COMBINING FORWARD AND BACKWARD SEARCH IN DECODING

Mirko Hannemann[1], Daniel Povey[2], Geoffrey Zweig[3]

[1]Speech@FIT, Brno University of Technology, Brno, Czech Republic
[2]Center for Language and Speech Processing, Johns Hopkins University, Baltimore, MD USA
[3]Microsoft Research, Redmond, WA USA

ihannema@fit.vutbr.cz, dpovey@gmail.com, gzweig@microsoft.com

ABSTRACT

We introduce a speed-up for weighted finite state transducer (WFST) based decoders, which is based on the idea that one decoding pass using a wider beam can be replaced by two decoding passes with smaller beams, decoding forward and backward in time. We apply this in a decoder that works with a variable beam width, which is widened in areas where the two decoding passes disagree. Experimental results are shown on the Wall Street Journal corpus (WSJ) using the Kaldi toolkit, and show a substantial speedup (a factor or 2 or 3) at the "more accurate" operating points. As part of this work we also introduce a new fast algorithm for weight pushing in WFSTs, and summarize an algorithm for the time reversal of backoff language models.

Index Terms— speech decoding, beam width, search errors

1. INTRODUCTION

Due to the huge search spaces in speech decoding, it is necessary to use heuristic pruning techniques. The most used technique is beam search [1] - a breadth-first style search, comparing partial paths of the same length (time-synchronous). At each time only those paths are kept and further expanded, whose path score is better than the current best score extended by a beam width. The beam width is a trade-off between speed and accuracy. Usually, a constant beam width is applied to the whole test set.

The idea of this paper is to speed up decoding by using the (dis)agreement of two decoding passes - decoding forward and backward in time. The second decoding pass uses information gathered from the forward pass to increase the decoding beam in places where the two passes disagree. The speed-up is achieved by using a narrow beam during the forward pass, and in the backward pass in places where no disagreement is detected.

In order to implement this we need to be able to construct a decoding graph that operates "backwards" in time. In order to have good pruning behavior, this cannot just be the reverse of the "forwards" decoding graph, but must be constructed separately from reversed inputs. The hardest input to reverse was the ARPA-format language model, and we will describe in this paper how we create an equivalent but "time-reversed" language model for a given input.

We test our method on a Wall Street Journal decoding task. We find that our method gives a substantial speedup of two to three times or even more, at the "more accurate" operating points of decoding where search errors are small. However, in our setup, the speed-ups are diminishing for operating points faster than ≈ 0.6 real-time using our method. The issue seems to be that if the beams are too narrow, the two decoding passes disagree substantially and too much effort is expended in decoding areas that disagree.

Some of the work described here was done when the authors were at Microsoft Research, Redmond, WA. We thank Sanjeev Khudanpur for his input on the weight-pushing algorithm. This work was partly supported by the Intelligence Advanced Research Projects Activity (IARPA) BABEL program, the IT4Innovations Centre of Excellence CZ.1.05/1.1.00/02.0070 and Czech Ministry of Education project No. MSM0021630528.

2. RELATION TO PRIOR WORK

Using multiple decoding passes has been used for a long time (e.g. [2]). Usually inexpensive and approximate models are used in a first pass to generate an intermediate representation (e.g. N-best lists and word lattices) which is then re-scored using more complex models. [3] introduced the idea of performing the second pass backwards in time. From a Viterbi beam search in the forward pass they obtain the active words for each time frame and the corresponding word end scores. The former are used to limit the word expansion in backward search and the latter serve as a good estimate of the path cost of the remaining speech. Thus the second pass usually takes only a fraction of the time of the first pass, so that more complex algorithms or models can be used, or the forward pass can be sped up using approximate models [4]. A more recent re-discovery of the same idea is [5],[6] that use a word trellis and stack decoding (A-star) in the backward pass.

