Fast Vocabulary-Independent Audio Search Using Path-Based Graph Indexing

Olivier Siohan *Michiel Bacchiani*

IBM T.J. Watson Research Center
1101 Kitchawan Rd., Rte 134/PO BOX 218
Yorktown Heights, NY 10598, USA
{siohan,bacchian}@us.ibm.com

Abstract

Classical audio retrieval techniques consist in transcribing audio documents using a large vocabulary speech recognition system and indexing the resulting transcripts. However, queries that are not part of the recognizer's vocabulary or have a large probability of getting misrecognized can significantly impair the performance of the retrieval system. Instead, we propose a fast vocabulary independent audio search approach that operates on phonetic lattices and is suitable for any query. However, indexing phonetic lattices so that any arbitrary phone sequence query can be processed efficiently is a challenge, as the choice of the indexing unit is unclear. We propose an inverted index structure on lattices that uses paths as indexing features. The approach is inspired by a general graph indexing method that defines an automatic procedure to select a small number of paths as indexing features, keeping the index size small while allowing fast retrieval of the lattices matching a given query. The effectiveness of the proposed approach is illustrated on broadcast news and Switchboard databases.

1. Introduction

The rapidly increasing amount of spoken documents calls for solutions to archive, index and search these documents based on their content. The classical audio search paradigm consists in transcribing audio using a large vocabulary automatic speech recognition system (ASR) and applying standard indexing and information retrieval (IR) techniques on the transcripts. Indeed, such an approach has been shown to be very effective on broadcast news (BN) documents, as illustrated by the TREC (Text Retrieval Conference) Spoken Document Retrieval evaluations [1]. Key contributors to that success were the low words error rates (WER) on BN transcripts, the redundant nature of broadcast news documents, and the use of long queries.

However, a significant drawback of such an approach is that out-of-vocabulary (OOV) queries will not return any document. It has been experimentally observed that over 10% of user queries can be out-of-vocabulary[2], as queries often relate to named entities that typically have a poor coverage in the ASR vocabulary. Moreover, in many applications (e.g. BN) the OOV rate may get worse over time unless the recognizer's vocabulary is periodically updated. Hence, special attention should be paid to searching OOV queries. In addition, in applications dealing with conversational telephony quality audio, the high word error rates can significantly impair the retrieval performance, especially on short, single-word queries. This is compounded by the need to transcribe the audio very quickly (faster than real-time), which further degrades word error rates. Under these conditions, relying only on a single (1-best) transcript can significantly limit the overall performance of the retrieval system.

Popular approaches to deal with OOV queries are based on sub-word transcripts, where the sub-words are typically phones, syllables, or word-fragments (sequences of phones)[3, 4, 5]. The word queries are converted accordingly into sub-word sequences and searched for in the sub-word transcripts. To account for the high recognition error rates, lattices are often used to represent alternative recognition hypotheses. Phone lattices are attractive as they accommodate high error rate conditions as well as allow for OOV queries to be used [6, 7, 8, 4, 5, 9, 10]. Indeed, the use of phonetic lattices significantly improves recall and does provide the vocabulary independence required to handle arbitrary queries.

When using a phonetic lattice based approach, the audio search involves locating a given phone query string in a collection of phonetic lattices. While searching for a phone string in a phonetic lattice can be implemented efficiently [5, 10], linearly scanning the entire lattice collection does not scale to large databases. For efficiency reasons, inverted indices are commonly used in IR system, and indeed some form of inverted index is required to speed-up the audio search. Efficient indexing of a collection of lattices is however non-trivial, as the choice of the indexing unit is unclear. In [4], individual lattice arcs are used as indexing units to invert an entire collection of lattices. Single-label queries can be very efficiently processed, as the matching lattices are immediately retrieved from the inverted index. However, some form of linear search is still required when searching for multi-label queries (such as phonetic strings), as sequences of labels are not directly available in the index. In [8], a general indexation of lattices is proposed, which indexes expected term frequencies (EFT) of fixed-length sub-strings but approximates the EFT of a query by the minimum EFT of its sub-strings. Documents are retrieved and ranked based on the EFT of the query. The approach does not scale to large audio collections due to the complexity of the index construction, and cannot directly index long sub-strings. Some of the shortcomings of [8] are addressed in [9], where a N-gram phone-level language model is associated to each lattice and used to efficiently compute the EFT of any query.

