Indexing and search methods for spoken documents [*]

Lukáš Burget, Jan Černocký, Michal Fapšo, Martin Karafiát, Pavel Matějka, Petr Schwarz, Pavel Smrž, and Igor Szöke [**]

Speech@FIT, Faculty of Information Technology, Brno University of Technology, Božetěchova 2, 612 66 Brno, Czech Republic
`speech@fit.vutbr.cz`, `http://www.fit.vutbr.cz/speech/`

Abstract. This paper presents two approaches to spoken document retrieval – search in LVCSR recognition lattices and in phoneme lattices. For the former one, an efficient method of indexing and search of multi-word queries is discussed. In phonetic search, the indexation of tri-phoneme sequences is investigated. The results in terms of response time to single and multi-word queries are evaluated on ICSI meeting database.

1 Introduction

It is very likely that today's success of Google in text search will excite interest in searching also other media. Among these, search in speech is probably the most interesting, as most of human-to-human communication is done by this modality. We can imagine applications for example in meeting processing and eLearning. Search in recordings is also becoming more important with the new US legislation on record-keeping, and there are many security and defense related applications. These techniques are often referred to as "keyword spotting", this term however implies that only one keyword can be searched at a time. Our approach allows for more complex queries, therefore, we prefer to rank it rather under spoken document retrieval (SDR).

Unlike search in text, where the indexing and search is the only "science", SDR is a more complex process that needs to address the following points:

- conversion of speech to discrete symbols that can be indexed and searched – large vocabulary continuous speech systems (LVCSR) and phoneme recognizers are used. Using phoneme recognizer allows to deal with out-of-vocabulary words (OOVs) that can not be handled by LVCSR.

[*] This work was partially supported by EC project Augmented Multi-party Interaction (AMI), No. 506811 and Grant Agency of Czech Republic under project No. 102/05/0278. Lukáš Burget was supported by post-doctoral grant of Grant Agency of Czech Republic No. 102/06/P383.

[**] The authors are listed in alphabetical order, Jan Černocký `cernocky@fit.vutbr.cz` is the corresponding author.

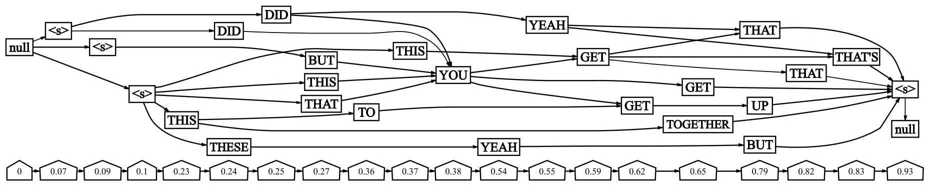

Fig. 1. Example of a word lattice

- accounting for inherent errors of LVCSR and phoneme recognizer – this is usually solved by storing and searching in word, respectively phoneme lattices (Fig. 1) instead of 1-best output.
- determining the confidence of a query – in this paper done by evaluating the likelihood ratio between the path with searched keyword(s) and the optimal path in the lattice.
- processing multi-word queries, both quoted (exact sequences of words) and unquoted.
- providing an efficient and fast mechanism to obtain the search results in reasonable time even for huge amounts of data.

In this paper, we do not deal with pre-processors such as LVCSR system and phoneme recognizer, but concentrate on indexing and search issues. Section 2 reviews the LVCSR-based search with confidence computation and indexing. Section 3 details the technique used for two- and multi-word queries. The phonetic search is covered in section 4 with a tri-phoneme approach to indexing described in section 5. Section 6 presents the experimental results in terms of index sizes and response-times evaluated on 17-hour subset of ICSI meeting database.

2 LVCSR-based search

LVCSR lattices (example in Fig. 1) contain nodes carrying word labels and arcs, determining the timing and acoustic (L_a^{lvcsr}) and language model (L_l^{lvcsr}) likelihoods generated by an LVCSR decoder. Usually, each speech record is first broken into segments (by speaker turn or voice activity detector) and each segment is represented by one lattice. The confidence of a keyword KW is given by

$$C^{lvcsr}(KW) = \frac{L_\alpha^{lvcsr}(KW) L^{lvcsr}(KW) L_\beta^{lvcsr}(KW)}{L_{best}^{lvcsr}}, \qquad (1)$$

where the $L^{lvcsr}(KW) = L_a^{lvcsr}(KW) L_l^{lvcsr}(KW)$.

