ABSTRACT

The IBM large vocabulary continuous speech recognition system is based on an asynchronous stack decoding scheme. This is essentially a tree search, as described in [1]. The main advantages - efficient memory utilization and a single-pass search strategy - make the system extremely suitable for real-time applications. This article describes further improvements in efficiency of the search method. These improvements are achieved in part by more efficient word context dependent acoustic model expansion, producing equivalent search results and thus not affecting the recognition accuracy. Additional improvements are achieved by introducing an approximation in the computation of the likelihood of the hypothesized path. The basic idea is to allow sharing of some branches in the search tree and results in effectively a tree to network transformation.

1. INTRODUCTION

Most speech recognition systems today utilize Hidden Markov Models (HMM) [1]. Regardless of the search method, a speech recognizer can be viewed as one huge HMM. A major division can be drawn between synchronous [3] and asynchronous ([4], [5]) search algorithms. The former method finds the best unique HMM state sequence which maximizes the a posteriori probability of the acoustic observation. The Viterbi algorithm is usually used. In the asynchronous scheme, the decoded path can be selected using a Maximum Likelihood criterion, while the exact state sequence is not determined. There is a qualitative difference between these two algorithms, since in the Maximum Likelihood case the final probability can be the result of summation over several paths, not just the best path. This implies potentially higher robustness of this method. In practice, the difference is not usually significant, since the best path has a significantly higher score. The relationship between the methods can be expressed in the approximation:

$$\log(a + b) \approx \max(\log(a), \log(b)).$$  \hspace{1cm} (1)

In fact, picking either side of this expression as the elementary function performed at each node of the HMM time trellis yields either maximum or Viterbi likelihood. For this reason, the Viterbi algorithm is sometimes referred to as an approximation of the maximum likelihood computation.

In either case, the HMM must be organized in a way which allows one to uniquely determine the decoded path, at the required level of detail. For example, if only the word sequence is required as the output, it is not necessary to uniquely determine which particular pronunciation was used, which reduces the complexity of the traceback mechanism. The HMM for Viterbi search can thus be organized as a network with one starting and one ending node. On the other hand, for asynchronous search, tree organization is a logical choice, since the the HMM has to have one ending node for each unique path. For this reason, the complete tree is never pre-built, but constructed dynamically.

2. TREE SEARCH

The IBM recognizer uses an envelope search, a version of A* search [2] which utilizes elements of the time-synchronous search. This is an iterative search. Starting at the root of the tree, in each iteration, one node is selected and the tree is extended. The new nodes created by the extension are inserted into a stack-based structure as new candidates for extension. The envelope search determines:

1. which node is selected for extension
2. which branches will be extended (using an acoustic Fast Match [8])
3. which of the new nodes will be inserted into the stack

Each branch represents a single output level lexical unit (word or a particular pronunciation of the word). For any node (leaf or a non-terminal), we can define the likelihood of the i-th path, consisting of n words, $$p_i^{(n)} = (w_0^{(i)}, w_1^{(i)}, \ldots, w_n^{(i)})$$ from the root of
the tree up to that node:

\[ L(p_i^{(n)}) = \sum_{\tau = t_{left}}^{t_{right}} \text{Prob}(x_0, \ldots, x_\tau, p_i^{(n)}) \]  

(2)

where \( x = (x_0, \ldots, x_\tau) \) is the sequence of observed acoustic feature vectors. Since the path is completely modeled by an HMM, this likelihood can be computed as:

\[ L(p_i^{(n)}) = \sum_{\tau = t_{left}}^{t_{right}} \alpha_{p,m,d}(\tau), \]

(3)

where \( \alpha_{p,m,d} \) is the forward probability of the final node of the HMM representing the path \( p_i^{(n)} \). The interval \([t_{left}, t_{right}]\) denotes the part of \([0, t_{max}]\) interval in which the probabilities are greater than zero after thresholding is applied.