In contrast to other approaches, this paper focuses on an efficient indexing of phonetic lattices both in terms of index size and retrieval time, for arbitrary phone sequence queries. Given a phone sequence query and a database of phonetic lattices, we first retrieve a list of candidate lattices that may contain the query, using a fast but approximate approach. This is followed by a verification step to ensure that the query is effectively present in the candidate lattices. The main contribution of this paper is on the first step of the search procedure, which uses an inverted index structure so that a list of candidate matching lattices can be retrieved very efficiently using look-up tables. As opposed to [9] which uses a complete set of short index keys and only the most probable long keys, our approach involves a key selection procedure that only selects a small subset of variable-length phone sequences as keys. The index construction procedure is based on a frequent structure-based approach for graph indexing [11], known for providing efficient query processing in large graph databases. The paper is organized as follows. In Section 2, we introduce the proposed graph indexing approach. Section 3 reports experimental results on both Broadcast News and Switchboard databases, and Section 4 concludes the paper.

2. Principle

In our application of interest, the audio recordings are first chopped into short segments (up to 30 sec long), called "documents". Speech recognition is then used to generate a phonetic lattice for each document. The search operates on single word queries represented as phone sequences[1] and involves retrieving the list of lattices matching the query. A phone sequence query is said to match a lattice if a path containing the same phone sequence can be found in the lat-

[1]The generation of phone-level queries from word queries is beyond the scope of this paper.

tice (after skipping non-speech symbols such as silence and noise, if any). The performance of the retrieval is measured in terms of precision/recall and the operating point is adjusted by pruning the lattices: large lattices can improve recall, but at the expense of getting a poor precision. We detail the procedure used to generate phonetic lattices in Section 2.1.

Though the search for a given phone string in a given lattice can be implemented efficiently, for example using one of the approaches presented in [5, 10], it is very inefficient to sequentially scan all the lattices to verify whether they contain the query, as the collection of lattices can be very large. Indeed, in order to scale up to processing hundreds or thousands of hours of audio, it is necessary to use an inverted index structure to quickly locate a small set of candidate lattices, using table look-up operations. The time-consuming detailed search for the phone sequence is then only carried out on the candidate lattices. The lattice indexing procedure in described in Section 2.2.

2.1. Phonetic lattice generation

As direct phonetic decoding may not have a very high accuracy, we adopted a two-step procedure to generate phonetic lattices. First, we generated word-fragment lattices where word-fragment are defined as variable-length sequences of phones. Then, we converted the word-fragment lattices into phonetic lattices. A similar 2-step approaches was also adopted in [5], though a different procedure was used to generate the word-fragment dictionary.

Our word-fragment dictionary is created by building a phone-based language model and pruning it to keep non-redundant higher order N-grams [12]. The list of the resulting phone N-grams is then used as word-fragments. For example, assuming that a 5-gram phone LM is built, the entire list of phone 1-gram, 2-gram,...5-gram is used as word-fragment dictionary. The LM pruning threshold is used to control the total number of phone N-grams, hence the size of the word-fragment lexicon.

Given a word-fragment lexicon, it is then possible to train a word-fragment language model and use it for recognition. As our decoding strategy is based on finite state machine, a word-fragment static decoding graph is eventually generated and used to decode and generate word-fragment lattices. The final conversion of the word-fragment lattices to phonetic lattices is then straightforward.

2.2. Path-based phonetic lattice indexing

Once the audio documents have been converted to phonetic lattices, our objective is to index these lattices so that all documents containing a given phonetic query can be efficiently retrieved. To solve that problem, we adopt a graph indexing approach originally developed to process XML queries [11]. Indeed, our lattice indexing problem is a simplified case of the classical graph query problem, which, given a graph database $D = \{g_1, ...g_n\}$ and a graph query q, consists in finding all graphs in which q is a subgraph. In our specific application, the graph g_i are the phonetic lattices, and the query q is simply a phone sequence.

The basic indexing unit selected to index the graph database[2] is a path, defined as a vertex sequence. The general idea of the graph indexing approach of [11] is to enumerate all paths in the database, up to a given length *maxL*, and index them. For a given path, the index then contains all graphs that include that path.

