VITERBI BASED STACK AND LEXICAL TREE SEARCH METHODS IN SPEECH RECOGNITION

Satya Prakash Bikkina
Department of Electrical Engineering
Mississippi State University, Mississippi State, MS 39759

ABSTRACT

One of the challenges in today's state-of-the-art speech recognition systems is the efficient implementation of decoder or search engine. Though many search algorithms have been developed in an effort to increase the efficiency of the systems, these generally fall under two categories: Viterbi decoding using beam search, or stack decoding which is a variant of the A^* algorithm. This paper describes two search techniques based on these methods. Improvement in recognition speed and considerable memory savings can be achieved while preserving or improving the recognition accuracy.

INTRODUCTION

The main objective of a recognition system is to search for the most likely sequence of words given the input speech. Using Bayesian approach [8] the search problem can be described as

$$W = \arg\max_{W} p(A/W)p(W)$$

The probability $p(A/W)$ that the data A is observed if a word sequence W is spoken is given by acoustic model. The probability $p(W)$ that enumerates the a priori probability of sequence of words is determined by language model. Figure 1 illustrates the basic structure of such a statistical approach. With the introduction of complex language models, very large vocabularies and context-dependent acoustic models, the problem of an efficient search for the most probable sentence became increasingly important. When all possible sequences of words are considered, speech recognition gives rise to an exponential search space. The search cost is highly influenced by the size of vocabulary and is very high in the case of large vocabulary recognition. Two search algorithms: stack search, which is based on Viterbi forward and backward search [2], and lexical tree-based search[1] are discussed in the following two sections. Much emphasis is given to lexical tree-based search since it is the basis to many of the widely used search algorithms in most of the recognition systems.

STACK DECODER

Stack decoding algorithm which is similar to A* search is an implementation of a best–first search. It maintains a stack of partial hypothesis in a breath-first manner [1]. The basic paradigm used by stack decoder is: pop the best partial hypothesis off the stack which are sorted in descending order, apply fast matches to find candidate list of successor words, evaluate log-likelihood. This basic operation of the stack decoder is as shown below:

1. Initialize the stack with NULL.
2. Pop the best partial hypothesis H off the stack.
3. If H is a complete hypothesis, output H & terminate.

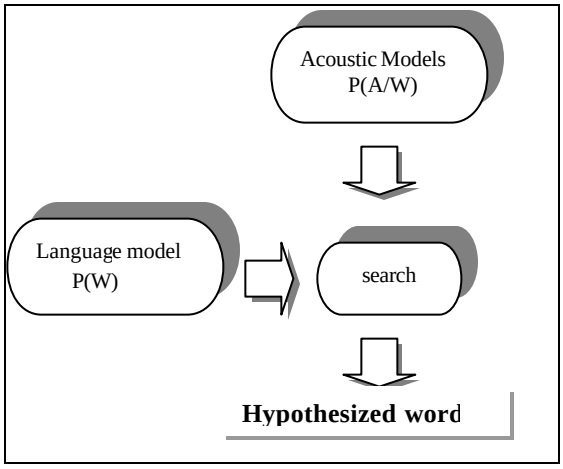

Figure 1. Statistical speech recognition system

4. Else apply acoustic & language –model fast matches to obtain a short list of candidate words.
5. for each word in the list
    a) apply acoustic, LM detailed matches & evaluate new log-likelihoods or partial hypothesis $H^{|}$.
    b) If $H^{|}$ is not complete hypothesis insert into the stack.
    c) If $H^{|}$ is a complete hypothesis insert into the stack with end of sentence flag = true.

