AN ASYNCHRONOUS WFST-BASED DECODER FOR AUTOMATIC SPEECH RECOGNITION

Hang Lv[1,2]*, Zhehuai Chen*[2,5]*, Hainan Xu*[2]*, Daniel Povey*[4]*, Lei Xie*[1]*, Sanjeev Khudanpur*[2,3]

[1] Audio, Speech and Language Processing Lab (ASLP@NPU),
School of Computer Science, Northwestern Polytechnical University, Xi'an, China
[2] Center of Language and Speech Processing, [3] Human Language Technology Center of Excellence,
Johns Hopkins University, Baltimore, MD, USA
[4] Xiaomi Corporation, Beijing, China
[5] SpeechLab, Department of Computer Science and Engineering, Shanghai Jiao Tong University
{hanglv,lxie}@nwpu-aslp.org, chenzhehuai@sjtu.edu.cn, {hxu31,khudanpur}@jhu.edu, dpovey@xiaomi.com

ABSTRACT

We introduce *asynchronous dynamic decoder*, which adopts an efficient A* algorithm to incorporate big language models in the one-pass decoding for large vocabulary continuous speech recognition. Unlike standard one-pass decoding with on-the-fly composition decoder which might induce a significant computation overhead, the asynchronous dynamic decoder has a novel design where it has two fronts, with one performing "exploration" and the other "backfill". The computation of the two fronts alternates in the decoding process, resulting in more effective pruning than the standard one-pass decoding with an on-the-fly composition decoder. Experiments show that the proposed decoder works notably faster than the standard one-pass decoding with on-the-fly composition decoder, while the acceleration will be more obvious with the increment of data complexity.

Index Terms— Automatic speech recognition, decoder, lattice generation, lattice pruning

1. INTRODUCTION

Automatic speech recognition (ASR) technologies have been widely and successfully applied in many real-world fields with recent advances in deep learning algorithms, thanks to the availability of ever increasing computational power. In particular, acoustic model (AM) inference and decoding are the main computing consumption parts of an ASR system. Researchers have proposed a variety of efficient methods to speedup acoustic model inference, including novel acoustic structures [1], frame-skipping [2] and quantization [3, 4].

At the same time, the decoding technology is also constantly developing. In the decoding field, the core problem is how to generate an accurate lattice. Having a high quality lattice allows post-processing steps to further improve performance, e.g. lattice rescoring. A common assumption underlying lattice generation methods is the word-pair assumption. In [5], a tree-based word graph generation method is proposed to generate the lattice. In the recent decade, the weighted finite-state transducer (WFST) based lattice generation method is applied to decoders [6]. In [7], the decoder is expanded down to the context-dependent phone level (i.e. CLG). After that, the algorithm in [8] is applied to WFSTs, expanded it down to the context-dependent state level (i.e. HCLG), which store the information of scores and state-level alignments. Normally, the larger the language model (LM) is employed, the more accurate the lattice will be generated. Nevertheless, because of the limitation of memory, we commonly perform a two-pass decoding, where in the first pass we generate lattices with a small (e.g. low-order or pruned) LM, and the lattice is re-scored with a relatively big LM in the second pass [9, 10]. But the two-pass procedure makes the latency issue unavoidable. To overcome it, a useful approach is to perform one-pass on-the-fly (also called on-demand) composition decoding, in which a decoding graph with a small LM is created and then composed with a graph representing the difference between a large LM and the small LM as needed, in order to generate the search space dynamically during decoding. However, the decoding speed decreases as the search space is not as optimized as offline decoding. Researchers present many algorithms to speed it up, such as pruning [11, 12], look-ahead [13, 14, 15] and on-the-fly hypothesis re-scoring idea under *phone-pair assumption* [16]. Unfortunately, it is still difficult to generate an exact lattice with a huge LM in a fast way, so it is worthy to explore faster approaches.

The paper proposes a novel method to optimize the on-the-fly composition decoding for exact lattice generation [8]. The proposed WFST-based decoder is denoted as *asynchronous dynamic decoder* (AsyncBigLM decoder). The core novel design of the proposed decoder is that it has two fronts, with one performing "exploration" and the other "backfill". An A* algorithm is employed to evaluate which tokens worth to be back-filled. By evaluating the proposed asynchronous dynamic decoder, we observe up to 20.17% relative speedup. This work is open-sourced under Kaldi [17]. It is a general-purpose decoder, which does not have any special requirements for AMs or LMs, and will be compatible with all released Kaldi recipes.

