GENERATING EXACT LATTICES IN THE WFST FRAMEWORK

Daniel Povey[1], *Mirko Hannemann*[1,2],
Gilles Boulianne[3], *Lukáš Burget*[4], *Arnab Ghoshal*[5], *Miloš Janda*[2], *Martin Karafíat*[2], *Stefan Kombrink*[2],
Petr Motlíček[6], *Yanmin Qian*[7], *Ngoc Thang Vu*[8], *Korbinian Riedhammer*[9], *Karel Veselý*[2]

[1] Microsoft Research, Redmond, WA, dpovey@microsoft.com
[2] Brno University of Technology, Czech Republic, ihannema@fit.vutbr.cz
[3] CRIM, Montreal, Canada [4] SRI International, Menlo Park, CA, USA
[5] University of Edinburgh, U.K. [6] IDIAP, Martigny, Switzerland
[7] Tsinghua University, Beijing, China [8] Karlsruhe Institute of Technology, Germany
[9] Pattern Recognition Lab, University of Erlangen-Nuremberg, Germany

ABSTRACT

We describe a lattice generation method that is exact, i.e. it satisfies all the natural properties we would want from a lattice of alternative transcriptions of an utterance. This method does not introduce substantial overhead above one-best decoding. Our method is most directly applicable when using WFST decoders where the WFST is expanded down to the HMM-state level. It outputs lattices that include state-level alignments as well as word labels. The general idea is to create a state-level lattice during decoding, and to do a special form of determinization that retains only the best-scoring path for each word sequence.

Index Terms— Speech Recognition, Lattice Generation

1. INTRODUCTION

In Section 2 we give a Weighted Finite State Transducer (WFST) interpretation of the speech-recognition decoding problem, in order to introduce notation for the rest of the paper. In Section 3 we define the lattice generation problem, and in Section 4 we review previous work. In Section 5 we give an overview of our method, and in Section 6 we summarize some aspects of a determinization algorithm that we use in our method. In Section 7 we give experimental results, and in Section 8 we conclude.

2. WFSTS AND THE DECODING PROBLEM

The graph creation process we use in our toolkit, Kaldi [1], is very close to the standard recipe described in [2], where the Weighted Finite State Transducer (WFST) decoding graph is

$$HCLG = \min(\det(H \circ C \circ L \circ G)), \tag{1}$$

where $\circ$ is WFST composition (note: view $HCLG$ as a single symbol). For concreteness we will speak of "costs" rather than weights, where a cost is a floating point number that typically represents a negated log-probability. A WFST has a set of states with one distinguished start state[1], each state has a final-cost (or ∞ for non-final states); and there is a set of arcs, where each arc has a weight (just think of this as a cost for now), an input label and an output label. In $HCLG$, the input labels are the identifiers of context-dependent HMM states, and the output labels represent words. For both the input and output symbols, the special label ϵ may appear meaning "no label is present."

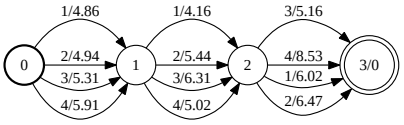

Fig. 1. Acceptor U describing the acoustic scores of an utterance

Imagine we have an utterance, of length T, and want to "decode" it (i.e. find the most likely word sequence, and the corresponding alignment). A WFST interpretation of the decoding problem is as follows. We construct an acceptor, or WFSA, as in Fig. 1 (an acceptor is represented as a WFST with identical input and output symbols). It has $T+1$ states, and an arc for each combination of (time, context-dependent HMM state). The costs on these arcs are the negated and scaled acoustic log-likelihoods. Call this acceptor U. Define

$$S \equiv U \circ HCLG, \tag{2}$$

which is the *search graph* for this utterance. It has approximately $T + 1$ times more states than $HCLG$ itself. The decoding problem is equivalent to finding the best path through S. The input symbol sequence for this best path represents the state-level alignment, and the output symbol sequence is the corresponding sentence. In practice we do not do a full search of S, but use beam pruning. Let B be the searched subset of S, containing a subset of the states and arcs of S obtained by some heuristic pruning procedure. When we do Viterbi decoding with beam-pruning, we are finding the best path through B.

Since the beam pruning is a part of any practical search procedure and cannot easily be avoided, we will define the desired outcome of lattice generation in terms of the visited subset B of the search graph S.

Thanks to Sanjeev Khudanpur for his help in preparing the paper, and to Honza Černocký, Renata Kohlová, and Tomáš Kašpárek for their help relating to the Kaldi'11 workshop at BUT.

