Confidence measure driven scalable two-pass recognition strategy for large list grammars

Miroslav Novak

IBM T.J. Watson Reseach Center
Yorktown Heights, NY 10598
miroslav@us.ibm.com

Diego Ruiz *

Université catholique de Louvain
Laboratoire de Télécommunications et Télédéction
1348 Louvain-La-Neuve, Belgium
ruiz@tele.ucl.ac.be

Abstract

In this article we will discuss recognition performance on large list grammars, a class of tasks often encountered in telephony applications. In these tasks, the user makes a selection from a large list of choices (e.g. stock quotes, yellow pages, etc). Though the redundancy of the complete utterance is often high enough to achieve high recognition accuracy, large search space presents a challenge for the recognizer, in particular, when real-time, low latency performance is required. We propose a confidence measure driven two-pass search strategy, exploiting the high mutual information between grammar states to improve pruning efficiency while minimizing the need for memory.

1. Introduction

On a typical automatic speech recognition (ASR) telephony platform one processor handles several recognition channels; so recognition speed has a direct impact on the hardware cost. Our proposed method reduces the average recognition CPU cost per utterance for the price of a small amount of tolerable latency.

ASR systems for telephony applications commonly use finite state transducers (FST), also called grammars, as language models. For many applications, such as digit strings, stock names and name recognition, the grammars are relatively easy to design. In this article, we focus on applications of this type, where the grammar simply enumerates all possible choices. In particular we are interested in the cases when the number of choices is large (thousands or more). We are not going to address natural language applications with complex grammars or N-gram based language models.

To make a clear distinction between a grammar and an N-gram based model (as used in large vocabulary ASR), we will always assume that a grammar assigns non-zero probability to only a limited set of possible word sequences. This property has a positive effect on both accuracy and search speed, as ASR with grammars tends to have higher recognition accuracy in comparison to N-gram based language models. The price paid is the inability of a grammar based system to handle situations when the speaker deviates from the exact utterances specified by the grammar.

As the size of the task grows, the search becomes more challenging. Though the overall word perplexity of the task can be low, the problem is that the perplexity varies significantly during the search; the number of legal word choices differs significantly from one grammar state to another. This makes a recognition system prone to search errors, especially if single pass real-time recognition is required. Pruning strategies developed for general large vocabulary recognition, in general, do not provide optimal results.

In this article we will further focus on the implications for search in the context of an asynchronous decoder. The IBM speech recognition system uses an envelope search [1], derived from A^* tree search. For this search to be admissible, one needs to be able to find, given a particular incomplete path, an upper bound on the likelihood of the remaining part of this path. If this upper bound is overestimated, the search is non-optimal.

In general, for large vocabulary ASR we assume that the context of any partial path has only a short range effect (basically given by the N-gram span), so the cost of finishing a particular path till the end of the utterance will be similar (within some difference δ) to the cost of any other partial path ending around the same time. This assumption allows us to use the likelihood of the best path at that time as the A^* estimate, the δ is thus used to trade between admissibility and optimality of the search.

This assumption is clearly inappropriate when a grammar is used, for a partial path with a high likelihood in the middle of an utterance the search may not find any legal ending at all. Thus a reliable estimate of the cost of the remaining path is difficult to find, without investigating the acoustic features all the way until the end of the utterance.

For this reason, the search needs to be much wider at

*The second author performed the work while at IBM T.J. Watson Research Center

the beginning of an utterance, where perplexity is usually the highest. It would also be useful to know about the rest of the utterance when the pruning decision is made.

	Stock name	Name dialer	e-mail
Vocabulary	8040	30000	103
$H(W_f)$	11.24	12.9	4.24
Perp(W_f)	2508	7623	19
$H(W_f\|W_l)$	5.03	2.16	3.02
$I(W_f;W_l)$	6.27	10.74	1.22

Table 1: Entropy of the first word in the utterance

Table 1 shows the entropy $H(W_f)$ of the first word in the utterance for three tasks with different vocabulary sizes. The first two tasks fall into the category of large lists. For comparison, the third task is also shown. It is a simple e-mail client application, which can be described as a command and control type of task.