2. WFST-BASED DECODER

2.1. Basic Decoder

In Kaldi, the standard decoding graph being used is very close to [6], where the WFST decoding graph is

$$S \equiv HCLG = \min(\det(H \circ C \circ L \circ G)), \quad (1)$$

where H, C, L, G represent the Hidden Markov Model (HMM) structure, phonetic context-dependency, lexicon and grammar re-

Lei Xie is the corresponding author. The codes associated with this work can be found from https://github.com/LvHang/kaldi/tree/async-a-star-decoder

spectively, and $\circ$ represents 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. A special symbol ϵ may occur on both input and output labels, which means "no label is present".

When we want to decode an utterance of T frames, we use the *token passing* algorithm [18] on the HCLG graph. We denote the acoustic log-likelihood graph as U, which is generated by the acoustic model. The algorithm can be regarded as composing U with the *HCLG* graph. We denote the result of the composition as W, which is the searching graph of the utterance.

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

If an exact search is performed, then W would have approximately $T + 1$ times more states than HCLG. After the composition, we find the best path (i.e. the path which has the lowest cost) in the searching graph. In practice, because of the time and memory limitations, the beam pruning [19] is performed instead of an exact search:

$$P = \text{prune}(W, \alpha), \tag{3}$$

where P is the pruned graph and α is the beam value. During the pruning operation, we discard all paths that are not within the beam α compared to the best cost.

In fact, 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. We record it as (*frame-index*, *HCLG-state*) pair. For each token, we keep the information of acoustic cost, graph cost and extra cost which indicate the difference between the best path through current token and the absolute best path under the assumption that any currently active states at decoding front can eventually succeed. Acoustic cost and graph cost are stored separately so that re-scaling and rescoring with higher-order LM subsequently are convenient.

2.2. BigLM Decoder

The basic idea of the on-the-fly composition decoder [11], denoted as *BigLM decoder*, is to create the decoding graph HCLG with a small LM, and compose it with a WFST representing the difference between a large LM and the small LM dynamically. Imagine that the small LM is G, and the large one is G'. The decoding graph can be regarded as a two-stage composition:

$$F = -G \circ G', \tag{4}$$

$$S_{big} = HCLG \circ F, \tag{5}$$

where $-G$ has the same topology as G but with its weights negated. We refer to F as *residual grammar*. In practice, we keep the *HCLG* and F separately. Compared with the basic decoder, we keep the 3-tuple (*frame-index*, *HCLG-state*, *F-state*) for each token. At each time step, the token passing executes the following steps:

1. Get the *HCLG-state* of a token, pass it on for one step in the *HCLG* graph, record the output label on that arc and obtain a new *HCLG-state'*.
2. Get the *LM-state* of the token, regard the output label as input label and pass it on for one step in the residual grammar (i.e. F). Obtain a new *LM-state'*.
3. Generate the new 3-tuple with the new *HCLG-state'* and *LM-state'*. The frame index depends on the input label (i.e.ϵ or not) in the *HCLG* graph.

The BigLM decoder is memory-efficient so that richer LM knowledge can be involved in one-pass decoding. Compared with the lattice re-scoring method with the same beam, the BigLM decoder usually gives better accuracy. The benefit comes from better pruning, where the information of the large LM is included, and the Viterbi beam pruning is done with closer-to-optimal language model probabilities. But the BigLM decoder is slow due to the computational overhead introduced by composition during decoding.

3. IMPROVED ASYNCHRONOUS BIGLM DECODER

3.1. Motivation

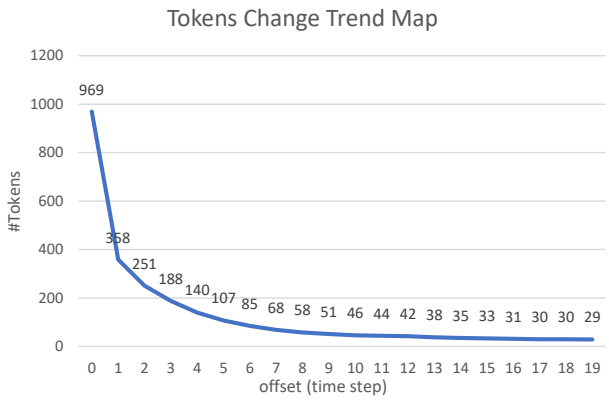

Fig. 1. The trend map of tokens change. It is an illustration in the case of a WFST-based decoder. The number of tokens decreases exponentially and levels off gradually.

