LET-Decoder: A WFST-based Lazy-evaluation Token-group Decoder with Exact Lattice Generation

Hang Lv, *Student Member, IEEE*, Daniel Povey, Mahsa Yarmohammadi, Ke Li, *Student Member, IEEE*, Yiming Wang, Lei Xie, *Senior Member, IEEE*, Sanjeev Khudanpur, *Member, IEEE*

***Abstract*—We propose a novel lazy-evaluation token-group decoding algorithm with on-the-fly composition of weighted finite-state transducers (WFSTs) for large vocabulary continuous speech recognition. In the standard on-the-fly composition decoder, a base WFST and one or more incremental WFSTs are composed during decoding, and then token passing algorithm is employed to generate the lattice on the composed search space, resulting in substantial computation overhead. To improve speed, the proposed algorithm adopts 1) a token-group method, which groups tokens with the same state in the base WFST on each frame and limits the capacity of the group and 2) a lazy-evaluation method, which does not expand a token group and its source token groups until it processes a word label during decoding. Experiments show that the proposed decoder works notably up to 3 times faster than the standard on-the-fly composition decoder.**

***Index Terms*—Speech recognition, WFST, on-the-fly composition, on-the-fly lattice rescoring**

I. Introduction

A decoder plays an important role in an automatic speech recognition (ASR) system where it integrates acoustic and language information to generate the most likely word sequence for an input speech signal. It is necessary for many applications that the decoded results are in the form of lattices [1]–[3] or N-best hypotheses lists [4], [5] since they are more informative than just one best hypothesis. Various lattice generation methods have been proposed, such as the word/phone pair assumption [2], [6]–[8], the N-best histories method [9], [10], and the exact lattice generation method [11] that is the most widely used compared with others.

Decoders based on weighted finite-state transducers (WFSTs) [12] can efficiently compose various source information, including acoustic, phonetic context decision tree, lexicon, and language model. Thanks to operations such as determinization and minimization in WFST making the decoders very compact, the WFST-based decoders generally work more efficiently, in a pithy and elegant manner than other classical approaches [13]. However, when individual models of knowledge sources become huge, such as high-order language model (LM) or million-word lexicon, the composed WFST can be memory-inefficient or even infeasible to construct.

To overcome this problem, two kinds of solutions have been proposed. The first one is multiple-pass decoding [8], [14], [15] which usually breaks up the decoding process into two stages. Fast, efficient (relatively small) models are used to generate restrict-sized lattices first. They are then rescored with richer knowledge sources for better performance. While the potential of the two-pass method is limited by the relatively small knowledge sources used in the first pass. The two-pass procedure makes the latency issue unavoidable. Such method is thus more suitable for offline tasks. The second method is on-the-fly (aka. on-demand or lazy) composition [16]–[18], in which WFSTs are separated into two (or more) groups and dynamically composed when needed. As such, it reduces memory usage and is more flexible than offline composition. However, decoding becomes slower since the search space is not optimised as well as the offline composition does. Moreover, on-the-fly composition results in extra computational overhead within decoding. Researchers proposed several algorithms to optimize on-the-fly composition, such as look-ahead composition [19], [20] and on-the-fly hypothesis rescoring under phone pair assumption [21]–[23].

This paper proposes a novel method to optimize the on-the-fly composition decoding for exact lattice generation [11]. The proposed innovative WFST-based decoder is denoted as *Lazy-evaluation Token-group decoder* (LET-Decoder). This work makes contributions from three aspects: 1) we propose a token-group method and apply lazy-evaluation method with it for speedup; 2) we employ "bucketqueue" to implement histogram pruning in a more natural way; and 3) we develop an online version of the LET-decoder. Experiments show that our proposed decoder can achieve up to 3 times relative speedup. This work is open-sourced under Kaldi [24][1]. The approach in [22] mostly relates to our work. That approach performs the Viterbi search in the first WFST and rescores the hypotheses on word level in the second WFST. However, their implementation includes an approximation during token recombination that may affect word accuracy, and a declination of time alignment can occurs. In contrast, our proposed decoder can perform exact decoding and generate exact lattices.