It can be clearly seen that in the case of the large list tasks the entropy of the first word W_f conditioned on the last word W_l of the utterance is significantly lower than the unconditioned entropy. There is high mutual information between the first and last word of the utterance, suggesting that knowledge about the end of the utterance would be very beneficial for search efficiency.

However, if we want to utilize such knowledge in a single-pass synchronous search, which provides the results with practically zero latency, this is the least suitable choice.

Use of multiple-pass search strategies [2],[3] would seem like a better choice; using a cheaper and wide open forward pass followed by tight and precise backward pass. But this introduces an inherent latency into the system. The cheaper the first pass, the more expensive is the second pass and the higher the latency. Another problem with a multiple-pass strategy is that the memory requirements for storing the results of the first pass can be significant.

2. The search algorithm description

The technique we will now describe is a variation of a two pass search strategy. An important feature of our approach is that we use the most accurate model during the first pass. To minimize the latency caused by the second pass (and memory need as well), we try to perform as much of the search work as possible in the first pass and minimize the cost associated with the second pass. The second pass is performed only if there is an indication that a search error occurred in the first pass.

The algorithm consists of the following steps:

• Perform the standard single pass search with a suboptimal search setting and store the intermediate search results.

• Based on a confidence measure applied to the recognized utterance, make a decision if a search error is likely to have occurred.

• Compute information needed to speed up the second pass.

• Perform the second pass.

By using a suboptimal search we understand the use of aggressive pruning techniques which will produce an unacceptable number of search errors. These errors are then corrected by the second pass.

For the presented technique to work efficiently, it is essential to use a search technique which allows the results of the first pass to be stored efficiently and to produce new search hypotheses in the second pass.

We will first describe the baseline search algorithm in detail. The IBM recognizer uses a multi-stack (one stack for each time) envelope tree search. The main components of the decoder are: fast match, detailed match and language model (grammar). It is an iterative search: starting after the initial silence match at the beginning of an utterance, with each iteration an incomplete path is selected for extension. The fast match [4] is called first to obtain a list of possible words for extensions with corresponding scores. After combining the fast match scores with the language model scores, a shorter list of candidates for the detailed match is created. The detailed match then evaluates these candidates so that new nodes of the search tree can be created and inserted into the corresponding stacks.

The selection of the right time stack for a new path is based on the "most likely boundary" time of the new hypothesis. It is important to note that this time is a discrete value, but an actual stack entry represent the whole interval of possible word endings with corresponding likelihoods.

There are several parameters which affect the search speed. The most important ones are:

• Envelope distance δ- equivalent of the beam width in Viterbi beam search. It is used to determine if a path should be extended or discarded. The envelope is constructed from the best state likelihoods observed at each time.

• Detailed match list size - limits the number of word extensions evaluated for each path.

Since we assign a unique boundary time to each incomplete path, the time-stack can be relatively sparse. The acoustic fast match uses context independent models that can be shared across all paths ending at the same time. We need to call fast match only at times for which the stacks are not empty. Typically the most expensive is the fast match at the beginning of the utterance where the perplexity is the highest. As the tree search progresses, the number of words the fast match needs to evaluate in subsequent calls is quickly reduced due to the grammar constraints. Saving the results of the first fast match call for later use in the second pass is inexpensive.

If the fast match produces list of candidates which is greater than some fixed limit, then only the top candi-

dates are selected to be processed by the detailed match. This is an effective way of pruning, as the fast match is allowed to look ahead as much as one second. Once this list is passed to the detailed match, time synchronous pruning can be used locally. Significant savings can also be achieved by caching the detailed match results for later use in the same acoustic context [5].

The standard algorithm ends when no path for extension can be found (all paths are either complete or dead). The complete path with the best likelihood is selected as the decoded one.