[1]This is the formulation that corresponds best with the toolkit we use.

3. THE LATTICE GENERATION PROBLEM

There is no generally accepted single definition of a lattice. In [3] and [4], it is defined as a labeled, weighted, directed acyclic graph (i.e. a WFSA, with word labels). In [5], time information is also included. In the HTK lattice format [6], phone-level time alignments are also supported (along with separate language model, acoustic and pronunciation-probability scores), and in [7], state-level alignments are also produced. In our work here we will be producing state-level alignments; in fact, the input-symbols on our graph, which we call *transition-ids*, are slightly more fine-grained than acoustic states and contain sufficient information to reconstruct the phone sequence.

There is, as far as we know, no generally accepted problem statement for lattice generation, but all the the authors we cited seem to be concerned with the accuracy of the information in the lattice (e.g. that the scores and alignments are correct) and the completeness of such information (e.g. that no high-scoring word-sequences are missing). The simplest way to formalize these concerns is to express them in terms of a lattice pruning beam $\alpha > 0$ (interpret this as a log likelihood difference).

- The lattice should have a path for every word sequence within α of the best-scoring one.
- The scores and alignments in the lattice should be accurate.
- The lattice should not contain duplicate paths with the same word sequence.

Actually, this definition is not completely formalized– we have left the second condition (accuracy of scores and alignments) a little vague. Let the lattice be L. The way we would like to state this requirement is:

- For every path in L, the score and alignment corresponds to the best-scoring path in in B for the corresponding word sequence[2].

The way we actually have to state the requirement in order to get an efficient procedure is:

- For every word-sequence in B within α of the best one, the score and alignment for the corresponding path in L is accurate.
- All scores and alignments in L correspond to actual paths through B (but not always necessarily the best ones).

The issue is that we want to be able to prune B before generating a lattice from it, but doing so could cause paths not within α of the best one to be lost, so we have to weaken the condition. This is no great loss, since regardless of pruning, any word-sequence not within α of the best one could be omitted altogether, which is the same as being assigned a cost of ∞). By "word-sequence" we mean a sequence of whatever symbols are on the output of $HCLG$. In our experiments these symbols represent words, but not including silence, which we represent via alternative paths in L.

4. PREVIOUS LATTICE GENERATION METHODS

Lattice generation algorithms tend to be closely linked to particular types of decoder, but are often justified by the same kinds of ideas. A common assumption underlying lattice generation methods is the *word-pair assumption* of [5]. This is the notion that the time boundary between a pair of words is not affected by the identity of any earlier words. In a decoder in which there is a different copy of the lexical tree for each preceding word, assuming the word-pair assumption holds, in order to generate an accurate lattice it is sufficient to store a single Viterbi back-pointer at the word level; the entire set of such back-pointers contains enough information to generate the lattice. Authors who have used this type of lattice generation method [5, 8] have generally not been able to evaluate how correct the word-pair assumption is in practice, but it seems unlikely to cause problems. Such methods are not applicable for us anyway, as we use a WFST based decoder in which each copy of the lexical tree does not have a unique one-word history.

The lattice generation method described in [3] is applicable to decoders that use WFSTs [2] expanded down to the C level (i.e. CLG), so the input symbols represent context-dependent phones. In WFST based decoding networks, states normally do not have a unique one-word history, but the authors of [3] were able to satisfy a similar condition at the phone level. Their method was to store a single Viterbi back-pointer at the phone level; use this to create a phone-level latice; prune the resulting lattice; project it to leave only word labels; and then remove ϵ symbols and determinize. Note that the form of pruning referred to here is not the same as beam pruning as it takes account of both the forward and backward parts of the cost. The paper also reported experiments with a different method that did not require any phone-pair assumption; these experiments showed that the more efficient method that relied on the phone-pair assumption had almost the same lattice oracle error rate as their more efficient method. However, the experiments did not evaluate how much impact the assumption had on the accuracy of the scores, and this information could be important in some applications.

The lattice generation algorithm that was described in [7] is applicable to WFSTs expanded down to the H level (i.e. $HCLG$), so the input symbols represent context-dependent states. It keeps both scores and state-level alignment information. In some sense this algorithm also relies on the word-pair assumption, but since the copies of the lexical tree in the decoding graph do not have unique word histories, the resulting algorithm has to be quite different. Viterbi back-pointers at the word level are used, but the algorithm keeps track of not just a single back-pointer in each state, but the N best back-pointers for the N top-scoring distinct preceding. Therefore, this algorithm has more in common with the sentence N-best algorithm than with the Viterbi algorithm. By limiting N to be quite small (e.g. N=5) the algorithm was made efficient, but at the cost of losing word sequences that would be within the lattice-generation beam.