Given the size of the graphs (in terms of edges) and the large number of graphs in the database, it is difficult to index all paths. [11] suggests to use frequent subgraph structures as indexing units, generated using a graph mining algorithm. Since in our application, queries are sequences (as opposed to graphs), we will still adopt frequent paths (up to a maximum length *maxL*) as *indexing features*. To avoid the combinatorial growth of the number of *frequent paths*, only a small subset of frequent paths will be kept as indexing features. In addition, a *discriminative ratio* concept is introduced [11] that attempts to reduce the redundancy among frequent paths selected as features. The concepts of frequent paths and discriminative ratio are described in Sections 2.2.1 and 2.2.2, respectively.

Let $V(g)$ be the vertex set of a graph g, and $E(g)$ be the edge set. In our application, a phone label is assigned to each edge. The size of a graph is defined as the number of edges, denoted $|E(g)|$. A path q is a sub-path of a graph g, denoted by $q \subseteq g$, if the label sequence along q also exists in g. The graph query processing is a 2 step process described as follows [11].

Index construction The feature set F is selected by enumerating all paths up to maximum length *maxL* satisfying both the frequency and discriminative ratio criteria. For any feature $f \in F$, the set of graphs containing f is denoted $D_f = \{g_i | f \subseteq g_i, g_i \in D\}$ where D is the entire graph database.

Query processing Since the index only contains a subset of all paths, the query is processed in 2 steps:

- **Search** Given a query q, enumerate all features f of the query. The candidate set is defined as the intersection of all sets of graphs containing the query features, $C_q = \bigcap_f D_f$ ($f \subseteq q$ and $f \in F$)
- **Verification** For each graph g in C_q, an exact search is carried out to verify whether g contains the query q. That step is required since C_q is only guaranteed to contain all the features in q, but not necessarily the sequence q.

The index construction is an offline procedure that does not affect the query response time. The query response time is defined as the time spent in the search step, which involves table look-up operations to retrieve each D_f in addition to computing the set intersection, and the verification time, that is equal to $|C_q|$ times the average time required for a graph verification operation. Most of the processing is therefore spent in the graph verification operation (unless the index cannot fit in memory in which case the search time may significantly increase), so reducing the response time involves reducing the candidate set $|C_q|$ as much as possible. A good index should therefore attempt to use a small number of indexing features to keep the index small and manageable, but still large enough so that it minimizes $|C_q|$. Indices constructed under the frequent path and discriminative ratio constraints will have that property.

2.2.1. Frequent paths

Let $|D_q|$ be the number of graphs in the graph database D containing a path query q. $|D_q|$ is called the *support* of q. By definition, a path (also called fragment) is frequent if its support is greater than a minimum support, *minSup*. Recall that a path has to be frequent to be selected as indexing feature.

If q is frequent, the graphs containing q can directly be retrieved from the index, since by definition frequent paths are indexed. If q is not frequent, according to the search procedure outlined in Section 2.2 we should first enumerate all the features f_i of q. Let f_i be the frequent feature of q with the smallest support, $support(f_i)$. As the candidate set C_q is defined as $\bigcap D_{f_i}$, its size is bounded by $support(f_i)$, that is likely to be close to *minSup*. As a result, the cost of verifying all graphs in C_q can be kept small by using a low minimum support.

This illustrates that is is possible to build high quality indices using only frequent paths [11]. However, while a small $|C_q|$ can be obtained using a small minimum support *minSup*, it will also lead to a combinatorial explosion of the number of frequent paths. As the number of distinct paths is directly related to the path length l, the minimum support should be a function of l, denoted $\phi(l)$. For completeness of the index, short fragment should therefore have a low minimum support, while for compactness reasons, long fragment should have a high support.

The frequency constraint will therefore be expressed as follows: A fragment g is frequent under the size increasing support constraint if and only if $support(g) \geq \phi(len(g))$, where $\phi(l)$ is a monotonically non-decreasing function, and $len(g)$ refers to the length of the fragment g.

2.2.2. Discriminative ratio

While the size-increasing support constraint can significantly limit the size of the index, large graphs may still contain a large number of

[2] We will refer to the collection of phonetic lattice as the graph database henceforth.

frequent fragments. The discriminative ratio constraint will further reduce the size of the feature set by retaining only the most "useful" fragments as indexing features.