Since we can express the likelihood of a path as a joint likelihood of the last branch and its parent path:

\[ L(p_i^{(n)}) = L(w_n^{(i)}, p_i^{(n-1)}), \]  

(4)

we can express the likelihood of a single branch:

\[ L(w_n^{(i)}|p_i^{(n-1)}) = \frac{L(p_i^{(n)})}{L(p_i^{(n-1)})}. \]  

(5)

Then the likelihood of the path can be written as:

\[ L(p_i^{(n)}) = L(w_n^{(i)}|p_i^{(n-1)})L(w_n^{(i)}|p_i^{(n-2)}) \ldots L(w_0). \]  

(6)

Each factor of this product represents the likelihood contribution of each branch and is in fact the result of the word's HMM likelihood computation (Detailed Match) performed at each iteration of the search.

We should always keep in mind that although the likelihood of the branch is always associated with a certain time interval, it is not directly a function of time. This makes application of time-synchronous methods difficult. Rather than working with a time interval, it is better to associate the likelihood with a single time value. Let us define the most likely boundary time as:

\[ p_{mb}^{(n)} = \arg \max \text{Prob}(x_0, \ldots, x_n, p_i^{(n)}). \]  

(7)

To show this association explicitly, we can write the likelihood as \( L(p_i^{(n)}, t_{mb}) \), which means likelihood of the path \( p_i^{(n)} \) ending most likely at time \( t_{mb} \). The pair of values \( (L(p_i^{(n)}), p_{mb}^{(n)}) \) is important for the search process, but we should keep in mind that the actual value of \( t_{mb} \) has no effect on the final likelihood of the decoded path (assuming no search errors are made). For the further extension of the path \( p_i^{(n)} \), we need the distribution of the actual ending time, called the end time distribution, defined as:

\[ P(t_{end}|p_i^{(n)}) = P(x_t|p_i^{(n)}) = \frac{P(x_t,p_i^{(n)})}{L(p_i^{(n)})}. \]  

(8)

where \( x_t \) is a sequence of acoustic feature vectors from time 0 to \( t \).

The efficiency of the algorithms (exhaustive search would be exponential in time) depends heavily on the pruning method. The pruning method decides which branches are unlikely to be part of the final best path and are not worthy of further extension. This pruning is somewhat an equivalent of the Viterbi decision in the time-synchronous search. The difference is that the comparison is done asynchronously, so it has to be done more carefully. In the envelope search, the comparison is made between the probability of the path \( p_i^{(n)} \) and the envelope value at the time \( p_{mb}^{(n)} \), which is an estimate of the best-path probability (details in [6]).

A typical way to avoid redundant computation in the Viterbi framework is to rearrange the tree into a network, with a trace-back mechanism needed to track the final best path. In our proposed method, the asynchronous search keeps the concept of a search tree at the word level, while representing the tree as a network and merging paths at the acoustic model level. This is possible because we are able to express the path likelihood as a product of factors which are not directly dependent, the dependency is limited to the immediate neighbors, or error caused by the actual dependency is very small. This virtual merge is shown in figure (1). Under certain conditions (which will be stated later), the two paths \( p_i^{(n-1)} \) and \( p_j^{(n-1)} \) can be merged using a Viterbi decision, so only one of them is actually extended. The likelihood of the extending HMM can then be used to update the likelihood of each path, so the both paths can be treated again as independent.

The likelihood of an extending branch (word) is determined by a context dependent HMM. The context dependency is practically limited to the preceding word (assuming left cross-word context dependency):

\[ L(w_n^{(i)}|p_i^{(n-1)}, t_{mb}^{(n-1)}) \approx \]

\[ L(M(w_n^{(i)}, w_{n-1}^{(i)}, t_{mb}^{(n-1)})) = \]

(9)

The dependence of the branch likelihood on its parent \( p_i^{(n-1)} \) is replaced by the dependence on the immediate predecessor only and its ending time \( t_{mb} \). The word model \( M \) is the actual HMM machine, which is constructed in the particular cross-word context of \( w_{n-1} \). The core of proposed speed up method lies in
the association of the likelihood of \( w_n \) with a discrete time value of \( t_{mb}^{w_n} \), i.e. the ending time of its predecessor. It is possible to reuse the value of \( P(M(w_n, w_{n-1}), t_{mb}^{w_{n-1}}) \) in any other part of the tree (path \( p_j^{(n)} \)), as long as:

\[
M(w_n^{(i)}, w_{n-1}^{(j)}) = M(w_n^{(j)}, w_{n-1}^{(j)}) \tag{10}
\]

and

\[
t_{mb}^{w_{n-1}} = t_{mb}^{w_n^{(i)}} \tag{11}
\]

