A TREE SEARCH STRATEGY FOR LARGE-VOCABULARY CONTINUOUS SPEECH RECOGNITION

P. S. Gopalakrishnan, L. R. Bahl and R. L. Mercer*
IBM T.J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

ABSTRACT

In this paper we describe a tree search strategy, called the *Envelope Search*, which is a time-asynchronous search scheme that combines aspects of the A* heuristic search algorithm with those of the time-synchronous Viterbi search algorithm. This search technique is used in the large-vocabulary continuous speech recognition system developed at the IBM Research Center.

1 INTRODUCTION

The large-vocabulary continuous speech recognition system developed at the IBM Research Center uses a tree search algorithm called the *Envelope Search* for determining the recognized word sequence in response to a spoken utterance. This is an asynchronous search scheme that combines aspects of heuristic search used in the A* algorithm [1] with techniques used in the Viterbi search algorithm [2]. A* type search algorithms for speech recognition have been used at IBM [3, 4] and MIT Lincoln Labs [5]. Most of the other large vocabulary continuous speech recognition systems currently under development, e.g. at CMU [6], BBN [7], and SRI [8], use search strategies based primarily on the Viterbi synchronous search algorithm. The introduction of some ideas from heuristic search leads to a reduction of computational and storage needs without sacrificing accuracy.

The goal of the recognizer is to find the word sequence that has the maximum *a posteriori* probability given an observed acoustic input, i.e., given a sequence of acoustic observations $Y = y_1 y_2 \dots y_T$ the decoder chooses a sequence of words $W = w_1 w_2 \dots w_N$ such that

$$\begin{aligned} \hat{W} &= argmax_W Prob(W|Y) \\ &= argmax_W Prob(Y|W) Prob(W) \qquad (1) \end{aligned}$$

*Currently at Renaissance Technologies, Stony Brook, NY

The *acoustic match* probability $Prob(Y|W)$ is obtained from acoustic models, which are generally hidden Markov models. The *language model* probability $Prob(W)$ is obtained from a model of the language, most commonly word n-gram models.

In order to find the most likely word sequence, we conduct a word level tree search. Conceptually, we can organize the set of all possible word sequences into a word-level tree. The root node corresponds to the null word sequence. Each branch in the tree corresponds to a word, thus each path starting at the root node corresponds to a sequence of words. Paths ending at terminal nodes of the tree correspond to complete sentences. Paths ending at non-terminal nodes correspond to partial sentences. Given a sequence of acoustic observations, the search algorithm explores the tree and attempts to find the complete path having the highest score. The decoder keeps a dynamically changing list of partial paths that have been explored so far. This list is initialized to contain the null path corresponding to the root node of the search tree. During the search process the decoder also keeps track of the best complete path found so far. Of course, during the early stages of the search, there will not be any such path. Complete sentences are compared on the basis of the cost function

$$\Lambda(W) = \log Prob(Y|W) Prob(W). \qquad (2)$$

For partial paths we use a heuristic evaluation function which is described in detail below. Based on this heuristic evaluation function the decoder designates each partial path as *alive* or *dead*. In each iteration of the search, one of the *alive* partial paths is chosen for extension. Extension of this path results in new paths that are each one word longer. Newly created partial paths are added to the list of partial paths. Each newly created complete path is compared to the best complete path and replaces it if it is better. The search terminates when there are no partial paths that are *alive*. The best complete path is then output as the recognized sentence.

2 SOME TERMINOLOGY

In order to explain the search in greater detail it is useful to establish some terminology. Each path corresponds to a word sequence $W_1^k = w_1 w_2 \ldots w_k$. Let $t_{end}(W_1^k)$ denote the most likely end time for W_1^k in the acoustic sequence Y. Given a word sequence W_1^k and the acoustic sequence Y, the most likely end times $t_{end}(W_i^j), j = 1, 2, \ldots k$ can be obtained from the acoustic match calculation. Details of a method for calculating the most likely end times can be found in Bahl *et al* [9]. Thus, given the word sequence W_1^k and the output sequence Y, we assume that:
word w_1 accounts for frames $1, 2, \ldots t_{end}(W_1^1)$,
word w_2 accounts for frames $t_{end}(W_1^1)+1, \ldots t_{end}(W_1^2)$,
etc. In general,
word w_j accounts for frames $t_{end}(W_1^{j-1})+1, \ldots t_{end}(W_1^j)$.
For each word sequence W_1^k and time frame index $t = 1, 2, \ldots t_{end}(W_1^k)$ we can define a log likelihood value