In our proposed algorithm, we evaluate a confidence measure to determine if there is no better solution that was pruned away by the search. Many confidence techniques can be found in the literature. For example, approaches based on word *a posteriori* probabilities computed from the word graphs are popular [6]. We have ruled out this approach because our word lattice is not dense enough in the presence of search errors. Our preference is for an inexpensive technique which can be tuned to provide a very low false acceptance rate. False rejections are much less costly in terms of error rate. It is important to note that our objective is different than finding if the sentence was correctly recognized, we use the confidence measure to asses the possibility of a search error. A detailed study of confidence measure techniques is not the main focus of this work though, we have experimented with several heuristic features and decided to use the following ones:

• Average frame likelihood of the decoded path, including normalization components of the likelihood computation. This normalization forces the likelihood of the correct path to be a roughly a linear function of time. A search error typically causes much lower likelihood for the path.

• Relative fast match score of the first word

$$S(W) = \frac{P_{fm}(W)}{\sum_{W' \in \mathcal{V}} P_{fm}(W')}, \quad (1)$$

where $P_{fm}(W)$ is the likelihood (not log likelihood) of the word based on the fast match. The first fast match call provides a list of all possible first words, so any complete path will contain one word from this list in the first position. This relative score can be viewed as an approximation of the first word *a posteriori* probability. The higher the score, the lower the chance that some other word will assume the first position in the path. We have discovered that this score is indeed a good predictor of search errors.

The decoded path is labeled as search error free (accepted) if either one of these measures is above some threshold. If the decoded path is rejected, we perform the second pass. If the second pass is performed, any computation we do should not be very expensive, because it adds to the latency of the whole system. To obtain a list of candidates for the last word, we call the fast match once, in reversed direction, from the end of the utterance (after the final silence match in reversed direction). We cannot really use the search results of the first pass to compile the list, it is quite likely that due to the search errors the correct words would be missing. This fast match call represents the single most expensive part of the second pass.

We now combine the fast match candidates from the utterance beginning computed during the first pass and the fast match candidates from the end of the utterance. Only some combinations are legal (as defined by the grammar), we sort these pairs by their combined log likelihoods

$$S(W_f, W_l) = \log P_{(forward)}(W_f) + \log P_{(backward)}(W_l) \quad (2)$$

The ranking of the candidates for the first word can now be significantly different from the ranking based on the forward match only. So we can revisit the list of detailed match candidates from the first pass. Starting with the top candidate in this new list, we check if it was already processed during the first pass. If not, we add it to the new list. We stop once the number of the added words reaches a certain limit. The rest of the search is basically the same as in the first pass, but new paths can be pruned more efficiently thanks to the search envelope built during the first pass.

3. Results

Our experiments were conducted on a telephony system. Cepstral coefficients were generated at a 15ms frame rate with overlapping 25ms frames. Nine frames are spliced together and linearly-transformed and projected using LDA+MLLT into a 39 dimensional feature vector. A cross-word left-context pentaphone acoustic HMM model is built with 1080 states and 160000 Gaussians.

The computation of HMM state probabilities is limited to the top 256 best states at each time frame. We store the probabilities in memory for the whole utterance, so they are available during the second pass. Instead of using the Gaussian mixture probabilities directly, we convert them to probabilities based on their rank when sorted by GMM probability.

The results are shown in figure 1 for the stock name task and in figure 2 for the name dialer task. The grammar contains 25 thousands choices for the stock names and and 86 thousand choices for the name dialer. In both cases, the average utterance length is 2.9 words.

The speed is represented by a ratio between the total duration of utterances and total CPU time consumed by the decoder. We prefer this form because it is directly correlated to the number of decoders which can run on one CPU concurrently.

We have considered the first task (stock name) as a development set, to explore a wide variety of parameter settings and chose the optimal ones. In particular, the confidence measure threshold was selected for this task. The second test set was then used to verify the robustness of the selected parameters.

The solid curve shows the sentence recognition error rate of the baseline (single pass) system when the value of the detailed match list is varied from 40 to 400. The dotted line shows the performance of the two pass system when the second pass is always performed. To achieve a visible speed improvement, we had to choose a relatively small DM list size for the first pass. Otherwise, the second pass only slows the system without contributing to any accuracy improvement. For the second pass, we varied the list size from 20 to 100. It can be seen that the overhead of the second pass can eliminate the speed improvement. The most significant part of this overhead is the computation of the reversed fast match. Only when we used the confidence measure to avoid the second pass, we have achieved a noticeable improvement (dashed line). Similar behavior is observed observed for the name dialer task. The error rate is slightly higher due to imperfections in the confidence measure.