Hang Lv was with CLSP, Johns Hopkins University, Baltimore, MD, USA. He is with ASLP@NPU, School of Computer Science, Northwestern Polytechnical University, Xi'an, China. (e-mail:hanglv@nwpu-aslp.org)

Daniel Povey is with Xiaomi, Beijing, China. (e-mail: dpovey@xiaomi.com)

Ke Li and Yiming Wang are with CLSP, Johns Hopkins University, Baltimore, MD, USA. (e-mail: keli26, yiming.wang@jhu.edu)

Lei Xie is with ASLP@NPU, School of Computer Science, Northwestern Polytechnical University, Xi'an, China. (e-mail: lxie@nwpu-aslp.org)

Mahsa Yarmohammadi and Sanjeev Khudanpur are with CLSP and Human Language Technology Center of Excellence, Johns Hopkins University, Baltimore, MD, USA. (e-mail: mahsa,khudanpur@jhu.edu)

[1]https://github.com/LvHang/kaldi/tree/bucket-d

II. Basic Decoder

This section briefly reviews background work closely related to the developed approach, including the WFST-based decoder and the BigLM decoder.

A. WFST-based Decoder

The standard *decoding graph* employed in Kaldi is similar to that in [12], where the WFST decoding graph is denoted by

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

where H, C, L, and G represent the Hidden Markov Model (HMM) structure, phonetic context-dependency, lexicon, and grammar, respectively, and $\circ$ denotes the *composition* operation of WFSTs. On an arc in *HCLG*, the input label is the identifier of a clustered context-dependent HMM state, the output label corresponds to a word, and the weight typically represents a negated log-probability.

In general, the *token passing* algorithm [25] is used to decode a T-frame utterance by composing the acoustic log-likelihood graph U with the *HCLG* graph.

$$W \equiv U \circ HCLG \tag{2}$$

The decoding task is effectively to find the best path through the *search graph* W of the utterance. In particular, W may have approximately $(T+1)$ times more states compared with *HCLG* itself when a precise search is performed. To save memory and decoding time, pruning methods [26], [27] are adopted in practise.

Notably, a token, which is indexed by *HCLG-state* at each frame step, represents the potential decoding information of an input utterance up to the current frame. The acoustic and graph costs are kept separately so that re-scaling and rescoring with higher-order LM subsequently are convenient.

B. BigLM Decoder

The basic idea of the on-the-fly composition decoder, denoted as *BigLM decoder*, is to create the decoding graph *HCLG* with a small (e.g. low-order or pruned) LM, and compose dynamically with a WFST representing the difference between a relatively large LM and the small one. The resulting decoding graph S_{big} can thus be obtained as follows:

$$G_r = G^{-} \circ G' \tag{3}$$

$$S_{big} = HCLG \circ G_r \tag{4}$$

where G and G' are the small and large LMs, respectively. Here, G^{-} is topologically analogous to G but with negated weights, and G_r is referred to *residual grammar*. In practice, *HCLG* and G_r are stored separately. Then the process of decoding is performed as follows:

$$W_{big} = U \circ S_{big} \equiv U \circ HCLG \circ G_r \tag{5}$$

Compared to the traditional decoder introduced in Section II-A where a token is represented by a pair (*frame-index*, *HCLG-state*), here a 3-tuple (*frame-index*, *HCLG-state*, G_r-*state*) is recorded for each token. At each time, the token passing executes the following steps:

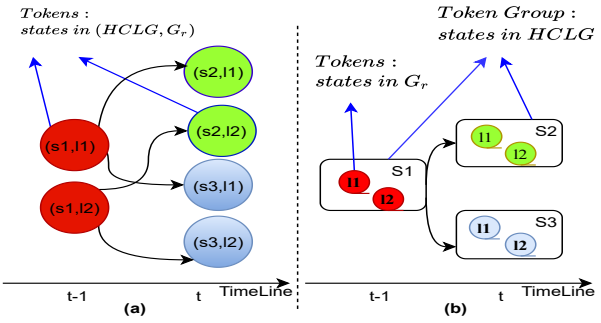

Fig. 1. The comparison of (a) tokens in BigLM decoder and (b) tokens in token-group method. For example, the states with the same color in (a) are grouped in the box in (b). (s,l) is short for (HCLG-state, G_r-state) pair space.