More similar to our idea is [7] (see also [8]), who use two symmetric forward and backward passes and combine the outputs based on confidence measures (Rover technique). Our technique has the advantage that it performs more careful search in areas where the two passes disagree, and so has a better chance to find the true lowest-cost path. Also, unlike the other citations, our algorithm uses the WFST approach [9] to speech recognition. We note that the baseline for our system is a basic WFST-based decoder. Other speed-ups, such as acoustic look-ahead [10] and various types of fast Gaussian score computation are also applicable, but we expect those types of methods to be complementary with the method we describe here.

3. DECODING GRAPHS FOR BACKWARDS DECODING

The experiments in this paper were conducted with the Kaldi toolkit [11]. The standard recipe for decoding graph creation is [9]:

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

where H, C, L and G represent the HMM structure, phonetic context-dependency, lexicon and grammar respectively, and $\circ$ is WFST composition (note: view HCLG as a single symbol). We use a "fully expanded" HCLG, i.e. the arcs correspond to HMM

transitions, the input labels are the identifiers of context-dependent HMM states, and the output labels represent words. For decoding forwards and backwards in time, we want to have two decoding graphs $\mathrm{HCLG}_{\mathrm{fwd}}$ and $\mathrm{HCLG}_{\mathrm{bwd}}$, which will assign the exact same overall cost for the same utterance. Because our method treats disagreement between the best paths found by the two methods as a search error, we want the backward decoding graph to be equivalent to the reverse of the forward one.

If we simply apply FST reversal to $\mathrm{HCLG}_{\mathrm{fwd}}$ to make $\mathrm{HCLG}_{\mathrm{bwd}}$ the search speed will be very slow, because the resulting FST will not be deterministic. Instead we construct the time-reversed versions of H, C, L and G, and construct the "backwards" graph in the normal way. These time reversed versions are not simply the WFST reverses of the forward ones, but must be separately constructed.

3.1. Reversing G, L, C and H

3.1.1. Reversing G

For the language model (LM), the task is to construct an LM acceptor G_{bwd}, that assigns exactly the same scores as G (to the reversed utterances). For our experiments, we used ARPA-format backoff LMs and we consider only how to reverse that type of LM. FST reversal is not sufficient because of the need to ensure that the reversed LM is deterministic and sufficiently "stochastic" (i.e. the transitions from each LM state should sum approximately to 1). A trivial solution is to train a new LM on the reversed training texts (e.g. [8]); however, we did not pursue this approach because i) it would not lead to exactly the same LM scores, and ii) it would make our approach inconvenient to use in cases where the original LM text was not available. We devised an approach to reverse the ARPA-format LM, which we summarize here. We may describe it in more detail in a future publication; regardless, the code for all the methods we describe here is available as part of the Kaldi toolkit. The sketch of our approach is as follows:

1. Modify the ARPA-format LM in such a way as to make the "backoff costs" zero while preserving sentence-level equivalence of scores, by pushing the costs onto higher-order N-gram scores. Our algorithm zeroes the backoff costs from lowest to highest order.
2. Convert the ARPA-format LM to a 'maxent'-like form of the ARPA, in which a probability is always multiplied in even if a higher-order one exists (this is done by subtracting lower-order from higher-order log-probabilities).
3. Reverse the 'maxent' form of the LM by replacing "A B" with "B A" and swapping the begin and end-of-sentence symbols.
4. Add (with zero log-probabilities) "missing" backoff states.
5. Convert from 'maxent' format back to standard ARPA format (but still with zero backoff costs).
6. Convert this 'un-normalized' ARPA into the WFST format.
7. Do our special form of weight pushing (See Section 4).

We have verified that our "reversed" WFST-format LM assigns the same score to a reversed sentence that our original WFST-format LM assigned to the original sentence.