5. OVERVIEW OF OUR ALGORITHM

5.1. Version without alignments

In order to explain our algorithm in the easiest way, we will first explain how it would be if we did not keep the alignment information, and were storing only a single cost (i.e. the total acoustic plus language-model cost). This is just for didactic purposes; we have not implemented this simple version. In this case our algorithm would be quite similar to [3], except at the state level rather than the phone level. We actually store forward rather than backward pointers: for each active state on each frame, we create a forward link record for each active arc out of that state; this points to the record for the destination state of the arc on the next frame (or on the current frame, for ϵ-input arcs). As in [3], at the end of the utterance we prune the resulting graph to discard any paths that are not within the beam α of the best cost. Let the pruned graph be P, i.e.

$$P = \mathrm{prune}(B, \alpha), \tag{3}$$

[2] Or one of the best-scoring paths, in case of a tie.

where B is the un-pruned state-level lattice. We project on the output labels (i.e. we keep only the word labels), then remove ϵ arcs and determinize. In fact, we use a determinization algorithm that does ϵ removal itself.

As in [3], to save memory we actually do the pruning periodically rather than waiting for the end of the file (we do it every 25 frames). Our method is equivalent to their method of linking all currently active states to a "dummy" final state and then pruning in the normal way. However, we implement it in such a way that the pruning algorithm does not always have to go back to the beginning of the utterance. For each still-active state, we store the cost difference between the best path including that state, and the best overall path. This quantity does not always change between different iterations of calling the pruning algorithm, and when we detect that these quantities are unchanged for a particular frame, the pruning algorithm can stop going backward in time.

After the determinization phase, we prune again using the beam α. This is needed because the determinization process can introduce a lot of unlikely arcs. In fact, for particular utterances the determinization process can cause the lattice to expand enough to exhaust memory. To deal with this, we currently just detect when determinization has produced more than a pre-set maximum number of states, then we prune with a tighter beam and try again. In future we may try more sophisticated methods such as a determinization algorithm that does pruning itself.

This "simple" version of the algorithm produces an acyclic, deterministic WFSA with words as labels. This is sufficient for applications such as language-model rescoring.

5.2. Keeping separate graph and acoustic costs

A fairly trivial extension of the algorithm described above is to store separately the acoustic costs and the costs arising from $HCLG$. This enables us to do things like generating output from the lattice with different acoustic scaling factors. We refer to these two costs as the graph cost and the acoustic cost, since the cost in $HCLG$ is not just the language model cost but also contains components arising from transition probabilities and pronunciation probabilities. We implement this by using a semiring that contains two real numbers, one for the graph and one for the acoustic costs; it keeps track of the two costs separately, but its $\oplus$ operation returns whichever pair has the lowest sum of costs (graph plus acoustic).

Formally, if each weight is a pair (a, b), then $(a, b) \otimes (c, d) = (a{+}c, b{+}d)$, and $(a, b) \oplus (c, d)$ is equal to (a, b) if $a{+}b < c{+}d$ or if $a{+}b = c{+}d$ and $a{-}b < c{-}d$, and otherwise is equal to (c, d). This is equivalent to the lexicographic semiring of [9], on the pair $((a{+}b), (a{-}b))$.

5.3. Keeping state-level alignments

It is useful for various purposes, e.g. discriminative training and certain kinds of acoustic rescoring, to keep the state-level alignments in the lattices. We will now explain how we can make the alignments "piggyback" on top of the computation defined above, by encoding them in a special semiring.

First, let us define $Q = \mathrm{inv}(P)$, i.e. Q is the inverted, pruned state-level lattice, where the input symbols are the words and the output symbols are the p.d.f. labels. We want to process Q in such a way that we keep only the best path through it for each word sequence, and get the corresponding alignment. This is possible by defining an appropriate semiring and then doing normal determinization. We shall ignore the fact that we are keeping track of separate graph and acoustic costs, to avoid complicating the present discussion.