1) Obtain current *HCLG-state* of a token, pass it one step forward in the graph *HCLG*, record the output label on that arc, and derive a new *HCLG-state'*.
2) Obtain current G_r-*state* of a token, pass it one step forward in the graph G_r by treating the output label of *HCLG-state* as input, and derive a new G_r-*state'*.
3) Generate a new triple tuple with the *HCLG-state'* and G_r-*state'*. The frame index is dependent upon the input label (i.e.ϵ or not) in the graph *HCLG*.

The BigLM decoder is memory-efficient so that richer knowledge sources can be involved in one-pass decoding. But it is slow due to the computational overhead introduced by composition during decoding. As described above, the search space of the BigLM decoder is (*HCLG-state*, G_r-*state*). G_r works only when a token crosses a word boundary. Since word output arcs are much less frequent than non-word output arcs, there are a lot of *repeated operations* for tokens with the same *HCLG-state* and different G_r-*state*s (e.g. Fig. 1(a)). Furthermore, as pruning methods are always applied inside the HMM structures of words, some tokens cannot reach the next word boundary, and it brings further *wasted operations*. To overcome these drawbacks, we propose LET-Decoder in the following section.

III. Lazy-evaluation Token-group Decoder

In this section, we describe our LET-Decoder, which is extended from the BigLM decoder under the exact lattice generation method.

A. Token-group structure and lazy-evaluation method

In general, to eliminate the "repeated operations" mentioned in II-B and also exemplified in Fig. 1(a), we group the tokens with same *HCLG-state* on each frame so that we can use unified token-group level operations to replace repeated token level operations, as shown in Fig. 1(b). To avoid the "wasted operations", we employ the lazy-evaluation method which will not fill tokens into token-groups until word output arcs are reached, as shown in Fig. 2. If the token-groups are early pruned, we can save the operations of generating tokens. We give the details of our implementation as follows.

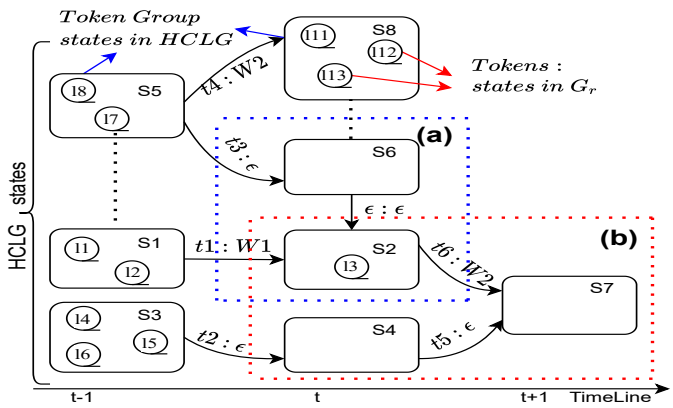

Fig. 2. The sketch of token-group structure and lazy-evaluation method.

First, we introduce the token-group concept. In general, we group together the tokens with the same *HCLG-state* but different G_r*-states* on each frame. The characteristics of the token-group (refer to *Token Group* in Fig. 2) are:

- Forwarding-prob of the best token.
- A finite length heap used to organize the real tokens, aiming to avoid unpromising hypotheses.
- Preceding token-group links which speed up back-tracing.
- Succeeding token-group links which record corresponding acoustic and LM information.
- "Expanded" or "not-expanded" status, which identifies the tokens has been generated or not. "Not-expanded" token-group only originates from non-word output arcs.

Second, we apply lazy-evaluation on top of the token-group structure. The core idea is that we do not expand (i.e. fill in "real" tokens) the token-group until a word label is emitted. In Fig. 2, we show all kinds of possible expansion cases. In particular, two kinds of operations are as follows:

- Traverse a non-word output arc and go into a "non-expanded" token-group: construct group level links only.
- Traverse a word output arc (case (b) in Fig 2) or go into an "expanded" token-group (case (a)/(b) in Fig 2): trace back from the destination token-group by following all the preceding links until an "expanded" token-groups is encountered in each path, and then expand the tokens into corresponding token-groups along all the paths.