The big-picture of the proposed work is as follows: periodically, the decoder will treat tokens on the current frame as final, and prunes the state-level lattice to the lattice-beam – discarding tokens whose costs are worse than the best one by a margin. A lot of tokens will be discarded so that the previous generation operations associated with these discarded tokens are wasted during decoding. These operations can be saved. This can be illustrated in Figure 1, where we report the number of tokens still alive at frame (t − offset) [1], on a lattice generated from a randomly picked example from the Librispeech dataset [20]. As the figure indicates, the number of active tokens reduces rapidly when future information is used during pruning.

3.2. AsyncBigLM decoder

Firstly, let's re-visit the A* method briefly.

$$H^*(s) = f(s) + g^*(s), \tag{6}$$

where $H^*(s)$ is the estimated score of the best complete path through state s. $f(s)$ is the score from the beginning to the state s in the partial path, which can be obtained by accumulating the acoustic and LM probabilities during decoding straight-forwardly. $g^*(s)$ is an estimate of the best partial path from state s to the end. The key to the A* method is how to estimate a reasonable $g^*(s)$.

Our proposed AsyncBigLM decoder is similar to the BigLM decoder in which its searching space is constructed in (*frame-index*, *HCLG-state*, *F-state*) space. Nevertheless, it works in a different way where it has two "decoding fronts", namely "exploration" front

[1] offset = $0 \ldots T_1$

and "backfill" front respectively. The former one occurs at the current frame t and the latter one at frame t − offset [2]. Basically, we process the "best-in-class" token, which has the best cost for each specific HCLG-state s on the "exploration" front. The "not-best-in-class" tokens will be processed on the "backfill" front. We deal with the two fronts alternately, so the sequence will be something like: explore for frame t, backfill for frame (t − offset), explore for frame ($t+1$), backfill for frame ($t+1-$offset), ... and so on [3]. The details of these two fronts are described in the following two sections.

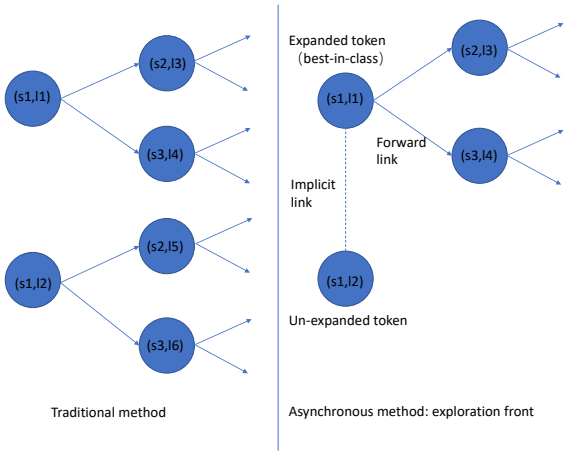

Fig. 2. The difference between traditional biglm decoding and improved biglm decoding. In the traditional case, each state pair is expanded in current frame. But in the improved case, only the best token for each HCLG state is expanded.

3.2.1. *Exploration Front*

We show the difference between BigLM decoder and the AsyncBigLM decoder in the forward pass in Fig 2. Let's write the state-pair (*HCLG-state*, *F-state*) on each frame as (s, l) for short. On the exploration front at frame t, the operations would be similar to the current BigLM decoder, with one exception – suppose there are some tokens having the same *HCLG-state* and different *F-state*s on the current frame (e.g. the (s1, l1) and (s1, l2) tokens in Fig. 2). In the BigLM decoder, all arcs leaving each state should be processed, but in the AsyncBigLM decoder, only the "best-in-class" token, which has the best forward cost (the α cost in the HMM sense) in all tokens with the same *HCLG-state* on the frame, will be expanded on the exploration front (e.g. Token (s1, l1) in Fig. 2). It is also called as "expanded" token. Then the "not-best-in-class" tokens, denoted as "un-expanded tokens", will not be expanded immediately (e.g. Token (s1, l2) in Fig. 2). Instead we create *implicit links* from them to the expanded token. We will consider processing all the un-expanded tokens on the backfill front. In conclusion, the procedure above would suppress the propagation of all but the "best-in-class" token for each *HCLG-state*.

3.2.2. *Backfill Front*

On the backfill front, we expand the previous un-expanded tokens. Firstly, we need to decide which un-expanded tokens should be processed with the help of the A* method. We treat the expanded tokens on the exploration front as final and compute the backward cost (the β cost in the HMM sense) of all the expanded tokens from the exploration front to the backfill front. Then we assume that the $g^*(s)$ of the un-expanded tokens are the same as the $g^*(s)$ of the expanded tokens that these un-expanded tokens link to (e.g. Token (s1, l2) can borrow the $g^*(s)$ of Token (s1, l1) in Fig. 2). So we can obtain $H^*(s)$ of each un-expanded token with from $f(s)$ and the $g^*(s)$ as shown in Eq.(6). Then $H^*(s)$ of each un-expanded token is compared with that of the best token to decide whether the token should be expanded on the backfill front.