The main advantage of stack decoding is its capability of using more complex language models than bigrams LMs, thus increasing the recognition accuracy. The A* or stack search uses the scores & word-segmentations produced by forward & backward viterbi passes. Since, for each word occurrence in the word lattice there are several end times in the case of forward-decoder pass, the scores as a function of time must be contained in the stack. This distribution is the input to the next word model. Since A* algorithm is not a time synchronous algorithm, the partial hypothesis or theory in the stack may account for a different segment of speech. This poses some difficulty to compare the path probabilities. Attaching a heuristic score with every partial theory H, that accounts for the remaining portion of the speech not included in H solves this problem. With this approach the scores in the stack can be compared to one another & can be expanded in a best–first manner. This A* algorithm produces correct result as long as the heuristic score is an upper bound of all possible likelihoods from H [8]. The maximum difference between this upper bound & each partial hypothesis is used to determine the best hypothesis. Hypothesis whose score is less than a threshold is pruned from stack. The backward pass viterbi algorithm provides an approximation to the heuristic score attached to the partial hypothesis. Since the backward pass & the forward viterbi search produces word lattices, the A* is restricted to search words in these two lattices to obtain acoustic probabilities for a wider range of word beginning & end times there by increasing the accuracy. As mentioned above since the heuristic score is only an approximation, the scores stored in the stack may not be in descending order of likelihood. But by choosing a large number of N best hypothesis there is a high probability that the best hypothesis is included. The output with the best score from this list is chosen as the recognized sentence.

LEXICAL TREE SEARCH

The search process makes a decision on the spoken words based on the information obtained from different knowledge sources: the language model, acoustic models, & the pronunciation lexicon. The complexity of the search space depends on the representation used for these knowledge sources. In case of small and medium–sized applications, word–internal, context-dependent acoustic models are found to yield satisfactory results. However in large vocabulary systems such as conversational speech cross-word context dependent models are used. Due to this the search complexity

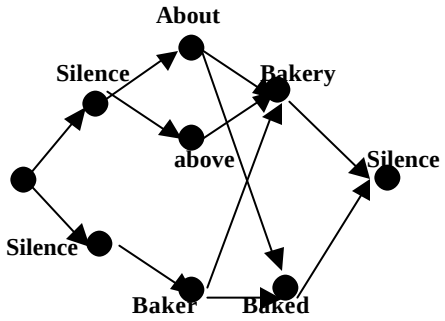

Figure 2: word network.

increases since the last phone in the current word depends on the next word, which is not known until latter in time. Figure 3 illustrates the cross-word network for the word network in figure 2. As can be seen from figure 3, the word end must be hypothesized many times for every possible next word. This increases the size of the network to a large extent. However, even though the number of words is very high, the phonemes used to represent these words are very less. Using this fact, a large amount of computational efficiency can be achieved by using a tree-based lexical search instead of a flat structure. In this approach the lexicon is organized in the form of a tree, in which each arc represents a phoneme model. An example lexical tree is shown in figure 4. Since we wish to use context-dependent phones such as triphones, some changes have to be made to figure 4 so it can handle cross-word triphone models. However with cross-word triphone modeling, the phonetic right contexts are not known since they belong to words that would occur later in time. This leads to a large amount of fan out at the leaves of the lexical tree, since all phonetic probabilities have to be considered. To avoid this problem, a technique known as *dynamic triphone mapping* [7] is used. Here the triphones resulting from different phonetic left contexts are multiplied with the states of the single root HMM. The lexical tree is constructed using a technique known as *dynamic generation of context–dependent phone models* [1]. In this case, context–dependent phones are generated dynamically by traveling the lexical tree nodes. However due to the phone sharing that occurs in the tree-structured lexicon the words does not have identifiable distinct initial states. Hence it poses a difficulty in applying LM score at the instantiation of the very first phone of the word. The identity of the word is known only when the leaf of the tree has been reached, causing a delay in applying correct LM score. A solution to this problem is to use *language model Lookahead* [1] technique. The basic idea in this technique is to incorporate LM probabilities as early as possible, there by reducing the growth in the complexity of search. This is achieved by having LM probabilities at each node that corresponds to the maximum LM probability over all words that can be reached via a particular node. This LM score is combined or attached with the score of the hypothesis. The maximum score that is obtained after this process is used for acoustic & histogram pruning. Once the leaf of the tree is reached, the actual word LM score is added to the path score [1]. Some of the computational advantages that can be obtained because of the use of lexical trees are: The number of words internal HMMs that need to be evaluated are reduced because of the high degree of sharing at the root nodes, the number of cross-word transitions are also reduced to a great extent [7].