$$L(W_1^k, t) = \log Prob(W_1^j) Prob(Y_1^t | W_1^j)$$

for

$$t_{end}(W_1^{j-1}) < t \leq t_{end}(W_1^j).$$

For $t > t_{end}(W_1^k)$, the log likelihood values are undefined. Thus each path W_1^k has an array of log likelihood values $L(W_1^k)$ associated with it, where

$$L(W_1^k) = [L(W_1^k, 1), \ldots L(W_1^k, t_{end}(W_1^k))].$$

The size of this array is $t_{end}(W_1^k)$. We refer to this array of values as the *profile* of the path. We will denote the last value in the array as $L_{end}(W_1^k)$, i.e.

$$L_{end}(W_1^k) = L(W_1^k, t_{end}(W_1^k))$$

Figure 1 shows a graphical representation of a path *profile*.

During the search process, the decoder keeps a dynamically changing list of partial paths that have been explored so far. We will refer to this list as PARTIAL. This list is initialized to contain the null path corresponding to the root node of the search tree. In PARTIAL, the paths are ordered in *longest-best* order. They are sorted with the primary sort index being the length of the path i.e. $t_{end}(W_1^k)$, and the secondary index being the log likelihood at the most likely end time of the path i.e. $L_{end}(W_1^k)$. This means that the paths in PARTIAL are ordered in descending order of their most likely end times; and paths having the

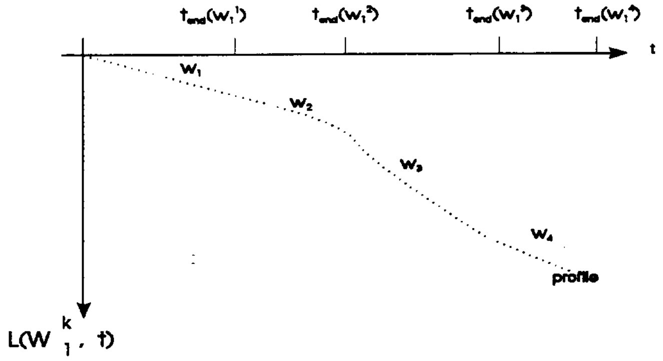

Figure 1. *Profile* of a path.

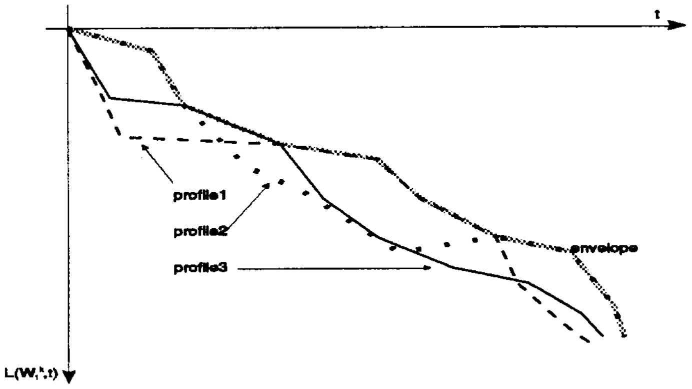

Figure 2. *Envelope* constructed from three *profiles*.

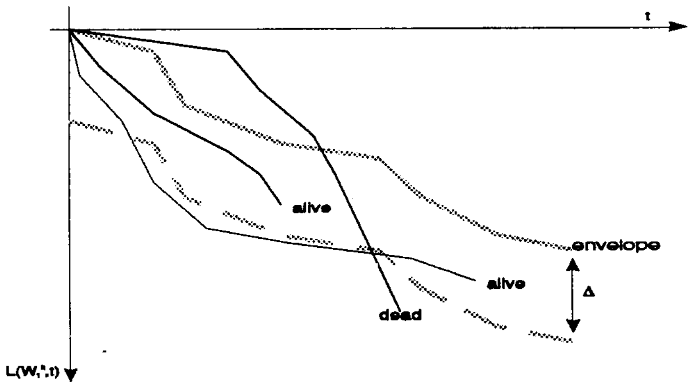

Figure 3. Examples of *alive* and *dead* paths.

same most likely end times are ordered in descending order of their log likelihood values at the most likely end times. During the *envelope* construction process (explained later), paths in PARTIAL will be examined in *longest-best* order.

