TWO-PASS SEARCH STRATEGY FOR LARGE LIST RECOGNITION ON EMBEDDED SPEECH RECOGNITION PLATFORMS

M. Novak, R. Hampl, P. Krbec, V. Bergl, J. Sedivy

IBM T.J. Watson Research Center, P. O. Box 218, Yorktown Heights, NY 10598, USA

ABSTRACT

This paper presents an efficient algorithm for a speech recognition system which can process large lists of items. The described two-pass search implementation focuses on maximizing the speed and minimizing the memory footprint of the search engine. The algorithm is designed to handle thousands or tens of thousands of words in a search space restricted by a grammar. A typical example of such a task is stock name recognition, street name finding, song selection etc. The intended application of this algorithm is in embedded ASR system in portable devices (e.g. iPAQ) or cars.

1. INTRODUCTION

Speech recognition applications implemented on low-resource platforms have received a lot of attention recently. The task complexity is limited by the available resources, such as the processing power of the CPU and available memory. The application's memory footprint is more critical since the targeted platforms typically try to extend battery life by equipping devices with smaller dynamic memory. Thus, many such recognizers target small vocabulary tasks with search spaces limited by grammars.

This paper presents an algorithm which can be used for large vocabulary speech recognition on existing portable devices such as the Compaq iPAQ. We focus on a particular class of tasks, which we refer to as "large list" tasks. The main characteristics are large vocabulary (several thousand words) and search space restricted by an acyclic finite state machine (FSM). The acyclic restriction is essential for our implementation, as it allows us to represent the whole search space as a tree.

This functionality is useful for many applications where the user is selecting from a large list of choices. These are the kinds of applications where speech recognition is bringing the largest benefit to the user interface. A typical example of such a task is stock name or mutual fund name selection, street name finding, song selection etc. An utterance is usually very short, so the user does not need any feedback during the utterance, as long as the answer is ready soon after the utterance is finished.

2. SEARCH STRATEGY

Recognition of large lists has been addressed in the context of using network search space [1]. We will show that we can take advantage of tree search by using it during the first pass of a two-pass search.

The decoder is based on synchronous Viterbi search. We have ruled out an asynchronous stack search approach [2] due to its relative complexity of implementation in comparison to the Viterbi search. For the level of complexity observed in embedded tasks, the advantage of stack search scalability is not apparent. Nevertheless, we have borrowed some concepts from stack search to implement the "large list" system.

We will first discuss the available options for search implementation. We will then present the reasons why we choose our two-pass search strategy and compare it to alternative solutions.

Recent developments in the application of finite state transducers to speech recognition have made the choice of single-pass Viterbi very popular [3] by providing a sound framework for operations such as weighted automata determinization, minimization, factorization etc. The search network can be either statically pre-compiled or composed dynamically with some additional CPU cost.

Our initial experiments showed that a statically compiled network would not allow us to perform single pass search with sufficient speed, so we have decided to use two-pass strategy. In the first pass, the search is performed on a static network using approximate models over the whole utterance. This "fast match" generates a list of n-best hypotheses, from which a smaller "detailed match" network is compiled.

This arrangement introduces certain latency into the system, since the detailed match phase is not started until the utterance is completed. But since the detailed match phase is much faster and we compute and store the detailed match state output probabilities during the fast match phase, this latency is very small.

Another reason for adopting the two pass strategy is the sensitivity of the targeted task to search errors. The perplexity of the search at the beginning is very high, possibly equal

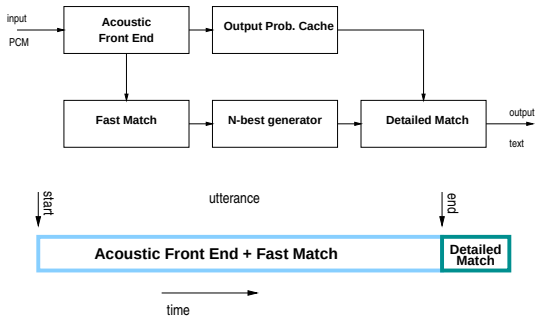

Fig. 1. Block and time diagrams of the two pass search strategy

to the number of words in the vocabulary. At that point, the correct path can be easily pruned off by another path which might have a better score at a particular time but may never reach the end of the utterance. Knowing the score of the whole phrase gives us significantly more information about the first word than just the score of this word alone. Thus, delaying the detailed match pass until the fast match is completed over the whole utterance results in a much sparser word trellis for the detailed match.