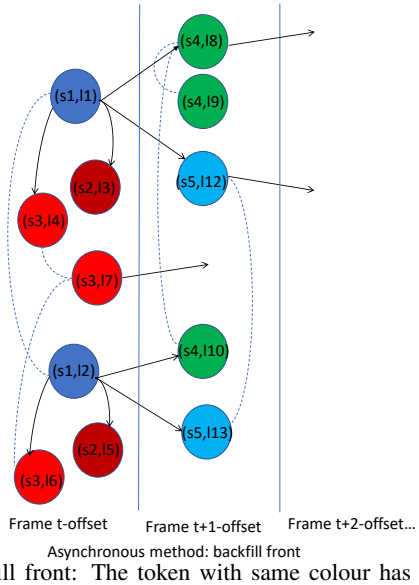

Fig. 3. Backfill front: The token with same colour has the same *HCLG-state* but different *F-states*. The "un-expanded" token follows the "expanded" token to expand itself. (The blue dotted lines represent the *implicit links* and the black arrow lines represent the *forward links*.)

When we expand an previous un-expanded token, we follow the footsteps of the arcs of the corresponding expanded token, using only the information present in the *forward links* leaving the expanded token, i.e., not revisiting the graph and the acoustic likelihoods. When we create new tokens on the destination-states, the implicit links of them are set. These implicit links will be used to expand these destination tokens appropriately when we do back-fill in the next time. The situation is illustrated in Fig 3. For example, the un-expanded token (s1,l2) is expanded by following the expanded token (s1,l1)'s footsteps and the implicit links of all destination tokens are set.

In addition, when processing these previously un-expanded tokens, we need to pass the information from the backfill front to the exploration front immediately in two special cases:
1) On the backfill front, when we expand a token and its destination token reaches an "existing state" (one that was created during a prior exploration step, for instance), and the destination token has a better forward cost than the token on existing state. In this case, before further exploration, we would propagate the cost change along all the paths starting from the existing state, so that in the future exploration we can decode with up-to-date forward-costs. Otherwise the exploration forward-cost would be permanently "out-of-step".
2) A new token is created on state (s, l), and it has better cost than the existing tokens on the same *HCLG-state* s. In this case, before the further "exploration front" is performed, we expand the new token till the current frame – this guarantees we process the correct "best-in-class" tokens on the exploration front.

[2] "offset" is set up empirically.

[3] The process on each front can be executed for a few frames to balance the accuracy and speed. E.g. explore for frame $t, t+1$; backfill for frame (t − offset), ($t+1-$offset); explore for frame $t+2, t+3$; backfill for frame ($t+2-$offset), ($t+3-$offset); ... and so on.

4. EXPERIMENT

We use the open-source speech recognition toolkit Kaldi [17] to conduct the experiments. We evaluate the algorithm with the corpus–LibriSpeech [20] (LIB) which contains about 960 hours training data and 4 kinds of separate test data-sets (dev-clean, dev-other, test-clean and test-other). Each of them has about 2 hours audio data.

All the acoustic models are trained with TDNN [21] structure and lattice-free maximum mutual information [22] (LF-MMI) criterion. Besides our improved AsyncBigLM decoder, we employ two methods as baselines. The first one is "lattice-rescoring" method which generates the lattices with a small LM and rescoring them with a large LM. The second one is the BigLM decoder which is described in Section 2.2. For the LibriSpeech testing, the customary standard LMs are used [20]. The small 3-gram LM (60MB) is employed to build the HCLG graph. The mild-pruned 3-gram LM (tgmed, 140MB), original 3-gram LM (tglarge, 760MB) and original 4-gram LM (fglarge) are used to build *residual grammar* separately.

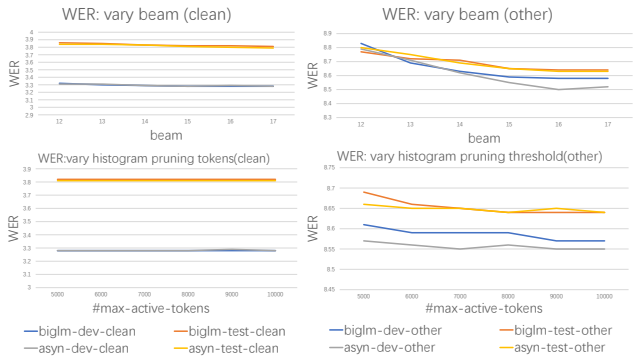

Fig. 4. The WER between BigLM and AsyncBigLM. It shows the performance of the two kinds of decoders are similar

In the following, unless otherwise specified, we show the results with the histogram pruning threshold (maximum-active-tokens=7000), beam pruning (beam=15) and lattice pruning (lattice-beam=8) for space reason. As shown in Fig 4 and Fig 5, we tried to tune the hyper-parameters and the trend of results is similar.

