A FAST VOCABULARY INDEPENDENT ALGORITHM FOR SPOTTING WORDS IN SPEECH

S. Dharanipragada *S. Roukos*
IBM T.J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598
Email: dsatya@watson.ibm.com

ABSTRACT

In applications such as audio-indexing, spoken message retrieval and video-browsing, it is necessary to have the ability to detect spoken words that are outside the vocabulary of the speech recognizer used in these systems, in large amounts of speech at speeds many times faster than real-time. In this paper we present a fast, vocabulary independent, algorithm for spotting words in speech. The algorithm consists of a preprocessing stage and a coarse-to-detailed search strategy for spotting a word/phone sequence in speech. The preprocessing method provides a phone-level representation of the speech that can be searched efficiently. The coarse search, consisting of phone-ngram matching, identifies regions of speech as *putative* word hits. The detailed acoustic match is then conducted only at the putative hits identified in the coarse match. This gives us the desired accuracy and speed in wordspotting. Overall, the algorithm has a speed of execution that is 2400 times faster than real-time.

1. INTRODUCTION

Most current audio-indexing systems use a combination of speech recognition and information retrieval. A large vocabulary continuous speech recognition system is used to produce time aligned transcripts of the given collection of speech. Information retrieval techniques are then employed on these recognized transcripts to identify locations in the text that are relevant to the search request. One limitation with this approach to audio-indexing is the finite coverage of the vocabulary used in the speech recognizer — words such as proper nouns and abbreviations that are important from an nformation retrieval standpoint are often found missing in the vocabulary and hence in the recognized transcripts. One approach to overcome this limitation is to complement the speech recognizer with a wordspotter for the out of vocabulary (OOV) words. For this approach to be practical, however, one requires the ability to detect spoken words in large amounts of speech at speeds many times faster than real-time.

Current wordspotting techniques are mainly of three types:

1. A large vocabulary continuous speech recognizer is used to produce N best transcriptions of the speech. From the N-best lists, the *a posteriori* probability of the word in question, is estimated. If this probability exceeds a user-defined threshold, the given word is deemed present. [1]
2. Separate acoustic models are built for background speech and keywords and a network consisting of these models in parallel is constructed. The network is updated time-synchronously using the standard Baum-Welch algorithm and the *a posteriori* probability of the keyword ending at any given time is computed. If this probability exceeds a user-defined threshold, the given word is deemed present. [2]
3. Speech is preprocessed and stored as a phone lattice by running a modified Viterbi decoder on a null-grammar phone network. The presence of a given word is determined by conducting a dynamic programming search on the phone lattice with penalties for phone insertions, deletions and substitutions. [3]

All the above methods have their shortcomings. In the the first method, wordspotting essentially reduces to searching through text. Consequently, retrieval is fast. However, it has the serious shortcoming that words that do not appear in the vocabulary of the speech recognizer cannot be spotted. The second method has no limitations on the words that can be searched for, but is very slow since it requires re-running the wordspotter every time a new word is specified and hence is not practical. The third method has both the flexibility of being able to search for any word and speed of retrieval. However, it relies heavily on phone recognition accuracy which is often very poor. In this paper we describe a novel method for wordspotting that has the same flexibility as the lattice-based approach but is much faster and more accurate. We accomplish this by adopting a three-step procedure – a preprocessing step and a two stage search strategy. In the preprocessing step we convert the speech waveform into a representation consisting of a table of phone-ngrams with the times at which they occur with a high likelihood. This representation allows us to search through the speech very efficiently. The two stage search consists of first a phone-ngram lookup to narrow down the time intervals where the word was likely to have been uttered and then a detailed acoustic match at these time intervals to finally decide more accurately whether the word was actually uttered in that time interval.

2. THREE STAGE ALGORITHM FOR WORDSPOTTING

The algorithm consists of three steps: (1) preprocessing (2) coarse acoustic match and (3) detailed acoustic match.