3.1.2. Reversing L, C and H

The construction of the reversed pronunciation lexicon transducer L_{bwd} (phones to words) is simple: the individual phone sequences (pronunciations) are reversed, and the disambiguation symbols are introduced after that. The context-dependency transucer C_{bwd} is constructed in the normal way, and is identical to C_{fwd}. The HMM structure transducer H_{bwd}, is constructed in the same way as H_{fwd}, except with two differences. Firstly, the phonetic context windows corresponding to the input symbols of C are backwards in time, and must be reversed. Secondly, the HMMs that are constructed after using the decision tree to look up the relevant PDFs, must be reversed and then "weight-pushed" to make the time-reversed probabilities sum to one.

4. MODIFIED WEIGHT PUSHING

Weight pushing is a special case of *reweighting* [9], which is an operation on WFSTs that alters the weights of individual transitions (and final-probabilities), while leaving unaffected the weights on successful paths (i.e. from initial to final states). Weight pushing aims to alter a WFST so that the transitions and final-probability of each state "sums to one" in the semiring. It is only possible to do this if the "total weight" of the entire WFST is $\bar{1}$. Otherwise, there is a left-over weight that must be handled. In practice this may be discarded, or put on the initial or final state(s) of the WFST. In the case of language models, we want to do the pushing in the "log semiring", meaning we want each language model state to sum to one in a probability sense. Real backoff language models represented as WFSTs ([9]) will not exactly sum to one because the backoff structure leads to duplicate paths for some word sequences. In fact, such language models cannot be pushed at all in the general case, because the total weight of the entire WFST may not be finite. For our language model reversal we need a suitable pushing operation that will always succeed.

Our solution is to require a modified pushing operation such that each state "sums to" the same quantity. We were able to find an iterative algorithm that does this very efficiently in practice; it is based on the power method for finding the top eigenvalue of a matrix. Both for the math and the implementation, we find it more convenient to use the probability semiring, i.e. we represent the transition-probabilities as actual probabilities, not negative logs. Let the transitions be written as a sparse matrix $\mathbf{P}$, where p_{ij} is the sum of all the probabilities of transitions between state i and state j. As a special case, if j is the initial state, then p_{ij} is the final-probability of state i. In our method we find the dominant eigenvector $\mathbf{v}$ of the matrix $\mathbf{P}$, by starting from a random positive vector and iterating with the power method: each time we let $\mathbf{v} \leftarrow \mathbf{P}\mathbf{v}$ and then renormalize the length of $\mathbf{v}$. It is convenient to renormalize $\mathbf{v}$ so that v_I is 1, where I is the initial state of the WFST[1]. This generally converges within several tens of iterations. At the end we have a vector $\mathbf{v}$ with $v_I = 1$, and a scalar $\lambda > 0$, such that

$$\lambda \mathbf{v} = \mathbf{P}\mathbf{v}. \tag{2}$$

Suppose we compute a modified transition matrix $\mathbf{P}'$, by letting

$$p'_{ij} = p_{ij} v_j / v_i. \tag{3}$$

Then it is easy to show each row of $\mathbf{P}'$ sums to λ: writing one element of Eq. 2 as

$$\lambda v_i = \sum_j p_{ij} v_j, \tag{4}$$

it easily follows that $\lambda = \sum_j p'_{ij}$. We need to perform a similar transformation on the transition-probabilities and final-probabilities of the WFST; the details are quite obvious, and the equivalence with the original WFST is easy to show. Our algorithm is in practice an order of magnitude faster than the more generic algorithm for conventional weight-pushing of [12], when applied to cyclic WFSTs.

[1]Note: in order to correctly deal with the case of linear WFSTs, which have different eigenvalues with the same magnitude but different complex phase, we modify the iteration to $\mathbf{v} \leftarrow \mathbf{P}\mathbf{v} + 0.1\mathbf{v}$.