We will define a semiring in which symbol sequences are encoded into the weights. Let a weight be a pair (c, s), where c is a cost and s is a sequence of symbols. We define the $\otimes$ operation as $(c, s) \otimes (c', s') = (c{+}c', (s, s'))$, where (s, s') is s and s' appended together. We define the $\oplus$ operation so that it returns whichever pair has the smallest cost: that is, $(c, s) \oplus (c', s')$ equals (c, s) if $c < c'$, and (c', s') if $c > c'$. If the costs are identical, we cannot arbitrarily return the first pair because this would not satisfy the semiring axioms. In this case, we return the pair with the shorter string part, and if the lengths are the same, whichever string appears first in dictionary order.

Let E be an encoding of the inverted state-level lattice Q as described above, with the same number of states and arcs; E is an acceptor, with its symbols equal to the input symbol (word) on the corresponding arc of Q, and the weights on the arcs of E containing both the the weight and the output symbol (p.d.f.), if any, on the corresponding arcs of Q. Let $D = \det(\mathrm{rmeps}(E))$. Determinization will always succeed because E is acyclic (as long as the original decoding graph $HCLG$ has no ϵ-input cycles). Because D is deterministic and ϵ-free, it has only one path for each word sequence. Determinization preserves equivalence, and equivalence is defined in such a way that the $\oplus$-sum of the weights of all the paths through E with a particular word-sequence, must be the same as the weight of the corresponding path through D with that word-sequence. It is clear from the definition of $\oplus$ that this path through D has the cost and alignment of the lowest-cost path through E that has the same word-sequence on it.

5.4. Summary of our algorithm

We now summarize the lattice generation algorithm. During decoding, we create a data-structure corresponding to a full state-level lattice. That is, for every arc of $HCLG$ we traverse on every frame, we create a separate arc in the state-level lattice. These arcs contain the acoustic and graph costs separately. We prune the state-level graph using a beam α; we do this periodically (every 25 frames) but this is equivalent to doing it just once at the end, as in [3]. Let the final pruned state-level lattice be P.

Let $Q = \mathrm{inv}(P)$, and let E be an encoded version of Q as described above (with the state labels as part of the weights). The final lattice is

$$L = \mathrm{prune}(\det(\mathrm{rmeps}(E)), \alpha). \quad (4)$$

The determinization and epsilon removal are done together by a single algorithm that we will describe below. L is a deterministic, acyclic weighted acceptor with the words as the labels, and the graph and acoustic costs and the alignments encoded into the weights. The costs and alignments are not "synchronized" with the words.

6. DETAILS OF OUR DETERMINIZATION ALGORITHM

We implemented ϵ removal and determinization as a single algorithm because ϵ-removal using the traditional approach would greatly increase the size of the state-level lattice (this is mentioned in [3]). Our algorithm uses data-structures specialized for the particular type of weight we are using. The issue is that the determinization process often has to append a single symbol to a string of symbols, and the easiest way to do this in "generic" code would involve copying the whole sequence each time. Instead we use a data structure that enables this to be done in linear time (it involves a hash table).

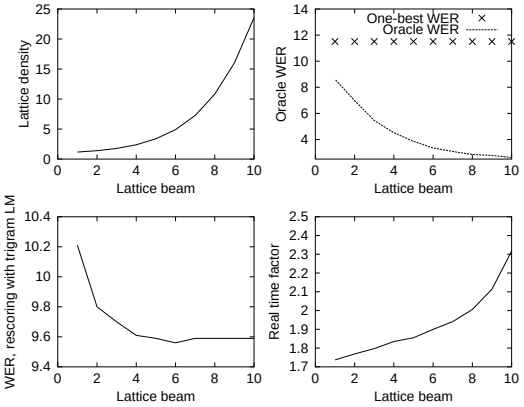

Fig. 2. Varying lattice beam α (Viterbi pruning beam fixed at 15)

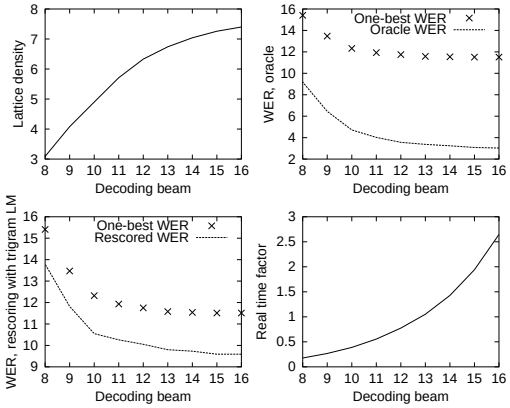

Fig. 3. Varying Viterbi pruning beam (lattice beam α fixed at 7)