During the search process the decoder also keeps track of the best complete path found so far. We refer to this path as BEST-COMPLETE. During the early stages of the search, there will not be any path correponding to BEST-COMPLETE. Complete sentences are compared on the basis of the cost function of equation (2).

Given a set of path *profiles* we can construct an *envelope* of log likelihood values which at each time frame index is the maximum value among the individual path *profiles*, i.e., the *envelope* is the *lowest upper bound (lub)* of the individual path *profiles.* Figure 2 shows a simple example of an *envelope* constructed from three path *profiles.* The three *profiles* are represented by dashed, dotted and solid lines respectively. The *envelope* which is the *lub* of the three *profiles* is the wider gray line.

Let $e(t)$ denote the *envelope* value at time frame index t. We will use $e(t)$ as a guide to decide whether a path is *alive* or *dead.* A path W_1^k is considered *alive* relative to an *envelope*, if

$$L_{end}(W_1^k) \geq e(t_{end}(W_1^k)) - \Delta$$

i.e. the log likelihood of the path at its most likely ending time is no more than Δ below the *envelope* value. If this condition is not satisfied a path is designated as *dead.* Figure 3 shows some examples of *alive* and *dead* paths relative to a given *envelope.*

The parameter Δ controls the width of the search. Small values of Δ will result in a fast search with some search errors; increasing Δ will reduce the number of search errors at the expense of more computation.

Amongst the *alive* paths, we will choose the *shortest-best* for extension. This means that amongst all the *alive* paths, the one having the smallest value of $t_{end}(W_1^k)$ will be chosen. If there is more than one *shortest* path, then the one having the highest value $L_{end}(W_1^k)$ amongst them will be chosen.

Once a path is chosen for extension, we will construct a *candidate list* of words for this path. Each word in the *candidate list* will result in a path that is one word longer than the chosen path. For small vocabulary systems the *candidate list* can be the entire vocabulary. For tasks where the sentences are constrained by a small finite-state grammar, the *candidate list* would contain all the words that can legally follow the partial path. In large-vocabulary systems the number of potential extensions of a path can be very large. Since it is expensive to compute the acoustic match probabilities for all possible extensions, we use a *fast match* [9] to perform a preliminary pruning of the set of successors to be evaluated.

3 THE ENVELOPE SEARCH ALGORITHM

```
1. Initialize PARTIAL to contain the null path.
   Mark BEST-COMPLETE to be non-existent.
2. If BEST-COMPLETE exists
     • then, initialize the envelope to be the
       profile of BEST-COMPLETE.
     • else, set all envelope values to $-\infty$.
3. For each path $W_1^k$ in PARTIAL
   (in longest-best order)
     • compare the most likely end time log likeli-
       hood value $L_{end}(W_1^k)$ to the current envelope
       value $e(t_{end}(W_1^k))$.

       If $L_{end}(W_1^k) \geq e(t_{end}(W_1^k)) - \Delta$
         – then, mark this path alive and update
           the envelope by including the profile of
           this path in the envelope construction process.
         – else, mark this path dead.
4. If all paths in PARTIAL are dead
     • then, terminate the search and output
       BEST-COMPLETE as the recognized
       output.
     • else, choose the shortest-best path for exten-
       sion, and remove it from PARTIAL.
       For each word in the candidate list of this
       path
         – Make an extension.
           If extended path is complete,
             * then, compare to BEST-COMPLETE.
               If new path is better then replace BEST-
               COMPLETE with new path.
             * else, insert new path in PARTIAL.
5. Go to step 2.
```

Each iteration of the search algorithm consists of executing steps 2,3 and 4. Initially, the *shortest-best* will be the null path. As the search progresses the *shortest-best* path will typically get longer and longer. However, the *shortest-best* path does not necessarily get longer monotonically. This is because *dead* paths are not discarded for ever. The *dead* or *alive* status of each partial path is re-evaluated in each iteration. It is possible for a *dead* path to become *alive* at a later iteration of the search. This is because the *envelope* value $e(t)$ does not necessarily rise from one iteration to the next. An *alive* path which contributes to the *envelope* may later become *dead*, resulting in a lowering of the *envelope*, which in turn may cause *dead* paths to become *alive*.

The search terminates when the *envelope* is identical to the *profile* of BEST-COMPLETE, and there are no *alive* paths within Δ of the *envelope*.