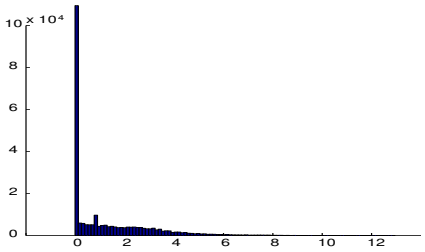

Fig. 1. *Histogram of score differences: current best path and final best path (decode beam 13.0, WSJ Nov'92 test set at WER 10.8%).*

5. TRACKED DECODING

Our decoding approach is to do a first pass (which happens to be a forward pass) with a narrow beam, and then to do a second pass in the opposite direction, also with a narrow beam, but using knowledge obtained during the first pass. The first pass outputs a lattice with state-level alignments [13]. Note that this does not contain everything visited in the first pass, but only those word-sequences that are within a specified beam of the best word-sequence. We want to treat the paths in the lattice in a special way in the second decoding. That is,

1. We want to avoid pruning out paths that appeared in the first-pass lattice.
2. On frames where we would otherwise have pruned out those paths, we want to increase the pruning beam.

Part of our motivation is that for most frames of speech data, a very narrow beam is sufficient. Fig. 1 plots a histogram of the score difference between the current best token, and the token that will be ultimately successful. Most of the time this difference is much smaller than the typical beam of between 10 and 15. We aim to use the initial forward pass to identify the problematic frames on the backward pass.

5.1. Tracking tokens with an arc-lattice

We need to be able to identify which tokens in our second-pass decoder correspond to paths in the first-pass lattice. One possible way to do this would be to designate a set of pdf-ids (context-dependent HMM states) on each frame that are "special" because they appear in the first pass lattices. But we did not pursue this because it could lead to too many irrelevant tokens being kept in the beam. Instead, we chose to identify those paths through the second-pass decoding graph that correspond to paths in the first-pass lattice. We implemented this as a separate process, outside the decoder code. It takes the standard lattice output by the first pass, and process it into something we call an *arc-lattice*, whose symbols identify arcs in our second-pass decoding graph $\mathrm{HCLG}_{\mathrm{2nd}}$. We explain the arc-lattice generation process below (Section 5.3).

The second-pass decoder, which we will refer to as our *tracking decoder*, is a lattice-generating decoder that takes an extra input, namely the arc-lattices for each utterance. Let a *token* be a record of a particular state in HCLG that is active on a particular frame. Our tracking decoder gives tokens an extra, boolean property that identifies whether they are *tracked* or not. A tracked token is one that corresponds to a state in the arc-lattice. Tracked tokens are never pruned. Tracked tokens are also used to determine the pruning beam used on each frame.

5.2. Beam-width policy

For the second-pass decoding with our tracking decoder, we use the *tracked* tokens to determine the beam width to use for each frame. Here we describe the policy we use to set the beam width. The decoder has three configurable values that specify how it sets the frame-specific beam: the *beam*, the *max-beam* and the *extra-beam*. On a particular frame, let the cost difference between the lowest-cost token and the highest-cost tracked token be D. Then the beam width on that frame is given by

$$\max(\text{beam}, \min(\text{max-beam}, D + \text{extra-beam})).$$

Unless otherwise specified we let *extra-beam* be zero and *max-beam* be large (e.g. 100, although this may be too large); we try various values of the *beam* for our experiments here.

Regardless of the beam-width, we never prune away the *tracked* tokens. Note that even if we kept the beam equal to *beam*, our method is doing more than simply choosing the best path from two (forward and backward) passes, because it is possible in this decoder for paths found by the first-pass search to "recombine" with paths that were found by the second-pass search.

5.3. Generation of the arc-lattice

As mentioned above, the arc-lattice is a special kind of lattice that allows us to identify arcs in $\mathrm{HCLG}_{\mathrm{2nd}}$ that were present in the first-pass lattice. This arc-lattice is an *acceptor* FST, i.e. it has only one symbol on each arc. These symbols correspond to arcs in $\mathrm{HCLG}_{\mathrm{2nd}}$. We first construct a mapping between the integers, and the individual arcs in $\mathrm{HCLG}_{\mathrm{2nd}}$; this involves creating tables for an integer mapping, because the product of (#states) $\times$ (maximum #arcs) may be greater than the 32-bit integer range.