Notably, during the process of back-tracing, our lazy-evaluation token-group design guarantees that updating the token-group which is not on the current frame does not change its forward-prob. We denote this as α*-stable* property, and it is convenient for pruning methods and generating exact lattices.

Comparing to the BigLM decoder, lazy-evaluation token-group design has the following clear advantages: 1) When traversing the *HCLG* graph, we use one unified token-group operation to replace the duplicated token operations (step1 of II-B). 2) If "non-expanded" token-group is pruned, the token level operations are saved (step2 of II-B). 3) When tracing back to produce the tokens, we use the information on succeeding token-group links directly, so that we can avoid retrieving the *HCLG* graph and acoustic model (II-A).

B. BucketQueue for histogram pruning

For general WFST-based decoders, normally emitting and non-emitting arcs are processed separately. The destination tokens of non-emitting/emitting arcs only appear on the current/next frame so that histogram pruning is convenient. But we have to iterate over all outgoing arcs of one state in the decoding graph twice. To avoid this issue, we process these two types of arcs simultaneously (denoted as "ProcessUnify") to reduce the redundant iteration.

As "ProcessUnify" makes the destination tokens also be produced at the current frame, we need a new method to do histogram pruning. We employ the "BucketQueue" data structure, which is a priority queue for prioritizing elements whose priorities are integers. It is implemented with a priority-indexed array where elements in the same bucket have the same priority. In practice, the token-groups/tokens are inserted into the queue according to the integer part of their forward-probs. Compared with the conventional decoders, the "BucketQueue" method, which exactly controls the number of processed elements, is a more natural way to implement "histogram pruning".

C. Online Lazy-evaluation Token-group Decoder

Online decoding requires generating partial results in real time. Although an extra backward pointer is stored in token structure as general decoders, our LET-decoder still needs to deal with two issues: 1) "ProcessUnify" leads to the last token-group list half-baked when an utterance is not finished. 2) Token-groups may stay on "not-expanded" status. To resolve the first issue, we can collect best token-group on the penultimate/last token-group list according to the status of the utterance, e.g. ongoing or finished. For the second issue, because of the α*-stable* property, back-tracing from token-group/token does not lose accuracy.

After figuring out the above issues, we can perform online decoding and generate partial results in real time. We find the best token-group on the penultimate/last frame in the first place, and then trace back along the narrow-band preceding token-group links to an "expanded" token-group. We then follow the backward token pointer to recover the remaining part of the path. The whole back-tracing path can be regarded as the partial best path.

IV. Experimental Results

A. Experimental Setup

1) LibriSpeech corpus: Systematic evaluation of the proposed LET-decoder is conducted using open-source speech recognition toolkit Kaldi [24]. We experiment with the LibriSpeech (LIB) corpus [28], which contains approximately 960 hours training data and four testing datasets (i.e. {dev/test}_{clean/other}), of which each has about two hours audio data. The "other" test sets are more challenging compared to the "clean" ones, and prone to cause search errors for decoding algorithms.

We used a time-delay deep neural network (TDNN) model [29], [30] trained by lattice-free maximum mutual information (LF-MMI) criterion [31]. For the LibriSpeech testing,

the customary standard LMs are used [28]. The small 3-gram LM (60MB) is used to build the *HCLG* decoding graph. The mild-pruned 3-gram LM (tgmed, 140MB), original 3-gram LM (tglarge, 760MB) and original 4-gram LM (fglarge) are used to construct *residual grammar* G_r respectively.

2) Mandarin-English code-switching corpus: An industry-level internal Mandarin-English code-switching model is also used to evaluate the LET-decoder. A "CNN-TDNN" model is trained by LF-MMI criterion with 20,000 hours Mandarin-English data. The experiments are conducted on an internal Mandarin 3.5 hours test set (test-pure) and the offical ASRU Mandarin-English code-switching challenge testset [32] (test-switch). The 950k-vocabulary LMs are trained from 48GB Mandarin-English text. The *HCLG* decoding graph is built with a 2-gram LM (1GB). The 2.3GB 3-gram LM (tgmed), 4.4GB 3-gram LM (tglarge) and 5.6GB 3-gram LM (fglarge) are used to build *residual grammar* separately.

