Anatomy of an extremely fast LVCSR decoder

George Saon, Daniel Povey and Geoffrey Zweig

IBM T. J. Watson Research Center, Yorktown Heights, NY, 10598
e-mail: {gsaon,dpovey,gzweig}@us.ibm.com

Abstract

We report in detail the decoding strategy that we used for the past two Darpa Rich Transcription evaluations (RT'03 and RT'04) which is based on finite state automata (FSA). We discuss the format of the static decoding graphs, the particulars of our Viterbi implementation, the lattice generation and the likelihood evaluation. This paper is intended to familiarize the reader with some of the design issues encountered when building an FSA decoder. Experimental results are given on the EARS database (English conversational telephone speech) with emphasis on our faster than real-time system.

1. Introduction

Recent advances in decoding algorithms coupled with the availability of ever increasing computing power has made accurate, real-time LVCSR possible for various domains such as broadcast news transcription [7] or conversational telephone speech recognition [6]. One such advance is the use of weighted finite-state transducers which allow to efficiently encode all the various knowledge sources present in a speech recognition system (language model, pronunciation dictionary, context decision trees, etc). The network resulting from the composition of these WFSTs, after minimization, can be directly used in a Viterbi decoder [4]. Such decoders have been shown to yield excellent performance when compared to classic approaches [3].

This approach is currently so successful at IBM that no less than five different Viterbi decoders using FSA technology have been written, three of which were written by the authors of this paper. While these decoders share many common characteristics, we focus here only on the recognizer that has been used during the past two Darpa EARS evaluations.

2. Decoder description

2.1. Static decoding graphs

Our decoder operates on static graphs obtained by successively expanding the words in an n-gram language model in terms of their pronunciation variants, the phonetic sequences of these variants and the context dependent acoustic realizations of the phones. The main advantage of using static graphs is that the graphs can be heavily optimized at "compile" time (e.g. through determinization and minimization [4]) in advance, so that minimal decoding work is required at "decode" time.

The decoding graphs that we use have some distinctive characteristics when compared to standard WFSTs. The first characteristic is that they are *acceptors* instead of transducers. The arcs in the graph can have three different types of labels:

- *leaf* labels (context-dependent output distributions),
- *word* labels and
- *epsilon* labels (e.g. due to LM back-off states).

Although not assumed by the decoder, it is helpful if the word labels are always at the end of a word, i.e. right after the sequence of corresponding leaves. This ensures that the time information associated to a word sequence is always correct which is not the case for WFSTs since word labels can be shifted around. In the latter case, the 1-best word sequences or lattices have to be acoustically re-aligned to get the correct times and scores. In addition, the FSA representation is more compact since only one integer per arc is required to store the label. The drawback of having word labels at the end is that suffixes from *different* words cannot be shared anymore.

The second characteristic has to do with the types of states present in our graphs:

- *emitting* states for which all incoming arcs are labeled by the *same* leaf and
- *null* states which have incoming arcs labeled by words or epsilon.

This is in effect equivalent to having the observations emitted on the *states* of the graph not on the arcs. The advantage is that the Viterbi scores of the states can be directly updated with the observation likelihoods and the scores of the incoming arcs. It also makes the decoder conceptually simpler: there is no need to keep track of active arcs during the search, only of active states. It can happen however that, after determinization and minimization, arcs with different leaf labels point to the same emitting state. In this case, the state is split into several different states each having incoming arcs labeled by the

same leaf. Even when using large span phonetic context such as cross-word septaphones, this phenomenon is relatively rare and leads to only a small increase in graph size (<10%). Finally, each emitting state has a self-loop labeled by the leaf of the incoming arcs. Null states can have incoming arcs with arbitrary word or epsilon labels (but no leaf labels). An illustration of our graph format is given in Figure 1.

