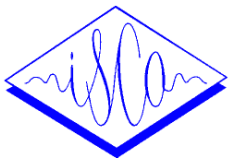

First International Conference on
Spoken Language Processing (ICSLP 90)
Kobe, Japan
November 18-22, 1990

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

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 a forward, time-synchronous, trellis search and a backward, time asynchronous, tree search. The well known Viterbi algorithm is used for finding the best hypothesis in a trellis and for recording the scores of all partial paths time synchronously. A backward A^* algorithm based tree search is used to grow partial paths 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

Search for the correct sentence hypothesis in a spoken language recognition system is usually performed in two consecutive stages: a continuous speech recognizer followed by a higher level language processing system. Various existing efforts for finding the N best sentence hypotheses can be found 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, are derived from the sentence segmentation of the best or higher ranked candidates. As a result, the top N candidates 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 a "stack" algorithm [6,7,8] for continuous speech recognition.

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 [10] or the stack algorithm for finding the top N sentence hypotheses. However, different from other A^* 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 path scores are used. The new algorithm also generates the N best hypotheses sequentially and there is no need to preset N. The search can terminate at any time whenever a string hypothesis is accepted as a valid sentence. Computation wise, the new algorithm only needs a fraction more computation than a normal Viterbi search for finding the top N (10 - 20) sentence hypotheses. This new algorithm has also been incorporated into the MIT VOYAGER system [11].

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

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 the stack algorithm [6,7,8] with a trellis search based modified Viterbi algorithm. A block diagram of the algorithm is shown in Fig. 1.

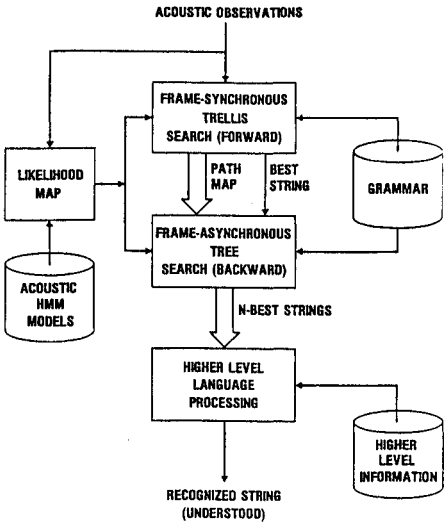

Fig. 1. Block diagram of the tree-trellis search

As shown in the diagram, input acoustic observations are first compared with HMM models and a log likelihood map is generated. The trellis search is then performed time (frame) synchronously in the forward (left-to-right) direction. A partial path map which contains the likelihood scores of all partial paths leading to every grammar node at every frame is also generated in the Viterbi search.

At the end of the trellis search, a backward tree search is initiated to find the N best sentence hypotheses. The search is based upon an A^* search [10] and is performed frame asynchronously. The N best candidates are found one at a time and each candidate is fed sequentially to higher level processing modules. The final result of the whole system is a recognized (understood) string. In the following subsections each individual module of the tree-trellis search is discussed in detail.

II.1 Modified Viterbi Algorithm (MVA)

The Viterbi algorithm (trellis search) is modified as follows to generate a partial path map needed in the tree search.

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 I over time indices frame synchronously, then LOOP II over all grammar nodes, LOOP III over all arcs (words) of a grammar node and finally, over the innermost LOOP IV of all states within each arc (word).

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. The tree search is implemented using an A^* search. However, different from a typical A^* search where the incomplete portion of a partial path is estimated (or predicted) by 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 is exactly known. The tree-trellis search algorithm is also advantageous over a breadth-first N-best search [4] 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.

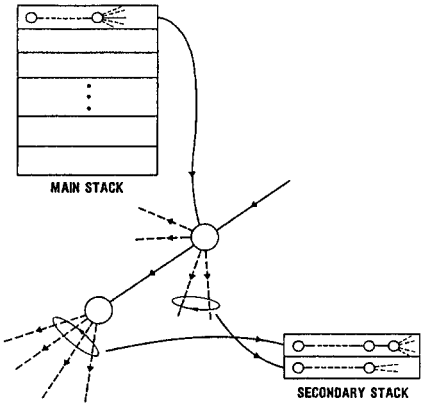

Fig. 2. Conceptual diagram of path growing in a tree

Two list data arrays, the main stack and the secondary stack, are used in the tree search. All partial paths are rank ordered on the main stack according to the likelihood scores of full but yet incomplete paths. 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 they are then reinserted back into the main stack. 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 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.

ENDIF

LOOP CONTROL

II.3 Partial Path Merging at a Grammar Node

The search of the N-best paths in continuous speech recognition is more complicated than a typical graph search problem due to the time varying aspects, i.e., there are usually many paths which are made of the same word content but associated with different time trajectories and path scores. Since we consider only paths of different word content, paths with the same word content but different temporal trajectories have to be compared first and only the best path is retained. Then the retained path is 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 which passes any node in-between 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.