The forward likelihood $L_\alpha^{lvcsr}(KW)$ is the likelihood of the best path through lattice from the beginning of lattice to the keyword and the backward likelihood $L_\beta^{lvcsr}(KW)$ is the likelihood of the best path from the keyword to the end of lattice. For node N, these two likelihoods are computed by the standard Viterbi formulae:

$$L_\alpha^{lvcsr}(N) = L_a^{lvcsr}(N) L_l^{lvcsr}(N) \max_{N_P} L_\alpha^{lvcsr}(N_P) \quad (2)$$

$$L_\beta^{lvcsr}(N) = L_a^{lvcsr}(N) L_l^{lvcsr}(N) \max_{N_F} L_\beta^{lvcsr}(N_F) \quad (3)$$

where N_F is a set of nodes directly following node N (nodes N and N_F are connected by an arc) and N_P is a set of nodes directly preceding node N. The algorithm is initialized by setting $L_\alpha^{lvcsr}(first) = 1$ and $L_\beta^{lvcsr}(last) = 1$. The last likelihood we need in Eq. 1: $L_{best}^{lvcsr} = L_\alpha^{lvcsr} = L_\beta^{lvcsr}$ is the likelihood of the most probable path through the lattice.

The **indexing** of LVCSR lattices is inspired by [1]. It begins with the creation of lexicon which provides a transformation from word to a unique number (ID) and vice versa. Then, a forward index is created storing each hypothesis (the word, its confidence, time and nodeID in the lattice file) in a hit list. From this index, a reverse index is created (like in text search) which has the same structure as the forward index, but is sorted by words and by confidence of hypotheses.

Each speech record (ie. meeting) is represented by many lattices. The reverse index tells us, in which lattice the keyword appears and what is it's nodeID in this particular lattice.

In the **search** phase, the reverse index is used to find occurrences of words from query. An important feature of our system is the generation of the most probable **context** of the found keyword – a piece of the Viterbi path from the found keyword forward and backward. For all matching occurrences, the searcher therefore loads into the memory a small part of lattice within which the found word occurs. Then, the searcher traverses this part of lattice in forward and backward directions selecting only the best hypotheses; in this way it creates the most probable string which traverses the found word.

3 Multi-word queries

A usable system for SDR should support queries of type

`word1 word2 word3` and `"word1 word2 word3"`

with the former one representing finding words in random order with optional spaces in between (in opposite to text-search where we work within a document, we specify a time-context) and the later one representing the exact match. Provided the query Q is found in the lattice, we again need to evaluate its confidence $C^{lvcsr}(Q)$. Similarly to Eq. 1, this is done by evaluating the likelihood of the path with all the words w_i belonging to the query and dividing it by the likelihood of the optimal path:

$$C^{lvcsr}(Q) = \frac{L_{rest}^{lvcsr} \prod_i L^{lvcsr}(w_i)}{L_{best}^{lvcsr}}, \quad (4)$$

where L_{rest}^{lvcsr} is the likelihood of the "Viterbi glue": optimal path from the beginning of the lattice to $w_{earliest}$, connections between words, w_i (for unquoted

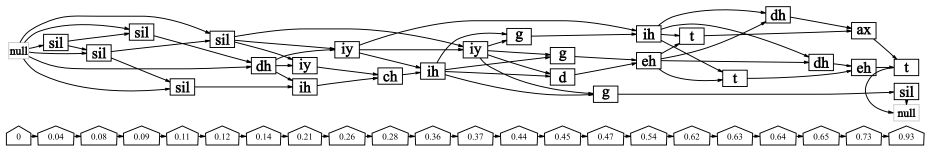

Fig. 2. Example of a phoneme lattice

query) and optimal path from w_{latest} to the end of the lattice. In other words L_{rest}^{lvcsr} represents everything except the searched words. We should note, that each time we deviate the Viterbi path from the best one, we loose some likelihood, so that $L^{lvcsr}(Q)$ is upper-bounded by $\min_i C^{lvcsr}(w_i)$ — actually the confidence of the worst word in the query.

The same index as for single-word queries (keywords) is used here. Processing of a query involves the following steps:

1. Based on frequencies of words, the least frequent one from the query, w_{lf}, is taken as first and all its occurrences are retrieved.
2. The search proceeds with other words and verifies if they are within the specified time interval from w_{lf} (for non-quoted queries) or joint to w_{lf} (for quoted ones). The internal memory representation resembles again a lattice. In such way, a candidate list is created.
3. The list is pre-sorted by the upper-bound of query confidence, as described above. The list is then limited to the pre-determined number of candidates (usually 10).
4. For these candidates, the evaluation of correct confidence is done according to Eq. 4. While looking for the "Viterbi glue", the Viterbi algorithm is extended before and after the part of lattice containing Q in order to obtain the left and right contexts.

4 Phonetic search

The main problem of LVSCR is the dependence on recognition vocabulary. The phonetic approach overcomes this problem by conversion of query to string and searching this string in a phoneme lattice (Fig. 2). The lattice has similar structure as word lattice (section 2), phonemes P populate nodes instead of words.