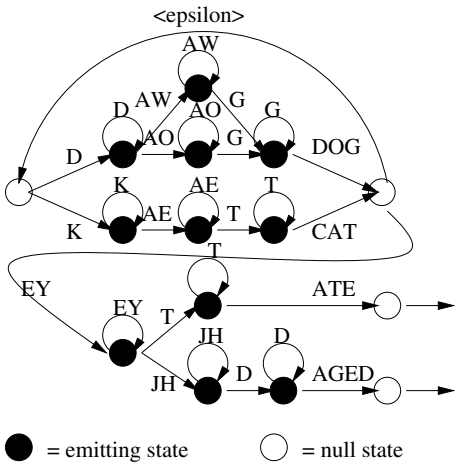

Figure 1: Example of an FSA decoding graph (with phone labels instead of leaf labels).

2.2. Viterbi search

At a high level, the Viterbi search is a simple token passing algorithm without any context information attached to the tokens. It can be basically written as a loop over time frames and an inner loop over sets of active states. A complication arises in the processing of null states, which do not account for any observations. Because of this, an arbitrary number of null states might need to be traversed for each speech frame that is processed. Furthermore, since multiple null-state paths might lead to the same state, the nulls must be processed in topological order.

In order to recover the Viterbi *word* sequence, it is not necessary to store backpointers for all the active states. Instead one can store only the backpointer to the previous word in the sequence. More precisely, every time we traverse an arc labeled by a word, we create a new *word trace* structure containing the identity of the word, the end time for that word (which is the current time frame) and a backpointer to the previous word trace. We then pass a pointer to this trace as a token during the search. This procedure is slightly modified for lattice generation as it will be explained later on. Storing only word traces rather than state traces during the forward pass reduces the dynamic memory requirements dramatically (several orders of magnitude for some tasks). The drawback of this technique however is that the Viterbi state sequence cannot be recovered anymore.

Even though we store minimal information during the forward pass, for very large utterances and/or wide decoding beams and/or lattice generation, the memory usage can be excessive. We implemented garbage collection of the word traces in the following way. We mark all the traces which are active at the current time frame as alive. Any predecessor of a live trace becomes alive itself. In a second pass, the array of traces is overwritten with only the live traces (with appropriate pointer changes). When done every 100 frames or so, the runtime overhead of this garbage collection technique is negligible.

2.3. Search speed-ups

Here we present some search optimization strategies which were found to be beneficial. They have to do with the way the search graph is stored and accessed and with the way pruning is performed.

- *Graph memory layout*. The decoding graph is stored as a linear array of arcs sorted by origin state, each arc being represented by a destination state, a label and a cost (12 bytes/arc). Each state has a pointer to the beginning of the sequence of outgoing arcs for that state, the end being marked by the pointer of the following state (4 bytes/state). These data structures are similar to the ones described in [1].

- *Successor look-up table*. The second optimization has to do with the use of a look-up table which maps static state indices (from the static graph) to dynamic state indices. The role of this table is to indicate whether a successor state has already been accessed and, if yes, what entry it has in the list of active states.

- *Running beam pruning*. For a given frame, only the hypotheses whose score are greater than the *current* maximum for that frame minus the beam are expanded. Since this is an overestimate of the number of hypotheses which survived, the paths are pruned again based on the absolute maximum for that frame (minus the beam) and based on a maximum number of active states (rank or histogram pruning). This resulted in a 10%-15% speed-up over standard beam pruning.

2.4. Lattice generation

The role of a lattice (or word-graph) is to efficiently encode all the possible word sequences which have appreciable likelihood given the acoustic evidence. Standard lattice generation in (dynamic search graph) Viterbi decoding uses a word-dependent N-best algorithm where multiple backpointers to previous words are kept at word ends [5, 8]. When using static graphs however, there is a complication due to the merges of state sequences that can happen in the middle of words.

The strategy we adopt is to keep track of the N-best *distinct* word sequences arriving at every state. This is achieved through hashing of the word sequences from the beginning of the utterance up to that state. More precisely, during the forward pass, we propagate N tokens from a state to its successors. Token i contains the forward score of the ith-best path, the hash code of the word sequence up to that point and a backpointer to the previous *word trace*. Once we traverse an arc labeled by a word, we create a new word trace which contains the word identity, the end time and the N tokens up to that point. We then propagate only the *top-scoring* path (token). At merge points, we perform a mergesort uniq operation to get from 2N down to N tokens (the tokens are kept sorted in descending score order). This lattice generation procedure is illustrated in Figure 2.