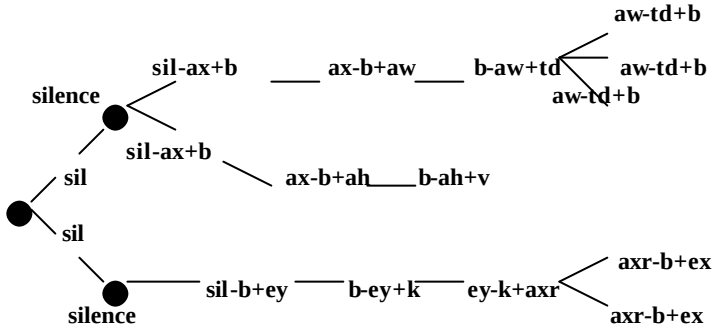

Figure 3: word network expanded using cross-word trigrams.

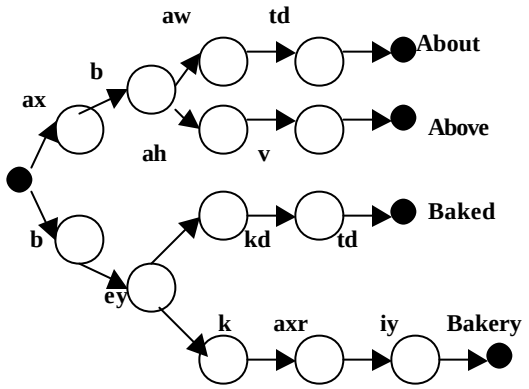

Figure 4: Lexical tree

CONCLUSIONS

Two search techniques that are incorporated into most of the LVCSR system are discussed. Though sub-optimal, these methods showed significant improvements in terms of speed & recognition accuracy.
These techniques still have their own deficiencies, like for example; they have a very high word error rate for conversational speech performed under ideal conditions. Pruning techniques that are used along with these methods are not discussed in this paper. Many variations of these methods can be implemented. For e.g. the tree–based search can be used in a forward–backward implementation, where a simplified lexical tree produces forward hypothesis. The backward pass produces the detail scores & the final word sequence.

REFERENCES

[1] Neeraj Deshmukh, Aravind Ganapathiraju and Joseph Picone, "Hierarchical search for large vocabulary conversational speech Recognition," *IEEE Signal Processing Magazine*, vol. 16, no. 5, pp. 84-107, September 1999.

[2] D.B. Paul, "Algorithms for an Optimal A* Serach and Linearizing the Search in the Stack Decoder," *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, pp.693-696, Toronto, Canada, 1991.

[3] D.B. Paul, "An Efficient A* Stack Decoder Algorithm for Continuous Speech Recognition with a Stochastic Language Model," *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, Vol. I, pp.405-409, San Francisco, California, USA, March 1992.

[4] D.B. Paul, "The Lincoln Large-Vocabulary Stack Decoder Based HMM CSR," *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, pp.374-379, Minneapolis, Minnesota, USA, April 1993.

[5] J. Zhao, J. Hamaker, N. Deshmukh, A. Ganapathiraju and J. Picone, "Fast Search Algorithms for Continuous Speech Recognition," *Proceedings of the IEEE Southeastcon*, pp. 36-39, Lexington, Kentucky, USA, March 1999.

[6] N. Deshmukh, J. Picone and Y.H. Kao, "Efficient Search Strategies in Hierarchical Pattern Recognition Systems," *Proceedings of the 27th IEEE Southeastern Symposium on System Theory*, pp. 89-91, Mississippi State, Mississippi, USA, March 1995.

[7] Mosur K. Ravishankar, "Efficient Algorithms for speech Recognition", Thesis research paper, Carnegie Mellon University, May 1996.

[8] F.Jelinek, "Statistical methods for speech recognition", MIT Press, Cambridge, Massachusetts, USA, 1997.