3. ACOUSTIC FRONT END

The role of the front end is to compute the HMM state output probabilities using Gaussian mixture models. We use decision trees to create 700 GMMs, each one corresponding to one HMM state in a tri-phone context. The context information is limited to word internal phones only. Rather than computing these probabilities on demand (an efficient approach for small vocabularies), we evaluate only a limited number of GMMs using a hierarchical scheme [4] so that the CPU cost is not dependent upon vocabulary or grammar size. Our two-pass strategy requires storage of these values for the whole length of the utterance, by changing the number of GMMs computed and stored we can trade accuracy for memory usage.

Instead of using the GMM probabilities directly, we convert them to probabilities based on their rank when sorted by GMM probability [5]. One particular advantage is that the probability space is then bounded and the values of the best and the worst state output probabilities remain the same for each time frame. This allows us to prune the search more aggressively. For any state output probabilities not actually computed, we simply use a single value from the tail of the rank probability distribution. Another advantage of using rank based probabilities is that their dynamic range is more comparable to the range of state transition probabilities, so their effect on the duration modeling capability of the HMM is more apparent.

4. FAST MATCH

For the fast match pass, an FSM graph representing the search space at the word level is expanded into a phonetic tree. We take advantage of a tree property: each leaf of the tree uniquely identifies a matched phrase. There is no need for back-tracing: the score of each hypothesis can be read at the leaves of the tree. This has an important implication for both memory and CPU usage.

To analyze these costs carefully, let us start with the tree. The memory cost can be divided into three distinct categories - ROM, statically allocated RAM and dynamically allocated RAM. This distinction is important on certain hardware platforms. In particular, the management of dynamically allocated RAM may represent an additional cost. All information which does no need to be changed during the runtime can be stored in a ROM. This is the static information associated with each node of the tree and links between the nodes. Statically allocated RAM is needed only for the state likelihoods, which are updated at each time frame. Because the tree search does not need to save back-pointers, there is no need for dynamically allocated RAM. The CPU cost is mainly in the update of the state likelihoods. The cost of traversing the tree can be minimized by factoring the tree into a set of linear state sequences (i.e. with no branching), which can be more effectively stored in memory in terms of locality of access.

As an alternative to the tree search, a network search could also be considered. Here, a network is obtained by minimizing the tree, which typically leads to a smaller network in terms of states, but not necessarily in transitions between states. The minimization process essentially merges the common word and phrase suffixes. We have observed that factorization of the minimized network produces shorter state sequences, which reduces the gain in both memory and CPU cost obtained by the factorization. Furthermore, the CPU cost and memory need is increased, because not only the state likelihood need to be updated, but also the trace-back pointer as well. The costs are further increased if there is a requirement that n-best list or a word trellis containing the n-best scoring hypothesis has to be found as well. The memory related to the trace-back needs to be dynamically allocated.

In the actual implementation we do not really use a strict tree structure. We start with a tree/network hybrid, illustrated in Figure 2. On the word level (*A,B,C,D,E*), it is still a tree, but all the pronunciation variants are locally merged into a network. The final determinization step then merges common phonetic prefixes across different words. This approach significantly reduces the tree size while maintaining all benefits of the tree search. One drawback is the inability to find the exact pronunciation of the best path, so the final n-best list will include all pronunciations. We do not

see it as a significant disadvantage, since in most cases the pronunciation variants of the same word are similar and it is likely that they would be included in the n-best list anyway.