The confidence of keyword KW consisting of string of phonemes $P_b \dots P_e$ is defined similarly as in Eq. 1 by:

$$C^{phn}(KW) = \frac{L_{\alpha}^{phn}(P_b) L_{\beta}^{phn}(P_e) \prod_{P \in P_b \dots P_e} L_a(P)}{L_{best}^{phn}}, \qquad (5)$$

where $L_{\alpha}^{phn}(P_b)$ is the forward Viterbi likelihood from the beginning of lattice to phoneme P_b, the product is the likelihood of the keyword, and $L_{\beta}^{phn}(P_e)$ is the likelihood from the last phoneme till the end of the lattice. L_{best} is the likelihood of the optimal path. As phoneme recognition is done without language (phono-tactic) model, the language model likelihoods are replaced by a constant – phoneme insertion penalty (PIP). It plays a role in the computation of $L_{\alpha}^{phn}(P_b)$, $L_{\beta}^{phn}(P_e)$ and L_{best}^{phn} and does not intervene in the product giving the likelihood of the keyword. The value of PIP needs to be tuned. The experiments of Szöke et al. [3] have shown that in case the phoneme lattice is dense, it is sufficient to look for an exact match of the searched string and not to take into account substitution, insertion and deletion errors.

5 Indexing phoneme lattices

While the indexing of word lattices is straightforward, indexing phoneme lattices is more tricky: in advance, we do not know what we will search for. Yu and Seide in [7] and Siohan and Bacchiani in [8] have chosen indexing sequences of phonemes with variable length, we have however investigated a simpler approach making use of overlapping tri-phonemes and indexing similar to multi-word queries. The use of tri-phonemes was also recommended in [6] as the best balance between number of units and number of units' occurrences in a corpus.

In the indexing phase, tri-phonemes T_i are selected in lattices. For each T_i, its confidence is evaluated by Eq. 5 as if T_i was a keyword. In case this confidence is higher than a pre-determined threshold, the tri-phoneme is inserted into the index.

The search stage consists of the following steps:

1. The searched keyword generates a set of overlapping tri-phonemes. Based on their frequencies in the index, the least frequent one T_{lf}, is taken as first and all its occurrences are retrieved.
2. The search proceeds with other tri-phonemes and verifies that they form a chain in time (with a security margin between adjacent tri-phonemes). Similarly to multi-word queries, the internal memory representation has again the form of lattice. In such way, a candidate list is created.
3. The confidence of keyword is again upper-bounded by the confidence of the worst tri-phoneme. Based on these, the list is pre-sorted and limited to the pre-determined number of candidates (usually 10).
4. For these candidates, we go into the respective phoneme lattices and evaluate the correct confidence using Eq. 5.

We have verified, that in case no thresholds are applied in the index, we obtain exactly the same accuracy of search that in case phoneme lattices are processed directly.

6 Experiments

The evaluation was done on 17 hours of speech from ICSI meeting database [4]. Attention was paid to the definition of fair division of data into training, development and test parts with non-overlapping speakers. We have also balanced the ratio of native/nonnative speakers and balanced the ratio of European/Asiatic speakers.

LVCSR lattices were generated by AMI-LVCSR system [2] and phonetic lattices were generated by a phoneme recognizer based on long-temporal context features with a hierarchical structure of neural nets [5].

The accuracies of different approaches were evaluated by Figure of Merit (FOM), which approximately corresponds to word accuracy provided that there are 5 false alarms per hour in average. In LVCSR-search, the FOM was 67% while for the phoneme-lattice search, we reached FOM of 60%. Detailed results are discussed in [3] – in this experimental evaluation, we have concentrated on response times, and disk footprints that are crucial for real deployment of the system.

The size of audio is 1.8 GB. The number of LVCSR lattices representing this audio is 25815 and they occupy 600 MB. LVCSR index needs 130 MB. Phoneme lattices (branching factor 4) need 2.1 GB of disk and the tri-phoneme index requires 220 MB.

In all tests, we report average time to process one query. The number of hits was set to 10-best in all experiments. The context to retrieve in LVCSR queries was set to ± 10 words and ± 7.5 seconds (whichever is shorter). The processing was done on a AMD Athlon 3200+. We made sure that the data to be searched (lattices, indexes) resided on the local hard-disk and that no other CPU/memory consuming processes run on the machine.

The first test in LVCSR-search aimed at single keywords. Two sets were defined: `Test17` containing 17 frequent words and `Test1` containing words occurring just once in the test set. The total number of different words in `Test1` is 2310, but only 50 were used in these evaluations.

The following test aimed at 2- till 4-word *quoted* queries. We have randomly chosen sequences of 2 to 4 words from the transcriptions of the test set and made sure at least one word within each sequence is at least 5 characters long. Examples of such sequences are:
2: `"A MATTER"`, `"NOUN PHRASES"`
3: `"THE DETECTOR TO"`, `"PERSON TO DO"`
4: `"BUY A TICKET OR"`, `"THE SITUATION OF LETTING"`
50 sequences of each length were selected. These tests are denoted `Quoted2 ... Quoted4`.