It should be emphasized that it is not necessary to satisfy \( w_n^{(i)} = w_{n-1}^{(j)} \) for the constructed HMM models to be identical, because this model depends on the decision trees. The decision tree asks questions about 5 phones to the left and 5 phones to the right, word boundary symbol treated as a phone. The condition (11) represent an approximation used in this method, since it does not guarantee identical likelihood results. The likelihood depends on the whole end-time distribution (8). But the practical results show that the negative effect of this approximation is negligible while computational saving can be substantial (more than 40 % in the HMM computation). This is not surprising, since the difference between the two end-time distributions will be very small at the \( t_{mb} \) itself, but this point contributes to the final likelihood with the highest weight.

![Figure 1: Virtual merge of the phonetic layer](image)

3. ALGORITHM DESCRIPTION

The new algorithm can be described with the help of figure (2). At time \( t_{mb}^{p_a} \) there are two paths on the stack which can be extended, \( p_a \) and \( p_b \). The search algorithm generates a list of candidates (using a Fast Match and Language Model [8]), first for the path \( p_a \). For each of these candidates, the corresponding HMM state sequence is found (each state represents a context dependent sub-phonetic unit). From this set of state sequences, a tree is built, with a corresponding leaf for each word in the candidate list. The tree is then passed to the Detailed Match to obtain the likelihoods \( L(w_n^{(i)}, t_{mb}^{w_n}) \). When the second path, \( p_b \), gets extended, its HMM state tree may be different. First, the candidate list will differ due to a different LM context, second, the HMM state sequence may differ due to a different cross-word context. So each state sequence in this new list is compared against the existing DM tree. If a proper leaf is found, the result of the DM can be used to extend this path (gray nodes in the search tree in figure 2). In case the leaf is not found, the DM tree is augmented (the dashed branch) and the new nodes are evaluated by the DM. The same procedure can be applied to all extension at the same time \( t_{mb}^{p_a} \).

4. RESULTS

The algorithm was tested on a research prototype of the IBM speech recognition system. The system uses different acoustic models for sub-phonetic units in different contexts. These instances of context dependent classes are identified by growing a decision tree from the training data. The acoustic feature vectors that characterize the training data at the leaves are modeled by a mixture of Gaussian pdf's, with diagonal covariance matrices (a total of 30k Gaussian were used). As far as the output distributions on the state transitions of the model are concerned, rather than expressing the output distribution directly in terms of the feature vector, the IBM system expresses it in terms of the rank of the context dependent class [7]. The acoustic front end uses a FFT based filter bank followed by cepstral rotation. Frame energy and dynamic parameters \( (\Delta + \Delta \Delta) \) were added to each feature vector. Sentence based cepstrum mean normalization was used.

The test set consists of 61 sentences each read by 10 speakers. The decoder runs in a speaker independent mode. The following table shows the recognition error rates and achieved speed improvements.

<table>
<thead>
<tr>
<th></th>
<th>Error rate</th>
<th>DM calls</th>
<th>DM time[s]</th>
<th>Total time[s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>11.17</td>
<td>460k</td>
<td>79.5</td>
<td>174</td>
</tr>
<tr>
<td>Speedup</td>
<td>11.16</td>
<td>289k</td>
<td>46.8</td>
<td>146</td>
</tr>
<tr>
<td>Improv</td>
<td>37 %</td>
<td>37 %</td>
<td>41 %</td>
<td>16 %</td>
</tr>
</tbody>
</table>

As can be seen, significant reduction of both Detailed Match (DM) calls and overall DM CPU time was achieved, causing a negligible change to the word error rate (in this test, and quite often in other tests, we have actually seen slight improvements!).

5. CONCLUSION

We have presented a new method which significantly increases efficiency of the asynchronous tree search. The ability to reuse segments of a path to extend
other paths of the searched tree can be seen as advantage of an the asynchronous decoding scheme combined with synchronous elements, compared to a strictly time-synchronous based scheme.

6. REFERENCES