We will briefly describe another unique aspect of our algorithm. Determinization algorithms involve weighted subsets of states, e.g.:

$$S = \{(s_1, w_1), (s_2, w_2), \ldots\}. \tag{5}$$

Let this weighted subset, as it appears in a conventional determinization algorithm with epsilon removal, be the *canonical representation* of a state. A typical determinization algorithm would maintain a map from this representation to a state index. We define a *minimial representation* of a state to be like the canonical representation, but only keeping states that are either final, or have non-ϵ arcs out of them. We maintain a map from the minimal representation to the state index. We can show that this algorithm is still correct (it will tend to give more minimal output). As an optimization for speed, we also define the *initial representation* to be the same type of subset, but prior to following through the ϵ arcs, i.e. it only contains the states that we reached by following non-ϵ arcs from a previous determinized state. We maintain a separate map from the initial representation to the output state index; think of this as a "lookaside buffer" that helps us avoid the expense of following ϵ arcs.

7. EXPERIMENTAL RESULTS

We report experimental results on the Wall Street Journal database of read speech. Our system is a standard mixture-of-Gaussians system trained on the SI-284 training data; we test on the November 1992 evaluation data. For these experiments we generate lattices with the bigram language model supplied with the WSJ database, and for rescoring experiments we use the trigram language model. The acoustic scale was $1/16$ for first-pass decoding and $1/15$ for LM rescoring. For simplicity we used a decoder that does not support a "maximum active states" option, so the only variables to consider are the beam used in the Viterbi beam search, and the separate beam α used for lattice generation.

Figure 2 shows how various quantities change as we vary α, with the Viterbi beam fixed at 15. Note that we get all the improvement from LM rescoring by increasing α to 4. The time taken by our algorithm started to increase rapidly after about $\alpha = 8$, so a value of α anywhere between about 4 and 8 is sufficient for LM rescoring and still does not slow down decoding too much. We do not display the real-time factor of the non-lattice-generating decoder on this data (2.26) as it was actually slower than the lattice generating decoder; this is presumably due to the overhead of reference counting. Out of vocabularly words (OOVs) provide a floor on the lattice oracle error rate: of 333 test utterances, 87 contained at least one OOV word, yet only 93 sentences (6 more) had oracle errors with $\alpha = 10$. Lattice density is defined as the average number of arcs crossing each frame. Figure 3 shows the effect of varying the Viterbi decoding beam, while leaving α fixed at 7.

8. CONCLUSIONS

We have described a lattice generation method that is to our knowledge the first efficient method not to rely on the word-pair assumption of [5]. It includes an ingenious way of obtaining state-level alignment information via determinization in a specially designed semiring.

9. REFERENCES

[1] D. Povey, A. Ghoshal, et al., "The Kaldi Speech Recognition Toolkit," in *Proc. ASRU*, 2011.

[2] Mehryar Mohri, Fernando Pereira, and Michael Riley, "Weighted finite-state transducers in speech recognition," *Computer Speech and Language*, vol. 20, no. 1, pp. 69–88, 2002.

[3] A. Ljolje, F. Pereira, and M. Riley, "Efficient General Lattice Generation and Rescoring," in *Proc. Eurospeech*, 1999.

[4] H. Sak, M. Saraçlar, and T. Güngör, "On-the-fly lattice rescoring for real-time automatic speech recognition," in *Proc. Interspeech*, 2010.

[5] S. Ortmanns and H. Ney, "A Word Graph Algorithm for Large Vocabulary Continuous Speech Recognition," *Computer Speech and Language*, vol. 11, pp. 43–72, 1997.

[6] S. Young, G. Evermann, M. J. F. Gales, T. Hain, D. Kershaw, X. Liu, G. Moore, J. J. Odell, D. Ollason, D. Povey, V. Valtchev, and P. Woodland, *The HTK Book (for version 3.4)*, Cambridge University Engineering Department, 2009.

[7] G. Saon, D. Povey, and G. Zweig, "Anatomy of an extremely fast LVCSR decoder," in *Proc. Interspeech*, 2005.

[8] J.J. Odell, *The use of context in large vocabulary speech recognition*, Ph.D. thesis, Cambridge University Engineering Dept., 1995.

[9] B. Roark, R. Sproat, and I. Shafran, "Lexicographic semirings for exact automata encoding of sequence models," in *Proc. ACL-HLT, 2011, Portland, OR*, 2011, pp. 1–5.