In the test of unquoted queries, all tested sequences contained only words with length ≥ 5 characters and we worked again with 2- till 4-word sequences (note that for unquoted sequence, the words can appear in any order). The context (or "document size") was set to 20 words. To define the sets of queries, we have divided the test set into windows containing 10 words, discarded windows with less than 10 words and selected one sequence satisfying the word-length constraint from each window. Then, these sequences were randomized and 50 were selected for each length. Examples of such sequences are:
2: `RELEVANT RANGES`, `WEDNESDAY ACTUAL`
3: `PERSON LISTENING FIRST`, `STUDY RIGHT GERMANY`

Test	time per query [s]
`Test17`	0.8
`Test1`	0.2
`Quoted2`	9.6
`Quoted3`	33.0
`Quoted4`	34.0
`Unquoted2`	1.2
`Unquoted3`	1.3
`Unquoted4`	1.8

Table 1. The results of LVCSR-based search.

Test	time per query [s]
`Test17`	10.5
`Test1`	9.3

Table 2. The results of phonetic search.

4: `TEACHER QUALITY THERE COURSE`, `TRAIN MODELS SUBTRACTION USING`
These tests are denoted `Unquoted2 ... Unquoted4`.

Table 1 summarizes the response times for LVCSR-based search.

In the tests of phonetic search, only single keywords from sets `Test17` and `Test1` were looked for. Measurement of response times were done on the same 17h test-set, the results are summarized in Table 2.

We see that in LVCSR search is very fast and that single word and multiple-word unquoted queries require only 1-2 seconds. It is very likely that these figures will extend well to bigger archives. The times required for quoted queries are quite prohibitive and we need to suggest optimizations. One of first targets will be the C++ STL library that is used for the creation of the internal lattice structures and which is quite slow.

The response times of phonetic search are longer than in LVCSR, but the search is still usable for the given size of archive. We should note that the comparison of response times for `Test17` and `Test1` is inverse for LVCSR and phonetic search. This is explained by the nature of the two algorithms: LVCSR can take advantage of rarity of words in `Test1` – they simply appear less frequently in the index so that the processing is faster. On contrary, phonetic search of items from `Test1` takes almost the same time as `Test17`, as this approach can do no difference between rare and frequent words (actually, it does not have a notion of "word" in both indexing and search).

7 Conclusion

We have presented several techniques of indexing and search in LVCSR and phonetic lattices for spoken document retrieval. They were evaluated on real meeting data from ICSI meeting database. In LVCSR, both one-word and multi-word queries are handled with fast response times, the processing of quoted queries still needs some investigation. In phonetic search, we have verified the functionality of indexing tri-phones derived from phoneme lattices, but speeding up is needed also here.

In our further research, we will investigate direct techniques to derive tri-phoneme indices without lattices - tri-phonemes can actually be seen as keywords and as such pre-detected by a standard acoustic keyword spotting and indexed. We will also investigate the importance of different tri-phonemes for indexing and search and suggest customized pruning thresholds to keep the index size manageable. Finally, our goal is to build and test a system combining LVCSR and phonetic search allowing to search multi-word queries with OOVs.

References

1. Sergey Brin, Lawrence Page: *The Anatomy of a Large-Scale Hypertextual Web Search Engine, Computer Science Department*, Stanford University, 1998.
2. T. Hain et al.: The 2005 AMI system for the transcription of speech in meetings, in *Proc. Rich Transcription 2005 Spring Meeting Recognition Evaluation Workshop*, Edinburgh, July 2005.
3. Igor Szöke et al.: Comparison of Keyword Spotting Approaches for Informal Continuous Speech, in *Proc. Eurospeech 2005*, Lisabon, Portugal, September 2005.
4. A. Janin and D. Baron and J. Edwards and D. Ellis and D. Gelbart and N. Morgan and B. Peskin and T. Pfau and E. Shriberg and A. Stolcke and C. Wooters: The ICSI Meeting Corpus, in *Proc. ICASSP-2003*, Hong Kong, April 2003
5. P. Schwarz, P. Matějka, J. Černocký: Hierarchical structures of neural networks for phoneme recognition, accepted to ICASSP 2006, Toulouse, France, May 2006.
6. K. Ng: *Subword-Based Approaches for Spoken Document Retrieval*, PhD thesis, Massachusetts Institute of Technology, USA, February 2000.
7. P. Yu and F. Seide: Fast two-stage vocabulary independent search in spontaneous speech, in *Proc. ICASSP 2005*, Philadelphia, 2005.
8. O. Siohan and M. Bacchiani: Fast vocabulary-independent audio search using path-based graph indexing, in *Proc. Eurospeech 2005*, Lisboa, Portugal, 2005.