2.1. Pre-processing:

The preprocessing step consists of converting the speech into a table consisting of times of occurrence and normalized likelihood scores of phone-ngrams (usually triphones) in the given speech. This is achieved by a time-synchronous Viterbi-beam search which uses a phone language model organized as tree to constrain the dynamic programming search.

Phonetic baseforms of the words in a large vocabulary are arranged in the form of a tree, called the fast match tree, which is then converted into a graph capable of representing arbitrary sequences of words by adding arcs which go from each leaf back to the root node. Transition probabilities are assigned to each arc as $1/N$ where N is the number of arcs leaving the source node. A self-loop probability of 0.5 is also assumed for each node. This graph is used to constrain a dynamic programming search and in effect plays the role of a phone language model in the search.

Phones attached to each arc of the fast match tree are represented either by a three-state context dependent HMM or a context independent single state HMM to reduce computation. Our initial experiments used a single state topology for each phone. When a single state topology is used, each node of the fast match tree directly corresponds to a state in the trellis and in order to maintain the desired three-frame minimum duration, transitions only every third frame are allowed.

The trellis is updated time-synchronosly using a standard Viterbi-beam search algorithm [4]. At every third frame, the scores of all the *active* nodes are updated and the identities of the top M nodes are noted. Here M is a user defined parameter. Each node in the trellis represents a sub-word or a phone sequence which is be determined by traversing the tree from the root node to that node. The active nodes thus signify the top triphones that best describe that local region of speech. For each active triphone, $\pi = p_1p_2p_3$, a normalized log-likelihood score is computed as

$$s_t(\pi) = \frac{logScore(S_e^{p_3}, t) - logScore(S_b^{p_1}, t_s) - ns(ts, t)}{t - t_s},$$

where $logScore(s, t)$ is the Viterbi score of state s at time t and $S_e^{p_3}$ and $S_b^{p_1}$ are the exit and entry states of p_3 and p_1 respectively. $ns(ts, t)$ is a normalization score computed using the phone probabilities between t_s and t. To reduce storage requirements, the time axis is discretized into T second intervals, where T is a user defined parameter, and all active triphones are binned into these intervals. A table of triphones versus their times of occurrence in the given speech along with their acoustic scores, is thus generated. For example, if the time axis is discretized into 1 sec intervals and triphones N EH DX, EH DX AX, and DX AX N were in the beam in the intervals [21 22][31 32], [21,22][51 52], and [31 32][40 41] respectively, then the table will appear as:

triphone	time	(secs)	
N EH DX	21	31	...
EH DX AX	21	51	...
DX AX N	31	40	...
⋮			

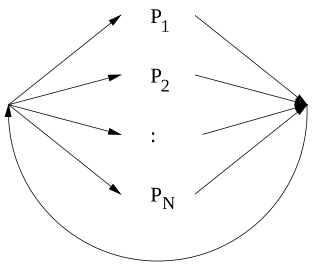
2.2. Coarse acoustic match for putative hits:

Given a new word, we determine its phonetic transcription (baseform) using either a large dictionary of baseforms or a spelling to baseform generation mechanism[1] [5]. The baseform specifies the triphones that need to be looked up from the triphone-times table. Let N_b be the number of these triphones. For each of these triphones we look-up, from the table generated in the preprocessing step, the times where the triphone was found in the speech. For each time-interval where more than a certain fraction of the total number of triphones is found, a crude acoustic score is computed for the word as

$$s_t(w_n) = \sum_{i=1}^{N_b} s_t(p_i), \qquad (1)$$

where $s_t(p_i)$, the acoustic score for each phone in the baseform, is determined using the normalized log-likelihood scores of the triphones that are found in that time interval and the beam-width used in the Viterbi-beam search of the pre-processing stage. This acoustic score is used to rank all the putative hits (time intervals).

[1] One could even prompt a user to provide the pronunciation.



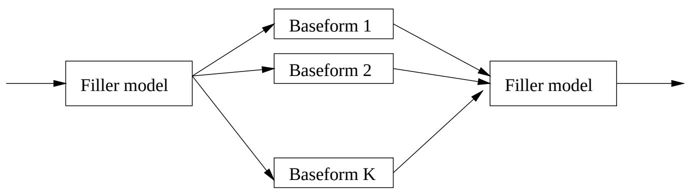


Figure 1. Network used in the detail match.

