporated the finite state network (i.e., a perplexity 9) as the second stage processing, the sentence accuracy was almost doubled to 75% as shown in the table. This result, in principle, can be similarly achieved by using a full finite state grammar search but with much higher search complexities. An example of the top-10 candidates obtained in the search is given as follows along with their corresponding average log likelihood (per frame).

The correct string is the 7-th candidate with an average log likelihood score of 11.806, only 0.03 less than the string with the highest score. Almost all major content words, especially those with longer durations, are recognized in the top-10 strings. The extra computation effort for finding the top-10 candidates in the tree search is 1.8 sec, about 2.5% of the time needed for the forward trellis beam search.

VI. Computation Breakdown of the Tree-Trellis Algorithm

By breaking the search into a trellis (modified Viterbi) and a tree search, the N-top sentence hypotheses can be obtained sequentially and the search effort is greatly reduced compared with other breadth-first based N-top candidate search. We used the internal timing routines of an Alliant computer to measure the CPU time spent on each individual module in the AMEX digit recognition trials. The breakdown is given in terms of percentage in Table III.

Type	Percentage (%)
Likelihood Map	56
Trellis	38
Tree (Top 10 + checksum)	6

Table III. Computation breakdown (%) for the new search

HOW SOON CAN ESTEEM CHOP TO ATLANTIC FLEET

time for likelihood map = 16.79

time for Viterbi beam search = 71.29

1	11.836	HOW SOON CAN ESTEEM IN SOUTH TWO ATLANTIC FLEET
2	11.824	HOW SOON CAN ESTEEM A SOUTH TWO ATLANTIC FLEET
3	11.823	HOW SOON CAN ESTEEM IN SOUTH TWO ATLANTIC FLEET TO
4	11.823	HOW SOON CAN ESTEEM IN SOUTH TWO ATLANTIC FLEET IN
5	11.811	HOW SOON CAN ESTEEM A SOUTH TWO ATLANTIC FLEET TO
6	11.811	HOW SOON CAN ESTEEM A SOUTH TWO ATLANTIC FLEET IN
7	11.806	HOW SOON CAN ESTEEM CHOP TO ATLANTIC FLEET ***
8	11.804	HOW SOON CAN ESTEEM TO SOUTH TWO ATLANTIC FLEET
9	11.804	HOW SOON CAN ESTEEM THE SOUTH TWO ATLANTIC FLEET
10	11.799	HOW SOON CAN ESTEEM IN SOUTH ATLANTIC FLEET

time for multi-candidate tree search = 1.79

As shown in the table, the likelihood map computation, for the specific task (AMEX trials) and the HMMs used constitutes about 56% of the total CPU time. More efficient algorithm can be implemented, e.g., a partial rather than a full table of likelihood functions can be computed on demand, but it was not incorporated in our implementation. The trellis search, or the modified Viterbi algorithm, consumes about 38% of the CPU time while the final top-10 candidate, tree-search needs the rest 6%. The check-sum test, based upon very simple arithmetics, takes virtually no CPU time at all. The extra computational needed for finding the top N candidates of the tree-trellis search is minimal.

VII. Conclusion

In this paper, we propose a new, tree-trellis based, fast search algorithm for finding the top N sentence hypotheses in continuous speech recognition. The algorithm uses a bi-directional search consisting of a forward, time synchronous, trellis search and a backward, time asynchronous, tree search. In the new algorithm, due to the partial path map prepared in the trellis search, the backward tree search is highly efficient and a shallow stack, i.e., of a size N, is needed. The algorithm has been tested successfully on two different data bases: the American Express credit service data base and the DARPA resource management data base. In the former data base, multiple candidate digit strings were successively generated and tested against some check-sum rules, the digit string accuracy was improved by 14-15%. For the DARPA database, when the finite state grammar was used to screen out invalid sentence hypotheses in the top 100 candidates, the string error was reduced by more than a half. It was also shown in the experiments that the top N candidates were obtained with a minimal computational overhead.

References

[1] Meyer, C. S. and Rabiner, L. R., "Connected Digit Recognition Using a Level Building DTW Algorithm," IEEE trans. on ASSP, Vol. ASSP-29, pp. 351-363, June 1981.

[2] Lee, C. H. and Rabiner, L. R., "A Frame Synchronous Level Building Algorithm for Connected Word Recognition," Comput. Speech Language, Vol. 1, no. 1, pp. 29-45, Mar. 1986.

[3] Steinbiss, "Sentence Hypothesis Generation in a Continuous Speech Recognition System," Proc. European Conf. on Speech Comm. and Tecl. pp. 51-54, Paris, Sept. 1989.

[4] Chow, Y. and Schwartz, R., "The N-Best Algorithm: An Efficient Procedure for Finding top N Sentence Hypotheses," Proc. Speech and Natural Language Workshop, Oct., 1989, pp. 199-202, also Proc. ICASSP-90, pp. 81-84, Apr. 1990, NM.

[5] Paul, D., "A CSR-NL Interface Specification, Version 1.5," Proc. Speech and Natural Language Workshop, Oct. 1989, pp. 203-214.

[6] Jelinek, F. "A Fast Sequential Decoding Algorithm Using a Stack," IBM J. Res. Develop., Vol. 13, pp. 675-685, Nov. 1969.

[7] Jelinek, F., Bahl, L. R., Mercer, R. L., "Design of a Linguistic Statistical Decoder for the Recognition of Continuous Speech," IEEE Trans. on Information Theory, Vol. IT-21, No. 3, pp. 250-256, May 1975.

[8] Sturtevant, D., "A Stack Decoder for Continuous Speech Recognition," Proc. Speech and Natural Language Workshop, Oct. 1989, pp. 193-198.

[9] Soong, F. K. and Huang, E.-F., "A Fast Tree-Trellis Search for Finding the N-Best Sentence Hypotheses in Continuous Speech Recognition," J. Acoust. Soc. AM. S-1, Vol. 87, pp. 105-106, May 1990.

[10] Nilsson, N., *Problem-Solving Methods in Artificial Intelligence,* NY, NY, McGraw Hill, 1971.

[11] Rabiner, L. R., Wilpon, J. G., Soong, F. K., "High Performance Connected Digit Recognition Using Hidden Markov Models," IEEE Trans. on Acoust. Speech and Sig. Proc., Vol. 37, No. 8, pp. 1214-1225, Aug. 1989.

[12] Bahl, L., Brown, P. DeSousa, P., and Mercer, R., "A New Algorithm for the Estimation of Hidden Markov Model Parameters," Proc. ICASSP-88, pp.493-496, Apr. 1988.

[13] Doddington, G., "Phonetically Sensitive Discriminants fro Improved Speech Recognition," Proc. ICASSP-89, pp. 556-559, May 1989.

[14] Huang. E.-F. and Soong, F. K., "A Probabilistic Acoustic Map Based Discriminative HMM Training," Proc. ICASSP-90, pp.693-696, Apr. 1990.

[15] Chow, Y.-L., "Maximum Mutual Information Estimation of HMM Parameters for Continuous Speech Recognition Using the N-Best Algorithm," Proc. ICASSP-90, pp.701-704, Apr. 1900.

[16] Lee, K.-F., *Large Vocabulary Speaker-Independent Continuous Speech Recognition: The Sphinx System,* Ph.D Dissertation, Computer Science Department, Carnegie-Mellon Univ., Apr. 1988.