B. Performance

In Table I, we compare the WERs (MERs [32] for code-switching testset) among the two-pass lattice rescoring method, BigLM decoder, and our proposed LET-decoder. It shows that WERs of the three methods are close, while the last two methods outperform the first marginally. The better performance results from the more accurate LM information of the relatively large LM used in decoding. Compared with the WER criterion, the average log-likelihood of lattices provides more accurate information for decoder evaluation. We thus compare the average log-likelihood between BigLM decoder and the proposed one. We observe comparable accuracy of the two decoders under the equivalent pruning condition (i.e. beam=15, lat-beam=8, etc).

TABLE I
WER STATISTICS: RESCORING / BIGLM DECODER/ LET-DECODER

	tgmed	tglarge	fglarge
dev-clean	4.27/4.25/4.26	3.38/3.38/3.38	3.27/3.28/3.27
dev-other	11/11/11.01	9.14/9.1/9.1	8.7/8.59/8.6
test-clean	4.74/4.77/4.77	3.94/3.93/3.92	3.83/3.82/3.82
test-other	11.2/11.18/11.19	9.21/9.17/9.18	8.72/8.65/8.67
test-pure	3.82/3.82/3.82	3.68/3.66/3.67	3.63/3.61/3.63
test-switch	3.33/3.08/3.24	3.32/3.07/3.17	3.33/3.07/3.14

C. Speedup

We compare the real time factor (RTF) between BigLM decoder and LET-decoder with the same pruning for both. We observe about 1-fold to 3-fold speedups with a single process as shown in Table II.

TABLE II
RTF: BIGLM DECODER/LET-DECODER

	tgmed	tglarge	fglarge
dev-clean	0.1365 / 0.0599	0.1435 / 0.0649	0.2341 / 0.0709
dev-other	0.2698 / 0.1496	0.2799 / 0.1631	0.3127 / 0.1647
test-clean	0.1314 / 0.0606	0.1401 / 0.0728	0.1548 / 0.0741
test-other	0.2679 / 0.1625	0.2838 / 0.1756	0.3151 / 0.2158
test-pure	0.7151 / 0.2952	0.7561 / 0.3147	0.7442 / 0.3149
test-switch	0.5803 / 0.2728	0.5716 / 0.2770	0.6463 / 0.3039

We further analyze the proposed LET-decoder to show the reasons of speedup. Experiments are reported on the LIB dataset. The same trend is observed from the code-switching corpus.

1) Token-group structure without lazy-evaluation method: To investigate how much the token-group structure and lazy-evaluation method contribute to the speedup individually. So we test our token-group structure without lazy-evaluation (i.e. always expand token-groups at once) method. There is not significant speedup over the BigLM decoder. We believe the extra overhead of maintaining the token-group structure undermines the speed improvement by eliminating the repeated operations via using the token-group structure. So the speedup mainly comes from the lazy-evaluation method.

2) The effect of token-group size: Figure 3(a) shows the effect of token-group size in the proposed LET-decoder. Generally, larger speedups are achieved with smaller token-group size, but differences are not apparent. We further explore the saturation of each token-group. Fig. 3(b) shows that each group contains about two tokens on average, which leads to the unapparent gap. But there are still some token-groups contain dozens of real tokens and most of them are located in the narrow band around the best path.

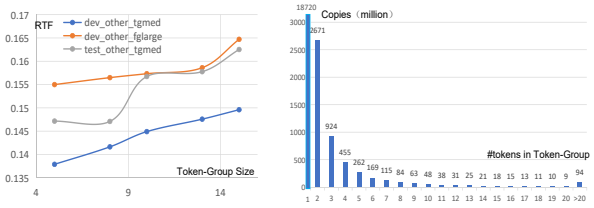

Fig. 3. (a) the relationship between RTF and token-group size. (b) an example to show the number of real tokens in token-group.