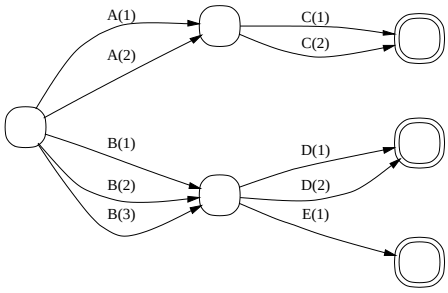

Fig. 2. The word-tree/pronunciation-network structure of the Fast Match

The fast match pass uses simplified phone models [6], derived from detailed match models. The HMM topology of each phone in the tree consists of a single node with one self-loop arc and one forward arc. The state output probabilities are computed as the maximum observation probability over all states corresponding to the modeled phone.

$$P(\boldsymbol{o}|phone) = \max_{\boldsymbol{s}\in\mathcal{S}_{phone}} P(\boldsymbol{o}|\boldsymbol{s}) \qquad (1)$$

To improve the ability of such simple structure to capture the phonetic duration we use state transition probabilities trained on these models. In addition (to enforce the minimum duration), we allow the transition from one phone to another every third time frame only. This has a significant benefit in terms of speed, because it allows us to merge the fast match state output probabilities into triplets and perform the Viterbi search with effectively one third of the time frames only.

To control the speed of the search, we use standard beam pruning techniques. To form the beam, we use the maximum of the state likelihoods observed in the previous time frame rather than the current time frame. This arrangement allows us deactivate the states with low likelihood immediately after they are updated for the current frame. The last frame maximum is indeed a good predictor of current frame maximum, since the best state output probability in each frame is constant when the state output probabilities are rank based.

In addition to the beam pruning, we use an n-best active phone strategy in the fast match pass. Only those phones which are considered active use the actual observation probability, for the rest we use some default low probability value. This technique helps us to deactivate states with low likelihood much earlier. The n-best phones can be found at no additional cost, since the use of rank based probabilities requires sorting anyway.

5. DETAILED MATCH

The role of the detailed match is to search a word lattice constructed from the n-best list provided by the fast match using the accurate state observation probabilities. Both the lattice building and the search must be very fast, since they are performed only after the utterance is finished. There is no need to compute the state output probabilities , since they were stored in memory during the first pass. If the fast match models are accurate enough, only relatively few hypotheses need to be passed to the detailed match.

The lattice construction is quite efficient. By selecting the n-best leaves of the fast match tree, we define a subtree to be processed by the detailed match. This subtree is already determinized on the phonetic level, each phone is then expanded into context dependent states using decision trees. This expansion is fast due to the use of word internal tri-phone context.

6. RESULTS

We present our results on two platforms. The development platform is a PC with Intel Pentium 4, clock speed 1700MHz with 500 MB of RAM. The OS is Windows 2000.

The second platform is a Compaq iPAQ with the following hardware specifications: CPU ARM SA1110 206 MHz Intel StrongARM 32-bit RISC, memory 63.14 MB RAM, OS Microsoft Pocket PC Version 3.0.11171.

The tasks contains a list of 1475 song titles. The vocabulary has 3699 unique word pronunciations. The test set contains 1376 utterances, one song name in each utterance.

First, we have demonstrated that our decision to use a tree search was right. On the same task, we tried both tree search and the network search. The results are shown in table 1 for the fast match pass only. The speed factor (faster/RT) shows how many times is the decoding time shorter than the duration of speech. This factor is more illustrative than the more conventional real-time factor. Neither the timing nor the memory includes the additional cost of n-best list construction.

	faster/RT	memory[MB]
Tree	39.6	2.06
Network	15.34	2.14

Table 1. Comparison between the tree search and network search

Figure 3 shows the tradeoff between speed and accuracy controlled by the number of n-best phrases processed by the detailed match. The speed factor depicts how many times the two-pass system (fast match and detailed match) is faster than running the single pass detailed match search

only on the whole grammar, using the same beam search width.