2.3. Detailed Match at the putative hits:

In order to reduce the number of false alarms a more detailed match is performed at the regions classified as putative-hits in the previous stage. To speed up the detailed acoustic match a simple two step approach is taken. First, a network consisting of "mumble" or "filler" models and all the alternative baseforms for the given word is constucted as shown in Fig 1. For each putative hit, the best path through this network is computed using a Viterbi algorithm with traceback and the start and end times, t_s , t_e, for the word are determined. Next, the filler model and each of the alternative baseforms are scored between the start and end-time determined in the previous step and the duration normalized log-likelihood ratio (DNLLR) is computed as follows:

$$DNLLR = \frac{logScore(S^w_e, t_e) - logScore(S^f_e, t_s)}{t_e - t_s},$$

where $logScore(s,t)$ is the Viterbi score of state s at time t and S^w_e, S^f_e are the exit states of the word and filler HMMs respectively. All putative hits are ranked based on the best DNLLR amongst the alternative baseforms.

3. EXPERIMENTS

Experiments were conducted on the HUB4 corpus which consists of approximately 100 hours of broadcast news. Acoustic models were trained using the first 50 hours of data. From the remaining 50 hours, 18 shows totalling to 10 hours of broadcast news was selected as a test set. From the transcripts of these shows 38 words that were out of the vocabulary used in the pre-processing stage were selected as a test set.

3.1. Acoustic models

Acoustic models were trained using the first 50 hours of the HUB4 data. Context-dependent sub-phone classes are identified by growing a decision tree using the WSJ training data and specifying the terminal nodes of the tree as the relevant instances of these classes [8, 9, 10]. Overall, the decision tree had 5700 leaves. The first 50 hours of the HUB4 training data is poured down this tree and 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. Each leaf of the decision tree is modeled by a 1-state Hidden Markov Model with a self loop and a forward transition. Overall the system has approximately 170,000 Gaussians. Output distributions on the state transitions are expressed in terms of the rank of the leaf instead of in terms of the feature vector and the mixture of Gaussian pdf's modeling the training data at the leaf. The rank of a leaf is obtained by computing the log-likelihood of the acoustic vector using the model at each leaf, and then ranking the leaves on the basis of their log-likelihoods. To update the scores in the pre-processing stage and in the detail match we require the probabilities of the phones attached to these nodes. We compute the probability of each phone as the maximum probability over all context-dependent sub-phonetic units of that phone – this gives us context independence [7].

3.2. Wordspotting Performance

Experiments were carried out both with handcrafted baseforms and baseforms generated using spelling to sound rules. Fig. 2 shows the trade-off between the false-alarms per keyword per hour (fa/kw-hr) and the detection rate for the algorithm. The figure-of merit (FOM) for this algorithm was computed by averaging the detection rates at false alarm levels ranging from 0 to 10 and is shown in Table 1. We observed a 13.6% degradation in FOM when automatically generated baseforms were used in place of handcrafted baseforms.

In a real retrieval situation, one is more interested in the performance of the algorithm when the absolute number of false-alarms is 10 or less. This corresponds to a fa/kw-hr level of 1 in Fig. 2 since we have 10 hours of test data. From Fig. 2 the detection rate at this level is 44.2% when hand-crafted baseforms are used for the new words. We believe that this detection rate is quite

	FOM
Dict. baseforms	54.3%
Auto. baseforms	46.9%

Table 1. Figure of Merit for the algorithm with handcrafted and automatically generated baseforms.

high given that it is achieved on words that are not part of the acoustic training vocabulary and also since the spotting algorithm does not employ a language model at any stage.