3) The reduction of on-the-fly composition operations: The number of on-the-fly composition operations presents the number of times we advance G_r states. Table III shows the number of these operations on the BigLM decoder and LET-decoder. Compared with the BigLM decoder, only about 1/10 operations are executed for the proposed LET-decoder, accelerating the decoding drastically as a result.

TABLE III
ON-THE-FLY COMPOSITION OPERATIONS (MILLION): BIGLM DECODER/LET-DECODER

	tgmed	tglarge	fglarge
dev-clean	3805 / 196	3462 / 185	3501 / 201
dev-other	7459 / 563	6955 / 556	7108 / 558
test-clean	3986 / 211	3634 / 202	3687 / 220
test-other	7945 / 596	7389 / 583	7550 / 644

V. CONCLUSIONS

In this paper, we introduce a Lazy-evaluation Token-group decoder to speedup the on-the-fly composition method with exact lattice generation. The proposed decoder achieves 1-to-3 fold speedup so that high-order language models can be efficiently used in the one-pass decoding.

REFERENCES

[1] H. Murveit, J. Butzberger, V. Digalakis, and M. Weintraub, "Large-vocabulary dictation using sri's decipher speech recognition system: Progressive search techniques," in *1993 IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 2. IEEE, 1993, pp. 319–322.

[2] X. Aubert and H. Ney, "Large vocabulary continuous speech recognition using word graphs," in *1995 International Conference on Acoustics, Speech, and Signal Processing*, vol. 1. IEEE, 1995, pp. 49–52.

[3] S. Young, G. Evermann, M. Gales, T. Hain, D. Kershaw, X. Liu, G. Moore, J. Odell, D. Ollason, D. Povey *et al.*, "The HTK Book," *Cambridge University Engineering Department*, 2002.

[4] S. Young, "Generating multiple solutions from connected word dp recognition algorithms," *Proc. of the Institute of Acoustics*, vol. 6, no. Part 4, pp. 351–354, 1984.

[5] Y.-L. Chow and R. Schwartz, "The n-best algorithm: An efficient procedure for finding top n sentence hypotheses," in *Proceedings of the workshop on Speech and Natural Language*. Association for Computational Linguistics, 1989, pp. 199–202.

[6] H. Ney and X. Aubert, "A word graph algorithm for large vocabulary, continuous speech recognition," in *Third International Conference on Spoken Language Processing*, 1994.

[7] S. Ortmanns, H. Ney, and X. Aubert, "A word graph algorithm for large vocabulary continuous speech recognition," *Computer Speech & Language*, vol. 11, no. 1, pp. 43–72, 1997.

[8] A. Ljolje, F. Pereira, and M. Riley, "Efficient general lattice generation and rescoring," in *Sixth European Conference on Speech Communication and Technology*, 1999.

[9] G. Saon, D. Povey, and G. Zweig, "Anatomy of an extremely fast lvcsr decoder," in *Ninth European Conference on Speech Communication and Technology*, 2005.

[10] S. F. Chen, B. Kingsbury, L. Mangu, D. Povey, G. Saon, H. Soltau, and G. Zweig, "Advances in speech transcription at ibm under the darpa ears program," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 14, no. 5, pp. 1596–1608, 2006.

[11] D. Povey, M. Hannemann, G. Boulianne, L. Burget, A. Ghoshal, M. Janda, M. Karafiát, S. Kombrink, P. Motlíček, Y. Qian *et al.*, "Generating exact lattices in the wfst framework," in *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2012, pp. 4213–4216.

[12] M. Mohri, F. Pereira, and M. Riley, "Weighted finite-state transducers in speech recognition," *Computer Speech & Language*, vol. 16, no. 1, pp. 69–88, 2002.

[13] S. Kanthak, H. Ney, M. Riley, and M. Mohri, "A comparison of two lvr search optimization techniques," in *Seventh International Conference on Spoken Language Processing*, 2002.

[14] A. Stolcke, Y. Konig, and M. Weintraub, "Explicit word error minimization in n-best list rescoring," in *Fifth European Conference on Speech Communication and Technology*, 1997.