Suppose that two similar frequent fragments f_1 and f_2 are contained in the same set of graphs, $D_{f_1} = D_{f_2}$, and let's assume that f_2 is a subsequence of f_1. The longest fragment, f_1, should not be placed in the index, since the more general query f_2 will return the same set of graph. f_1 is said to be a redundant fragment. By definition, fragment x is redundant with respect to feature set F if $D_x \approx \bigcap_{f \in F \wedge f \subseteq x} D_f$. The condition $D_x \approx \bigcap_{f \in F \wedge f \subseteq x} D_f$ simply indicates that the set of graphs containing x can be well predicted from the set of graphs containing all sub-fragments of x. In that regard, there is no point using x as indexing feature since its sub-fragments provide the required information to retrieve D_x.

On the opposite, if the set D_x cannot be predicted from the sub-fragments of x, x is said to be a discriminative fragment. Formally, x is discriminative with respect to feature set F if $D_x \ll \bigcap_{f \in F \wedge f \subseteq x} D_f$. A discriminative fragment should be included in the indexing feature set since not having it would imply retrieving a large candidate set C_x that will then have to be verified to reject false positive hits.

The discriminative property of a fragment x can be measured by the discriminative ratio γ, defined as [11]:

$$\gamma(x) = \frac{|\bigcap_{f \in F \wedge f \subseteq x} D_f|}{|D_x|}. \quad (1)$$

The discriminative ratio constraint will therefore be expressed as follows: A fragment g is discriminant under the discriminative ratio constraint if and only if $\gamma(g) \geq \gamma_{min}$, where γ_{min} is a minimum discriminative ratio threshold.

2.2.3. Index construction

Given a graph database D, a size-increasing support function $\phi(l)$, a minimum discriminative ratio γ_{min} and a maximum fragment length *maxL*, the feature set (index keys) is constructed according to Algorithm 1. As the computation of the discriminative ratio involves computing D_{f_i} for all features f_i, a side product of the feature selection algorithm is the creation of the index. It is worth noting that the feature set can be extracted on a subset of the graph database, and can then be used to construct the inverted index. It is then possible to incrementally update the index as more and more documents are available, without having to regenerate the feature set.

Algorithm 1 Feature selection (after [11])

$F \leftarrow \{f_\emptyset\}, D_{f_\emptyset} \leftarrow D, l \leftarrow 1$
while $l \leq maxL$ **do**
  for all fragment x, whose size is l **do**
    if $support(x) \geq \phi(len(x)$ and $\gamma(x) \geq \gamma_{min}$ **then**
      $F \leftarrow F \cup \{x\}$
    end if
  end for
  $l \leftarrow l + 1$
end while
return F

2.2.4. Search

The search procedure used to retrieve the candidate set C_q for a given query q is described in Algorithm 2. The main idea is to enumerate all sub-fragments of the query q that are available in the index, and intersect the corresponding sets of graphs.

The retrieval of the candidate set C_q can be implemented very efficiently as it only involves table look-up operations and the intersection of the sets of graphs retrieved from the index. A verification step should follow to identify the true matching graphs available in the candidate set, and reject false positive hits. In our experiments, the verification step was implemented as a full search through each lattice, though better implementations are possible [5, 10].

Algorithm 2 Search (after [11])

$C_q \leftarrow D$
for all fragment $x \subseteq q$ and $len(x) \leq maxL$ **do**
  if $x \in F$ **then**
    $C_q \leftarrow C_q \cap D_x$
  end if
end for
return C_q

Frag. Length	# Fragments	# Features
1	44	3
2	1553	1458
3	27820	25523
4	187348	28568
5	616190	13101
6	1389502	3108
7	2615542	550
8	4426629	70
9	7026082	17
10	10747449	4
Total	27038159	72402

Table 1: Total number of distinct fragments and number of indexing features as a function of the fragment length.