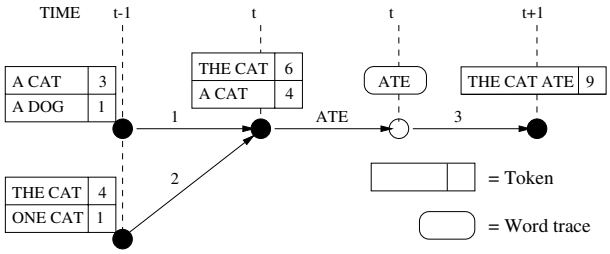

Figure 2: N-best lattice generation (N=2). Here arcs carry word labels and scores (higher scores are better). Word sequences are represented by hash codes.

In Table 1, we report the link density (number of arcs in the lattice divided by the number of words in the reference) as a function of N for the same pruning parameter settings. We normally use N=5 to achieve a good balance between lattice size and lattice quality.

N-best degree	2	5	10
Lattice link density	29.4	451.0	1709.7

Table 1: Lattice link density as a function of N.

Table 2 shows the word error rates for three different test sets of the EARS database obtained after language model rescoring and consensus processing of the lattices at the speaker adapted level. The language model used to generate the lattices has 4.1M n-grams while the rescoring LM is significantly larger with 100M n-grams (please refer to [9] for details on how these language models were trained).

2.5. Likelihood computation

In [6], we have presented a likelihood computation strategy based on a hierarchical Gaussian evaluation which is decoupled from the search. Here, we contrast this technique with "on-demand" likelihood computation in the sense that we evaluate the Gaussians only for the states which are accessed during the search as suggested in [7]. A further refinement is achieved by combining the two approaches. This works as follows: first, we perform a top-down clustering of all the mixture components in the system using a Gaussian likelihood metric until we reach 2048 clusters (Gaussians). At runtime, we evaluate the 2048 components for every frame and, for a given state accessed during the search, we only evaluate those Gaussians which map to one of the top N clusters for that particular frame. Figure 3 shows the word error rate versus run-time factor (including search) for the three different likelihood schemes: "hierarchical decoupled" (pre-computation and storage of all the likelihoods), "on-demand" and "hierarchical on-demand" (computing on-demand only those Gaussians which are in the most likely clusters). For both on-demand techniques, we use a likelihood batch strategy which computes and stores the likelihoods for eight consecutive frames into the future, as described in [7].

	RT03	DEV04	RT04
Speaker-adapted decoding	17.4	14.5	16.4
LM rescoring + consensus	16.1	13.0	15.2

Table 2: Word error rates for LM rescoring and consensus processing on various EARS test sets.

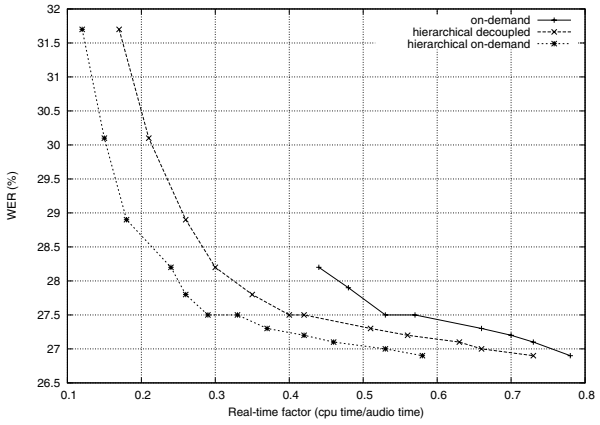

Figure 3: Word error rate versus real time factor for various likelihood schemes (EARS RT'04 speaker independent decoding). Times are measured on Linux Pentium IV 2.8GHz machines and are inclusive of the search.

3. Experimental setup

We study the behaviour of the LVCSR decoder on our EARS 2004 evaluation submission in the one times real-time (or 1xRT) category. The architecture we propose uses an extremely fast initial speaker-independent decoding to estimate VTL warp factors, feature-space and model-

space MLLR transformations that are used in a final speaker-adapted decoding [6]. The decoding graphs for the two decoding passes are built using identical vocabularies, similarly sized 4-gram language models, but very different context decision trees. For the compilation of the phonetic decision trees into FSTs, we applied an efficient incremental technique described recently in [2]. Table 3 shows various decoding graph statistics. The maximum amount of memory used during the determinization step was 4GB.