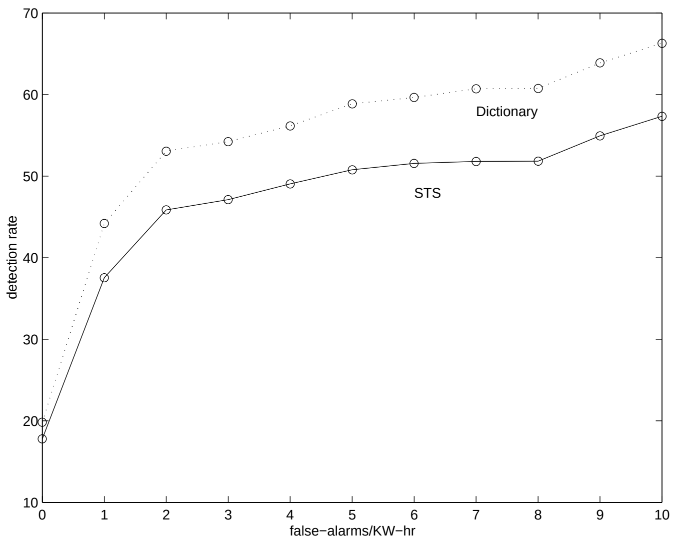


Figure 2. ROC curve for the wordspotting algorithm with automatically generated (Spelling-To-Sound rules) baseforms and with handcrafted (dictionary) baseforms

3.3. Speed of execution

The main feature of this algorithm is its speed of execution once the pre-processing is done. With our current implementation, searching through 10 hours of speech requires approximately 15 seconds of CPU time for a keyword on an IBM model 590 RS6000 workstation. This translates to a speed of execution that is approximately 2400 times faster than real-time.

3.4. CONCLUSIONS

We presented a novel and efficient algorithm for the problem of spotting arbitrary keywords in large amounts of speech at speeds several times faster than real-time. The algorithm is currently being integrated with a large vocabulary continuous speech recognizer and an information retrieval system to obtain an open vocabulary audio-indexing system. Refinements to the algorithm and implementation are currently in progress and we hope to achieve further improvements on the speed and accuracy of wordspotting.

REFERENCES

[1] M. Weintraub. “LVCSR Log-Likelihood Ratio Scoring For Keyword Spotting”,ICASSP 1995, Vol 1, pp 297-300.

[2] J.R. Rohlicek, W. Russel, S. Roukos, H. Gish. “Word Spotting”, ICASSP 1989, pp627-630.

[3] D.A. James, S.J. Young. “A Fast Lattice-Based Approach To Vocabulary Independent Wordspotting”, ICASSP 1994, pp 377-380.

[4] G.D. Forney, Jr. “The Viterbi Algorithm,” Proc. IEEE, vol 61, pp 268-278. 1973.

[5] J. M. Lucassen and R. L. Mercer. “An Information Theoretic Approach to the Auto matic Determination of Phonemic Baseforms”, in Proceedings of the IEEE Internat ional Conference on Acoustics, Speech and Signal Processing, pp. 4 2.5.1-42.5.4, 1984.

[6] P.S. Gopalakrishnan, L.R. Bahl, R.L. Mercer. “A Tree Search Strategy for Large-Vocabulary Continuous Speech Recognition.” ICASSP 1995, vol 1, pp 572-575.

[7] L.R. Bahl et al. “Performance of the IBM Large Vocabulary Continuous Speech Recognition System on the ARPA Wall Street Journal Task.” ICASSP 1995, vol 1, pp 41-44.

[8] L.R. Bahl and P.V. deSouza and P.S. Gopalakrishnan and D. Nahamoo and M.A. Picheny, “Robust methods for context-dependent features and models in a continuous speech recognizer,” in *Proc., Intl Conf. on Acoust., Speech, and Sig. Proc.*, 1994.

[9] P.S. Gopalakrishnan and L.R. Bahl and R. Mercer, “A tree search strategy for large vocabulary continuous speech recognition,” in *Proc., Intl Conf. on Acoust., Speech, and Sig. Proc.*, 1995.

[10] L. R. Bahl et al ., “Performance of the IBM large vocabulary continuous speech recognition system on the ARPA wall street journal task,” in *Proc., Intl Conf. on Acoust., Speech, and Sig. Proc.*, pp. 41–44, 1995.