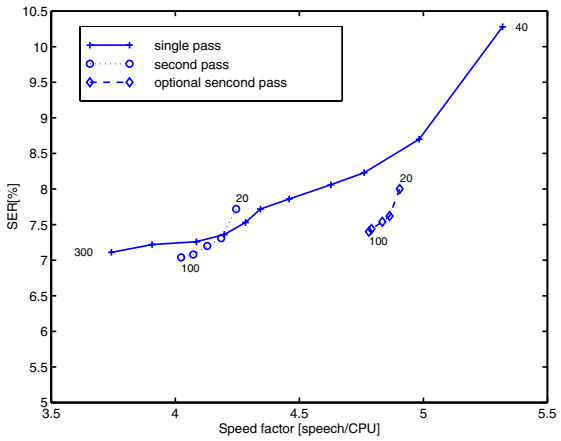

Figure 1: Speed/error rate dependency on the stock name task

On the name dialer task, the second pass search was performed on 56% of all utterances in the test set. The actual search time attributed to the second pass represents 28% of the total decoding time. The average latency was 0.12 seconds per utterance, across all utterances. When we consider only those utterances for which the second pass was computed, the average latency is 0.2 seconds.

4. Conclusion

The presented two pass search algorithm allows us to improve the speech recognition performance in telephony applications by trading a tolerable latency for reduced average CPU cost per utterance. Although the use of this technique is limited to specific category of tasks, these tasks are commonly encountered. This method can be easily generalized to extend the range of its application. The approach described here can be used whenever a grammar state with high mutual information between its

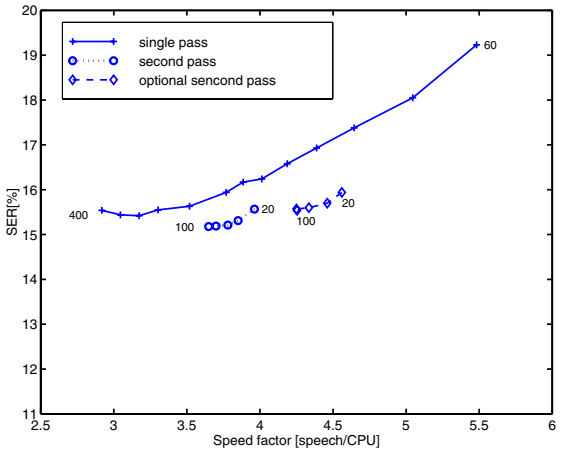

Figure 2: Speed/error rate dependency on the name dialer task

outgoing arcs and incoming arcs of the final state exists, with some modification it could be used between any two states of a grammar.

5. Acknowledgment

We would like to thank Benoît Maison and Vaibhava Goel for their help during Diego Ruiz's stay at T.J. Watson Research Center.

6. References

[1] P.S. Gopalakrishnan, L.R. Bahl, and R.L. Mercer, "A tree search strategy for large vocabulary continuous speech recognition," in *Proc. ICASSP '95*, May 1995, pp. 572–575.

[2] S. Austin, R. Schwartz, and P. Placeway, "The forward-backward search algorithm," in *Proc. ICASSP 1991*, April 1991, vol. 1, pp. 697 – 700.

[3] M. Novak, R. Hampl, P. Krbec, V. Bergl, and J. Sedivy, "Two-pass search strategy for large list recognition on embedded speech recognition platforms," in *Proc. ICASSP 2003*, Hong Kong, April 2003.

[4] 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 Transactions on Speech and Audio Processing*, vol. 1, no. 1, pp. 59–67, January 1993.

[5] M. Novak and M. Picheny, "Speed improvement of the tree-based time asynchronous search," in *Proc. ICSLP 2000*, Beijing, November 2000, pp. 334–337.

[6] F. Wessel, R. Schlueter, K. Macherey, and H. Ney, "Confidence measures for large vocabulary continuous speech recognition," *IEEE Transactions on speech and audio processing*, vol. 9, no. 3, pp. 288–298, March 2001.