3. Experiments and Results

3.1. Broadcast News

In a first series of experiments, we used the English Hub-4 broadcast news Eval97 and Eval98 data sets, representing about 6 hours of audio. The recordings were segmented into a total of 1484 short audio documents. The list of queries was defined as all words available in the reference transcript after removing stop words, leading to a total of 6116 single word queries. A document is said to be relevant to a given query if that query is contained in the document. On average, each query is associated to about 4 relevant documents. The fragment vocabulary consisted of about 25K word fragments, ranging from 1 to 5 phone long. A 3-gram fragment-based LM was trained and used to construct a static decoding network. Cross-word, speaker independent triphone models were used, for a total of 8k tied-states and 128K Gaussians. For each audio file, a word-fragment lattice was generated and then converted to a phonetic lattice. The phonetic lattices were then pruned using various beam pruning threshold to generate the lattices to index. The pruning strategy was based on likelihood, i.e. paths with low likelihood were pruned away.

For each lattice beam pruning, we carried out the following experiments: (1) Index construction, (2) Search, (3) Verification. In step (1), an index is constructed using Algorithm 1, given the database of pruned lattices. In step (2), for each query a set of lattice candidates is extracted from the index using Algorithm 2. In step (3), a full search is conducted on each lattice in the candidate set to verify whether the lattice contains the phone sequence query. Note that in this work, we do not associate any score to the retrieved documents, and the result of the search is an unordered list of documents.

Table 1 represents the total number of distinct fragments in the entire lattice database, and the final number of indexing features that were extracted from the database. The size increasing support function was set to 1 for all fragments up to 3-phone long, and then was set to increase exponentially with the fragment length. The effect of both the minimum support and the discriminative ratio constraints are seen in that Table. For example, only 25523 of the 3-phone fragments were selected as indexing feature (out of the 27820 available fragments), due to the discriminative ratio constraints.

In Table 2, we present for each beam pruning threshold (large beam means large lattice) the average precision and recall after the verification step, the average number of documents per query in the candidate set, and the final number of retrieved documents per query

Beam	Prec.	Rec.	$\lvert C_q \rvert$	# Doc
0	53.2	60.3	9	7
0.4	51.5	65.4	10	8
0.6	50.2	68.2	11	9
0.8	48.0	70.7	13	11
1.2	42.8	74.8	15	12
1.5	38.5	77.5	19	16

Table 2: BN - Pruning Beam (0 means 1-best decoding), Average Precision per query (Prec.), Average Recall per query (Rec.), Average # of documents in the candidate set per query ($|C_q|$), Average number of retrieved documents per query (# Doc)

Beam	Prec.	Rec.	$\lvert C_q \rvert$	# Doc
0.4	40.7	53.2	9	8
0.6	39.0	56.0	11	9
0.8	36.9	58.7	13	11
1.2	33.1	63.8	19	16
1.5	29.6	67.9	26	22

Table 3: RT'02 - Pruning Beam, Average Precision per query (Prec.), Average Recall per query (Rec.), Average # of documents in the candidate set per query ($|C_q|$), Average number of retrieved documents per query (# Doc)

after verification. The beam value 0 represents the baseline performance that corresponds to using a 1-best decoded string to generate the index. As the lattice beam gets larger, the recall increase as expected, for a moderate degradation of the precision (up to beam=0.6). As we do not rank the documents after verification (though it could easily done by deriving the expected count of the query during the verification step), the lattice beam allows to control the precision/recall trade-off. It is interesting to note that the size of the candidate set, $|C_q|$, that is the set of candidates lattices extracted from the index using Algorithm 2 is very small. Therefore, the expensive full search on each candidate lattice should only be carried out on a few lattices, and each query can be processed in a fraction of seconds on average (on 6 hours of audio). In contrast, the phonetic-lattice search of [10] is reported to take several seconds to search through 1 hour worth of audio.