The algorithm presented above allows for language models with unlimited memory. However, speech recognition systems often use finite-state language models with limited memory. The algorithm can be easily modified to take advantage of this to further reduce the search. Path A is said to *dominate* path B if each extension Aw has a higher log likelihood that the corresponding extension Bw. In this case, if path A is in PARTIAL, then path B can be eliminated from further consideration. In our implementation, before inserting a path in PARTIAL, we check to make sure that there is no other path in PARTIAL that ends in the same language model state and has a higher value of $L_{end}(W_1^k)$. If such a path exists, then the new path is not inserted in PARTIAL.

Minor modifications also make it possible to handle acoustic models that incorporate right-context. Initially, when the acoustic match is performed for path W_1^k right context information is not available for words at the end of the path. We can temporarily use a model that ignores right context or assumes some neutral right context. If this path becomes the *shortest-best* path and is chosen for extension, the acoustic match can be re-evaluated in light of the right context provided by each word in the *candidate list*.

Recognition results of a speech recognition system that uses the *envelope* search can be found in Bahl *et al* [10].

4 HISTORICAL NOTES

The origins of heuristic search are shrouded in the mists of time. Some scholars point to an arcane reference in *Genesis*. After all, how did that darned serpent find the forbidden fruit? Would the entire history of mankind have been different if the search had failed? One can only wonder ... However, revisionist scholars claim that the first documented appearance of A* was, in fact, over Bethlehem. Speaking of fruit – we cannot ignore rumors that Isaac Newton discovered an incredibly fortuitous tree search algorithm. Attempts to replicate his results have not always been fruitful, raising seeds of doubt among skeptics. We however, attach little gravity to their comments. Even Hollywood went gaga over heuristic search in *A* Is Born* (Gaynor 1937, Garland 1954, Streisand 1976). Fortunately for the reader, lack of space prohibits us from delving deeper into this subject.

REFERENCES

[1] N. Nilsson, *Problem Solving Methods in Artificial Intelligence*, McGraw-Hill, New York, 1971.

[2] G. D. Forney, Jr., "The Viterbi Algorithm," *Proc. IEEE*, vol. 61, pp. 268-278, 1973.

[3] L. R. Bahl, F. Jelinek and R. L. Mercer, "A Maximum Likelihood Approach to Continuous Speech Recognition," *IEEE Trans. Pat. Anal. and Mac. Int.*, vol. PAMI-5, pp. 179-190, 1983.

[4] F. Jelinek, L. R. Bahl and R. L. Mercer, "Design of a Linguistic Statistical Decoder for the Recognition of Continuous Speech," *IEEE Trans. Inf. Th.*, vol. IT-21, pp. 250-256, 1975.

[5] D. Paul, "An efficient A^* stack decoder algorithm for continuous speech recognition with a stochastic language model," *Proc. DARPA Workshop on Speech and Natural Language*, Harriman, NY, pp. 405-409, 1992.

[6] X. Huang, M. Belin, F. Alleva and M. Hwang, "Unified Stochastic Engine (USE) for Speech Recognition," *Proc. IEEE ICASSP-93*, Minneapolis, MN, pp. II-636-639, 1993.

[7] S. Austin, R. Schwartz and P. Placeway. "The Forward-Backward Search Algorithm," *Proc. IEEE ICASSP-91*, Toronto, Canada, pp. 697-700, 1991.

[8] H. Murveit, P. Monaco, V. Digalakis and J. Butzberger, "Techniques to Achieve an Accurate Real-Time Large-Vocabulary Speech Recognition System," *Proc. Human Language Technology Workshop*, Plainsboro, NJ, pp. 393-398, 1994.

[9] L. R. Bahl, S. V. De Gennaro, P. S. Gopalakrishnan and R. L. Mercer, "A Fast Approximate Acoustic Match for Large Vocabulary Speech Recognition," *IEEE Trans. Speech and Audio Proc.*, vol. 1, pp. 59-67, 1993.

[10] L. R. Bahl, S. V. Balakrishnan-Aiyer, J. R. Bellegarda, M. Franz, P. S. Gopalakrishnan, D. Nahamoo, M. Novak, M. Padmanabhan, M. A. Picheny and S. Roukos, "Performance of the IBM Large Vocabulary Continuous Speech Recognition System on the ARPA Wall Street Journal Task," *Proceedings of this conference.*