We now describe how we create the arc-lattice. First, let us point out that the standard Kaldi lattices [13] are WFSTs whose input symbols correspond to integers called *transition-ids* and whose output symbols correspond to words. The transition-ids may be mapped to *pdf-ids*, which correspond to context-dependent HMM-states (the transition-ids contain more information, but it is not needed here). We first map the transition-ids to pdf-ids, and also map the input symbols of $\mathrm{HCLG}_{\mathrm{2nd}}$ from transition-ids to pdf-ids. This is necessary because the order of self-loops versus "forward transitions" on the forward versus backward graphs differ, which makes the sequences of transition-ids differ even for paths that are "really" the same; this issue does not arise with pdf-ids. We then change the output symbols of $\mathrm{HCLG}_{\mathrm{2nd}}$ (which were previously words) to symbols identifying the arc in $\mathrm{HCLG}_{\mathrm{2nd}}$. Let the resulting FST be called $\mathrm{HCLG}_{\mathrm{arc}}$; it has the same structure as $\mathrm{HCLG}_{\mathrm{2nd}}$ but different labels on the arcs. After doing the symbols mappings described above, we reverse the lattice (to switch the time order) and compose it with $\mathrm{HCLG}_{\mathrm{arc}}$. We apply *lattice-determinization* [13] to retain only the best path for each sequence of pdf-ids. We then "project it on the output", which means we keep only the output labels, corresponding to arcs in $\mathrm{HCLG}_{\mathrm{2nd}}$, and lattice-determinize again (this time on the output labels). Since the Kaldi lattices contained the alignments (sequence of pdf-ids), also the resulting arc-lattices contain timing information (sequences of $\mathrm{HCLG}_{\mathrm{2nd}}$-arcs, e.g. repeated self-loops).

During deocding, a token is *tracked* if it was reached by a sequence of $\mathrm{HCLG}_{\mathrm{2nd}}$-arcs in the arc-lattice that correspond to a path in the first pass lattice. If another token with lower cost reaches the same state at the same time, the tokens recombine, i.e. it replaces the token, but inherits the status of being tracked.

6. EXPERIMENTAL RESULTS

We tested the proposed decoding method on the WSJ Nov'92 open vocabulary test set (333 utterances) using a standard triphone HMM+GMM system (Kaldi [11] recipe 'tri2a', trained on 'si84' portion of WSJ). The experiments were conducted with the extended 146k vocabulary pruned trigram language model 'bd_tgpr' trained on all WSJ training texts. Lattices [13] were generated with a lattice beam of 4.0, and the realtime factor was measured on a single core of an Intel(R) CPU i5-2500 (3.3GHz, 8GB RAM).

Search errors can be evaluated by aligning the recognition output to a decoding with a very wide beam. We confirm the intuition that forward and backward search errors are independent by aligning forward and backward decoding outputs - Tables 1 and 2 show that most of the search errors were eliminated.

Table 1. *Analysis of search errors on WSJ Nov'92 test set by aligning forward and backward search errors against decoding with a wide beam (29.0) Error co-occurrence does not necessarily mean the same error. With two-pass (pingpong) decoding, all independent search errors were corrected, and even a good portion of the co-occurring errors.*

beamwidth	forwd.	backwd.	co-occur	pingpong
11.0	144	230	32	**14**
13.0	84	108	14	**6**