[15] H. Xu, T. Chen, D. Gao, Y. Wang, K. Li, N. Goel, Y. Carmiel, D. Povey, and S. Khudanpur, "A pruned rnnlm lattice-rescoring algorithm for automatic speech recognition," in *2018 IEEE international conference on acoustics, speech and signal processing (ICASSP)*. IEEE, 2018, pp. 5929–5933.

[16] M. Riley, F. Pereira, and M. Mohri, "Transducer composition for context-dependent network expansion," in *Fifth European Conference on Speech Communication and Technology*, 1997.

[17] H. J. Dolfing and I. L. Hetherington, "Incremental language models for speech recognition using finite-state transducers," in *IEEE Workshop on Automatic Speech Recognition and Understanding, 2001. ASRU'01.* IEEE, 2001, pp. 194–197.

[18] T. Hori and A. Nakamura, "Speech recognition algorithms using weighted finite-state transducers," *Synthesis Lectures on Speech and Audio Processing*, vol. 9, no. 1, pp. 1–162, 2013.

[19] D. Nolden, H. Ney, and R. Schlüter, "Exploiting sparseness of backing-off language models for efficient look-ahead in lvcsr," in *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2011, pp. 4684–4687.

[20] D. Nolden, R. Schlüter, and H. Ney, "Advanced search space pruning with acoustic look-ahead for wfst based lvcsr," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, 2013, pp. 6734–6738.

[21] T. Hori, C. Hori, and Y. Minami, "Fast on-the-fly composition for weighted finite-state transducers in 1.8 million-word vocabulary continuous speech recognition," in *Eighth International Conference on Spoken Language Processing*, 2004.

[22] T. Hori, C. Hori, Y. Minami, and A. Nakamura, "Efficient wfst-based one-pass decoding with on-the-fly hypothesis rescoring in extremely large vocabulary continuous speech recognition," *IEEE Transactions on audio, speech, and language processing*, vol. 15, no. 4, pp. 1352–1365, 2007.

[23] H. Sak, M. Saraclar, and T. Güngör, "On-the-fly lattice rescoring for real-time automatic speech recognition," in *Eleventh annual conference of the international speech communication association*, 2010.

[24] D. Povey, A. Ghoshal, G. Boulianne, L. Burget, O. Glembek, N. Goel, M. Hannemann, P. Motlicek, Y. Qian, P. Schwarz *et al.*, "The Kaldi speech recognition toolkit," in *2011 IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*. IEEE, 2011.

[25] S. J. Young, N. Russell, and J. Thornton, *Token passing: a simple conceptual model for connected speech recognition systems*. Cambridge University Engineering Department Cambridge, 1989.

[26] H. Van Hamme and F. Van Aelten, "An adaptive-beam pruning technique for continuous speech recognition," in *Proceeding of Fourth International Conference on Spoken Language Processing. ICSLP'96*, vol. 4. IEEE, 1996, pp. 2083–2086.

[27] K. Kashino, T. Kurozumi, and H. Murase, "A quick search method for audio and video signals based on histogram pruning," *IEEE Transactions on Multimedia*, vol. 5, no. 3, pp. 348–357, 2003.

[28] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur, "Librispeech: an asr corpus based on public domain audio books," in *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2015, pp. 5206–5210.

[29] V. Peddinti, D. Povey, and S. Khudanpur, "A time delay neural network architecture for efficient modeling of long temporal contexts," in *Sixteenth Annual Conference of the International Speech Communication Association*, 2015.

[30] V. Peddinti, Y. Wang, D. Povey, and S. Khudanpur, "Low latency acoustic modeling using temporal convolution and lstms," *IEEE Signal Processing Letters*, vol. 25, no. 3, pp. 373–377, 2017.

[31] D. Povey, V. Peddinti, D. Galvez, P. Ghahremani, V. Manohar, X. Na, Y. Wang, and S. Khudanpur, "Purely sequence-trained neural networks for asr based on lattice-free mmi." in *Interspeech*, 2016, pp. 2751–2755.

[32] X. Shi, Q. Feng, and L. Xie, "The asru 2019 mandarin-english code-switching speech recognition challenge: Open datasets, tracks, methods and results," *arXiv preprint arXiv:2007.05916*, 2020.