	SI	SA
Phonetic context	± 2	± 3
Number of leaves	7.9K	21.5K
Number of words	32.9K	32.9K
Number of n-grams	3.9M	4.2M
Number of states	18.5M	26.7M
Number of arcs	44.5M	68.7M

Table 3: Graph statistics for the speaker-independent and speaker-adapted decoding passes. The number of arcs includes self-loops.

The drastic runtime constraints for the 1xRT submission forced us to choose quite different operating points for the speaker-independent and speaker adapted decoding. Thus, the SI decoding was allotted a runtime fraction of only 0.14xRT, whereas the SA decoding ran at a more "leisurely pace" of 0.55xRT. This had an influence on the number of search errors as can be seen from Table 4. In the same table, we indicate the error rates and various decoding statistics for the two passes. The test set consists of 36 two-channel telephone conversations (72 speakers) totaling 3 hours of speech and 37.8K words. Times were measured on a Linux Pentium IV 3.4 GHz machine (without hyperthreading).

	SI	SA
Word error rate	28.7%	19.0%
Search errors	2.2%	0.3%
Likelihood/search ratio	60/40	55/45
Avg. number of Gaussians/frame	7.5K	43.5K
Max. number of states/frame	5.0K	15.0K

Table 4: Error rates and decoding statistics on RT'04 for the 1xRT system.

Lastly, we discuss the memory requirements for the speaker adapted decoding, which is by far the most resource consuming. The memory usage can be summarized as follows: 1.2Gb of static memory divided into 932Mb for the decoding graph and 275Mb for 850K 40-dimensional Gaussians and 133Mb of dynamic memory (220Mb with lattice generation).

4. Conclusion

In this paper we explored some of the design issues encountered in FSA-based decoding. Specifically, we discussed: (a) the choice of acceptors instead of transducers as static decoding graphs with observations emitted on states instead of arcs (b) the use of word traces for traceback information (c) a lattice generation procedure based on N-best distinct word sequences and (d) an on-demand hierarchical likelihood computation. Using these techniques, we showed that it is possible to perform very accurate LVCSR decoding under tight time constraints.

5. Acknowledgment

The authors wish to thank Stanley Chen for the work on decoding graph construction and Miroslav Novak for suggesting the graph memory layout and the use of word traces.

6. References

[1] D. Caseiro and I. Trancoso. Using dynamic WFST composition for recognizing broadcast news. In ICSLP'02, Denver, CO, 2002.

[2] S. Chen. Compiling large-context phonetic decision trees into finite-state transducers. In Eurospeech'03, Geneva, Switzerland, 2003.

[3] S. Kanthak, H. Ney, M. Riley and M. Mohri. A comparison of two LVR search optimizations techniques. In ICSLP'02, Denver, CO, 2002.

[4] M. Mohri, F. Perreira and M. Riley. Weighted finite state transducers in speech recognition. In ISCA ITRW ASR'00, Paris, France, 2000.

[5] J. Odell. The use of context in large vocabulary speech recognition. PhD thesis. University of Cambridge, United Kingdom, 1995.

[6] G. Saon, G. Zweig, B. Kingsbury, L. Mangu and U. Chaudhari. An architecture for rapid decoding of large vocabulary conversational speech. In Eurospeech'03, Geneva, Switzerland, 2003.

[7] M. Saraclar, M. Riley, E. Bocchieri and V. Goffin. Towards automatic closed captioning: low latency real-time broadcast news transcription. In ICSLP'02, Denver, CO, 2002.

[8] R. Schwartz and S. Austin. Efficient, high-performance algorithms for n-best search. In Darpa Workshop on Speech and Natural Language, Hidden Valley, PA, 1990.

[9] H. Soltau, B. Kingsbury, L. Mangu, D. Povey, G. Saon, and G. Zweig. The IBM 2004 conversational telephony system for rich transcription. In ICASSP'05, Philadelphia, PA, 2005.