Table 2. *Alignment of search errors of forward (f), backwards (b) and ping-pong decoding (p) to decoding with very wide beam (w).('I' insert, 'S' substitute, '-' delete)*

```
f:BRIAN J.KILLING CHAIRMAN OF BELL - ATLANTA X.  INVESTMENT
     .    .    S      .     .    .    .    .    S      .
b:BRIAN J. DAILY  CHAIRMAN OF BELL AND LAND  SIX INVESTMENT
     .    .    .      .     .    .    I    S    S      .
p:BRIAN J. DAILY  CHAIRMAN OF BELL - ATLANTA ITS INVESTMENT
     .    .    .      .     .    .    .    .    .      .
w:BRIAN J. DAILY  CHAIRMAN OF BELL - ATLANTA ITS INVESTMENT
```

Fig. 2 shows, that for the lowest word error rates, the two-pass (ping-pong) decoding runs about 2-3 times faster than the individual forward/backward passes. The WER curve is not always smooth - it reminds that fixing a search error must not mean fixing a word error. We can compare the normal two-pass decoding with variable beam to decoding without generating extra tokens by disabling the variable beam ($maxbeam = beam$). This corresponds to just combining the lattices of the forward and backward pass. Fig. 2 shows that the variable beam ('2beam' vs. 'noextra') gives a substantial improvement on top of that.

We profiled the two-pass decoding in fig. 3. One question is why the two-pass decoding is not better than the one-pass decoding for higher error rates ($> 11.5\%$ in fig. 2). Going below a certain beam width, the error rates in the single passes grow rapidly (fig. 2) and also the divergence between the best paths from forward and backward decoding is increasing, so that the algorithm has to increase the variable beam a lot to track the first pass tokens. Thus, for low beam widths the most time consuming is the generation of extra tokens (fig. 3) which effectively means decoding with a higher beam.

7. CONCLUSIONS

We proposed how to integrate information from two decoding passes, forward and then backward in time. In the second (backward) pass, we modify the pruning behavior of the decoder to treat

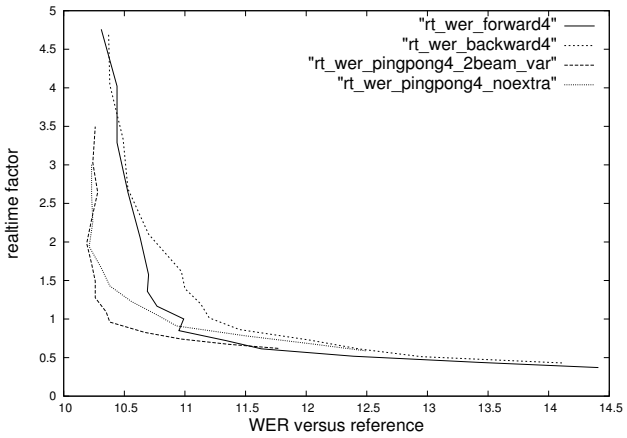

Fig. 2. *Shown are curves for word error rate (WER) vs. realtime factor on WSJ Nov'92 test set. For single pass decodings, the beam varies between 10-18, for the two-pass ('pingpong') decoding the beam varies between 7-13. We used $extrabeam = 0$ and found $maxbeam = 2 \cdot beam$ as a good tuning. The lattice-beam is 4.0, but for $beam < 10.0$ we decrease it stepwise to 0.5. We compare to decoding without generating extra tokens in the variable beam ('noextra') by setting $maxbeam = beam$, which shows the additional benefit of the variable beam over just combining lattices of forward and backward passes.*

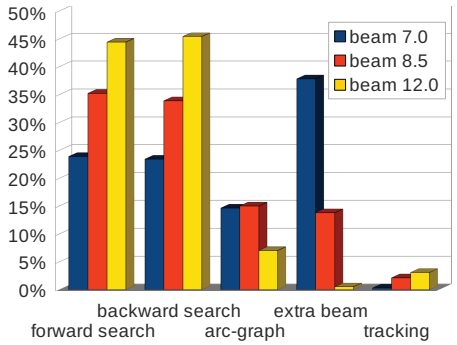

Fig. 3. *Profiling two-pass decoding. Shown is the percentage of time spent in different parts of the algorithm at 3 operating points (beam 8.5 as optimal, others as not optimal). The first pass is the lattice generating forward search and the second pass consists of a normal backward decoding (colum 2), generating the arc-lattice (col. 3), additionally tracking tokens from the first pass (col. 5) and generating extra tokens with the increased variable beam (col. 4). Acoustic scores were not cached between passes.*