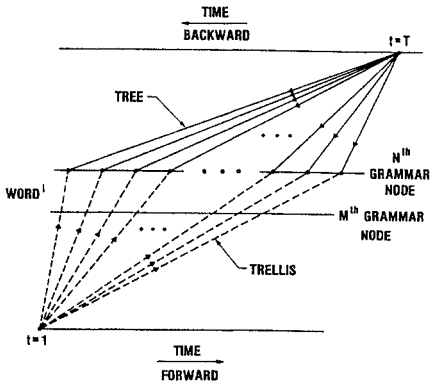

Fig. 3. Merging forward and backward partial paths

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 combined likelihood scores of the two partial (both forward and backward) paths.

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. Computationally, since the exact instead of an estimated score of an incomplete path is used, the A^* search efficiency is maximized. Storage requirements of the tree-trellis algorithm is also optimized; a stack of size N is sufficient to keep the N best hypotheses because the rank ordering, which is based upon the exact complete path score, of a partial path does not change throughout the whole A^* search procedure.

IV. Applications to 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. 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 fast tree-trellis search is ideal for this application. First the sentence hypotheses are found sequentially and then tested against the check-sum rules. A string is recognized if it passes the check-sum rules.

Two digit strings (credit card number and merchant I.D.) from each speaker were recorded over each telephone call (local or long distance) to the American Express processing facilities in Phoenix, Arizona. The training database consists of 1,800 digit strings recorded by 900 speakers. Additionally, 100 strings of merchant I.D.'s and 114 strings of credit card numbers recorded by a different set of speakers form the test database.

The vocabulary consists of 13 words, including: 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. The recognition results are tabulated in Table I.

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

Table I. String accuracy (%) for the AMEX recognition 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. When the check-sum rules were used to check the top 10 candidates obtained from the tree-trellis search, the string accuracy was improved from 84% to 98% for the credit card number and from 82% to 97% for the merchant I.D. An 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 rankings.

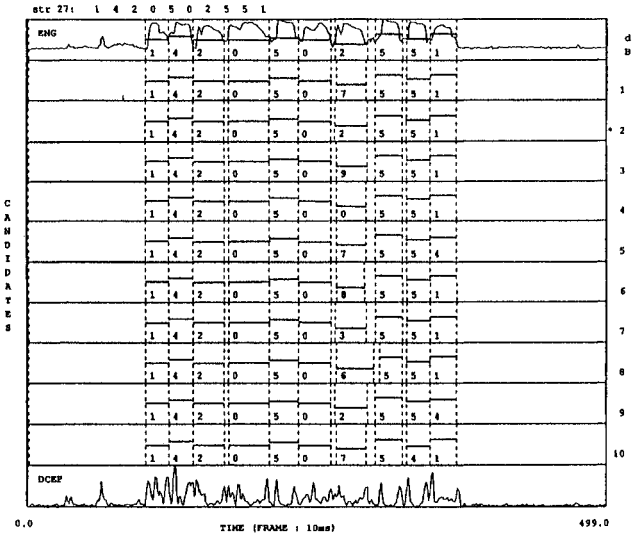

Fig. 4. Multi-candidates and their segmentations

In the figure the unmarked portions were recognized as either silence or extraneous speech. The top 10 strings are different from each other usually by one, or at most, two digits. The correct

string which is the second best recognized string, is different from the best string by a single digit (i.e., "2" confused with "7").

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 [12] was used in the recognition experiment. The recognition string accuracies are given in Table II.

Type	Top 1	Top 100 + FSN check
CMU150	38	75

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

When a conventional beam-search based Viterbi algorithm was used to decode input strings under a word-pair grammar constraint (perplexity 60) [12], a sentence string accuracy of 38% was obtained. When the new tree-trellis search was used in the first stage followed by a finite state network based sentence check (perplexity 9) of the RM database in the second stage, the string accuracy was almost doubled to 75%. An example of the top 10 candidates obtained in the search is given as follows along with the corresponding average log likelihood scores (per frame).

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

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

VI. Computation Breakdown of the Tree-Trellis Algorithm

We used the internal timing routines of an Alliant computer to measure the CPU time spent on each individual modules for the AMEX digit recognition trials and the result is given in Table III.

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

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

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 new algorithm is computationally efficient and the top N candidates are obtained with a minimal computational overhead. It is also efficient in storage and a shallow stack of size N, is needed. Significant performance improvement has been demonstrated when the new algorithm was tested on the two test databases.

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] Zue, V., Glass, J., Goodine, D., Leung, Hong, McCandless, M, Phillips, M., Polifroni, J. and Seneff, S., "Recent Progress on the VOYAGER System," Proc. DARPA Speech and Natural Language Workshop, June 1990.

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