3.2. Switchboard

Similar experiments have been carried out on the Switchboard RT'02 evaluation set [13]. The test set was segmented into 6080 short audio documents. The list of queries was made of all non-stop words in the reference transcripts, leading to a total of 3080 single word queries. On average, each query is associated to about 4 relevant documents. The acoustic models were built on about 400 hours of Switchboard data. Word-fragment lattices were generated using a 25K word-fragment lexicon, and were converted to phonetic lattices. As for the BN experiments, the lattices were pruned using different beam pruning thresholds, before indexing. For each beam pruning threshold, we present in Table 3 the average precision and recall after the verification step, the average number of documents per query in the candidate set, and the final number of retrieved documents per query after verification. As in the BN experiments, the index is able to retrieve a very small number of candidate documents, by using only table look-up operations, even on the largest lattices. As a result, on average, each query is process in a fraction of seconds. To improve the precision, we combined our vocabulary independent indexing with a standard word-based index, generated by 1-best decoding using a 35K-word lexicon. The word error rate is 36.5%. The combination is done using the cascaded approach presented in [4]. The word index is first used to retrieve a list of documents. If that list is empty, the vocabulary-independent index is then used. After the combination, the precision jumps to about 63% for all lattice beam-pruning thresholds, while the recall varies from 61.8% to 66.2%. This illustrates that vocabulary-independent audio search can be efficiently implemented using inverted index structures on both 1-best word transcripts and phonetic lattices.

4. Conclusion

We presented in this paper a vocabulary independent method for audio search. The main contribution of the paper is a scalable algorithm for efficient audio retrieval based on arbitrary phone sequence queries operating on phonetic lattices. The proposed approach is based on an inverted index structure for phonetic lattices that uses paths as indexing features. By design, the features are selected based on their "usefulness" for retrieval purposes using the notion of minimum support and discriminative ratio. We believe that such an indexing strategy can efficiently scale to much larger audio databases. In addition, some of the related work on phonetic search [9, 10] could benefit from our approach which provides a computationally efficient way to extract a small number of candidate lattices, on which more advanced search strategies could be carried out (for example to locate the position of the query within the audio document, or to associate a score to each document for ranking purposes). In future work, we will study the use of alternative lattice pruning strategies (e.g. posterior-based pruning) and extend our experiments to larger audio databases.

5. References

[1] J. Garofolo, G. Auzanne, and E. Voorhees, "The TREC spoken document retrieval task: A success story," in *Proceedings of TREC-9 conference*, 2000, http://www.nist.gov/speech/tests/sdr/sdr2000/papers/-01plenary1.pdf.

[2] B. Logan, P. Moreno, J. M. Van Thong, and E. Whittaker, "An experimental study of an audio indexing system for the Web," in *Proc. ICSLP*, Beijing, China, 2000.

[3] K. Ng, *Subword-based Approaches for Spoken Document Retrieval*, Ph.D. thesis, MIT, Feb. 2000.

[4] M. Saraclar and R. Sproat, "Lattice-based search for spoken utterance retrieval," in *Proc. of the annual meeting of the HLT conf. and North American Chapter of the Association for Computational Linguistics*, Boston, USA, May 2004.

[5] F. Seide, P. Yu, C. Ma, and E. Chang, "Vocabulary-independent search in spontaneous speech," in *Proc. IEEE ICASSP*, 2004.

[6] D. A. James and S. J. Young, "A fast lattice-based approach to vocbulary independent wordspotting," in *Proc. IEEE ICASSP*, 1994, vol. 1, pp. 377–380.

[7] D. A. James, "A system for unrestricted topic retrieval from radio news broadcasts," in *Proc. IEEE ICASSP*, Atlanta, Georgia, USA, 1996, vol. 1, pp. 279–282.

[8] C. Allauzen, M. Mohri, and M. Saraclar, "General indexation of weighted automata – application to spoken utterance retrieval," in *Proc. of the annual meeting of the HLT conf. and North American Chapter of the Association for Computational Linguistics*, Boston, USA, May 2004, pp. 33–40.

[9] P. Yu and F. Seide, "Fast two-stage vocabulary-independent search in spontaneous speech," in *Proc. IEEE ICASSP*, 2005.

[10] K. Thambiratnam and S. Sridharan, "Dynamic match phone-lattice searches for very fast and accurate unrestricted vocabulary keyword spotting," in *Proc. IEEE ICASSP*, 2005.

[11] X. Yan, P. S. Yu, and J. Han, "Graph indexing: a frequent structure-based approach," in *SIGMOD '04: Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, New York, NY, USA, 2004, pp. 335–346, ACM Press.

[12] K. Seymore and R. Rosenfeld, "Scalable backoff language models," in *Proc. ICSLP*, vol. 1, pp. 232–235.

[13] NIST, "Rich transcription 2002 evaluation," http://www.nist.gov/speech/tests/rt/rt2002/.