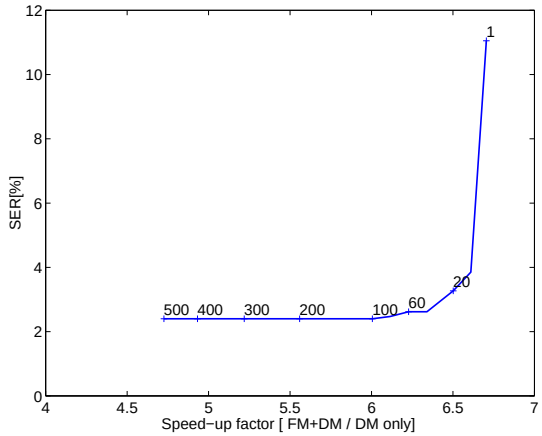

Fig. 3. Dependency of the sentence error rate and speed on the detailed match list size on Intel

Figure 4 shows the speed and accuracy measured on the iPAQ. The solid curve represents the total time when the second pass latency is not included. From a user's point of view, more important is the dashed curve showing the timing of the second (detailed match) pass only. It can be clearly seen that this pass represents only a fraction of the CPU cost and the perceived latency will be very small (our experience shows that the utterances are almost always shorter than 3 seconds).

Table 2 shows the comparison between single-pass and two-pass search with the 100 best hypotheses processed in the second pass. The comparison shows the sentence error rate, how many times is the system slower than real-time and the memory footprint.

	SER	slower/RT	memory[MB]
single-pass	2.33%	4.88	3.0
two-pass 100-top	2.40%	0.56	2.2

Table 2. Comparison between the single-pass and two-pass search

7. CONCLUSION

We have shown that a specific large vocabulary task can be implemented on today's portable devices such as the iPAQ. We have presented the reasons for choosing the two pass tree/network search strategy for the large list tasks. Our results confirmed our expectations.

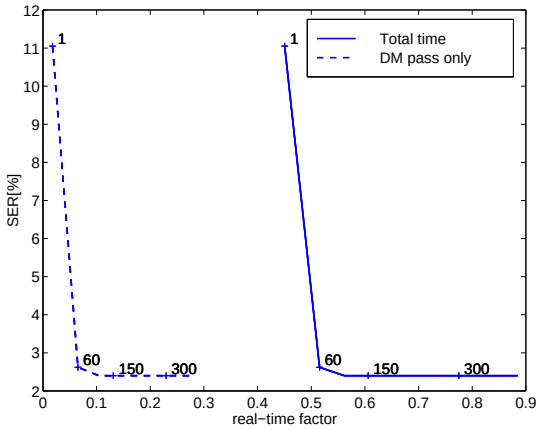

Fig. 4. Dependency of the sentence error rate and speed on the detailed match list size on iPAQ

8. REFERENCES

[1] K. Hanazawa, Y. Minami, and S. Furui, "An efficient search method for large-vocabulary continuous-speech recognition," in *Proc. ICASSP '97*, 1997, pp. 1787 – 1790.

[2] 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.

[3] M. Mohri, F. Pereira, and M. Riley, "Weighted finite-state transducers in speech recognition," *Computer Speech & Language*, vol. 16, no. 1, pp. 69–88, January 2002.

[4] M. Novak, R. A. Gopinath, and J. Sedivy, "Efficient hierarchical labeler algorithm for gaussian likelihoods computation in resource constrained speech recognition systems," *http://www.research.ibm.com/people/r/rameshg/novak-icassp2002.ps*.

[5] L.R. Bahl P.V. de Souza, P.S. Gopalakrishnan, D. Nahamoo, and M.A. Picheny, "Robust methods for using context-dependent features and speech recognition models in a continuous speech recognizer," in *Proc. ICASSP '94*, 1994.

[6] 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.