specially tokens that were part of successful paths in the forward pass, and to increase the decoding beam for parts of the utterance where the forward and backward pass disagree. In order to do this we need to construct reverse decoding networks that assign exactly the same scores as the forward decoding. This required the development of a method to time-reverse ARPA format language models and a new algorithm for weight pushing. Our decoding method results in a roughly two to three-fold speed-up at lower WERs.

The proposed method could be applied in the fast generation of lattices for audio indexing and to generate lattices that contain certain desired paths (e.g. for discriminative training).

8. REFERENCES

[1] Bruce Lowerre, *The Harpy Speech Recognition System*, Ph.D. thesis, Carnegie Mellon University, 1976.

[2] H. Murveit, J. W. Butzberger, V. V. Digalakis, and M. Weintraub, "Large-vocabulary dictation using SRI's decipher speech recognition system: Progressive search techniques," in *Proc. ICASSP Vol. 2*, 1993, pp. 319–322.

[3] S. Austin, R. Schwartz, and P. Placeway, "The forward-backward search algorithm," in *Proc. ICASSP*, 1991, pp. 697–700.

[4] L. Nguyen, R. Schwartz, F. Kubala, and P. Placeway, "Search algorithms for software-only real-time recognition with very large vocabularies," in *Proceedings of the Workshop on Human Language Technology*, 1993, pp. 91–95.

[5] Akinobu Lee, Tatsuya Kawahara, and Shuji Doshita, "An efficient two-pass search algorithm using word trellis index," in *Proc. ICSLP*, 1998.

[6] Akinobu Lee and Tatsuya Kawahara, "Recent development of open-source speech recognition engine Julius," in *Proc. APSIPA Annual Summit and Conference*, 2009.

[7] Wafi Abo-Gannemhy, Itshak Lapidot, and H. Guterman, "Speech recognition using combined forward and backward Viterbi search," in *IEEE Convention of the Electrical and ELectronic Engineers in Israel*, 2010.

[8] Min Tang and Philippe Di Cristo, "Backward viterbi beam search for utilizing dynamic task complexity information," in *Proc. Interspeech*, 2008, pp. 2090–2093.

[9] Mehryar Mohri, Fernando C. N. Pereira, and Michael Riley, "Speech recognition with weighted finite-state transducers," in *Handbook on Speech Processing and Speech Communication, Part E: Speech recognition*, Larry Rabiner and Fred Juang, Eds., Heidelberg, Germany, 2008, p. 31, Springer-Verlag.

[10] D. Nolden, R. Schlüter, and H. Ney, "Acoustic look-ahead for more efficient decoding in LVCSR," in *Proc. Interspeech*, 2011.

[11] D. Povey, A. Ghoshal, G. Boulianne, L. Burget, O. Glembek, N. Goel, M. Hannemann, P. Motlicek, Y. Qian, P. Schwarz, J. Silovsky, G. Stemmer, and K. Vesely, "The Kaldi speech recognition toolkit," in *Proc. ASRU*. IEEE, 2011.

[12] Mehryar Mohri, "Semiring frameworks and algoritms for shortest-distance problems," *Journal of Automata, Languages and Combinatorics*, vol. 7, pp. 321–350, March 2002.

[13] D. Povey, M. Hannemann, G. Boulianne, L. Burget, A. Ghoshal, M. Janda, M. Karafiat, S. Kombrink, P. Motlicek, Y. Quian, N. Thang Vu, K. Riedhammer, and K. Vesely, "Generating exact lattices in the WFST framework," in *Proc. ICASSP*. IEEE, 2012, pp. 4213–4216.