4.1. Accuracy

Table 1. WER statistics: Rescoring/BigLM/AsyncBigLM

	tgmed	tglarge	fglarge
dev-clean	4.27/4.25/4.24	3.38/3.38/3.38	3.27/3.28/3.28
dev-other	11/11/11.04	9.14/9.1/9.1	8.7/8.59/8.55
test-clean	4.74/4.77/4.77	3.94/3.93/3.92	3.83/3.82/3.81
test-other	11.2/11.18/11.18	9.21/9.17/9.15	8.72/8.65/8.65

In speech recognition task, the most straight-forward evaluation criterion is word error rate (WER). In Table 1, we compare the WER among "Lattice-rescoring", BigLM decoder and our proposed AsyncBigLM decoder. It shows that WERs of the three methods are close, but the last two methods are better than the first a little bit. The benefit comes from the fact that the information of the large LM is included when the decoder goes across the word boundary. Compared with WERs, average log-likelihoods of lattices will provide more accurate information for decoder evaluation. Thus in Table 2, we compare the average log-likelihood between BigLM decoder and AsyncBigLM decoder. As "lattice-rescoring" is generated from the small graph, the average log-likelihood is far behind. From Table 2, we can see that the differences are extremely small (< 0.0001), which means the accuracy of the two decoders is similar under the same condition. As the "BigLM"-kind decoders are better, we compare them further. In Figure 4, we show the results when we tune

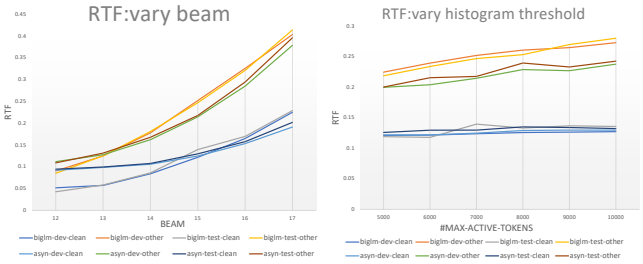

Fig. 5. The RTF between BigLM and AsyncBigLM. The latter one is better than the former when the searching graph is large the decoding beam/histogram pruning threshold. We can see that the WERs are comparable.

Table 2. Log-likelihood statistics: BigLM/AsyncBigLM

	tgmed	tglarge	fglarge
dev-clean	3.80056/3.80054	3.83091/3.83085	3.84251/3.84245
dev-other	3.32441/3.32439	3.34851/3.34839	3.3593/3.35919
test-clean	3.73661/3.73661	3.76555/3.76551	3.77599/3.77594
test-other	3.2984/3.2984	3.32344/3.32342	3.33388/3.33387

4.2. Speedup

Table 3. RTF: Biglm/AsyncBiglm

	tgmed	tglarge	fglarge
dev-clean	0.1178 / 0.1056	0.1158 / 0.1141	0.1216 / 0.1233
dev-other	0.1998 / 0.1877	0.2165 / 0.2061	**0.2517 / 0.2094**
test-clean	0.1019 / 0.1106	0.1242 / 0.1188	**0.1394 / 0.1294**
test-other	0.2075 / 0.1937	**0.2366 / 0.2017**	**0.2466 / 0.2172**

Under the same pruning parameters condition, we compare the real time factor (RTF) between BigLM and AsyncBigLM to show the speedup performance in Table 3. The bold figures in Table 3 show the speedup rate is about 7.6 – 20.17%. While as shown in Figure 5, the RTFs of AsyncBigLM are better than BigLM gradually as we increase the decoding beam or histogram pruning threshold. Table 3 also reflects the trend that as the increase of data complexity (i.e. noisy data or bigger LM), the speedup improvement is more obvious. We believe the reasons are that: 1) For noisy data, it might lead to less discriminative likelihoods from the acoustic model so that more hypotheses with similar scores need to be processed in BigLM decoder. 2) For the bigger LM, it will lead to a bigger residual grammar space. However, as the AsyncBigLM only propagates the best-in-class tokens on the exploration front and skip unpromising tokens on the backfill front, it apparently saves lots of operations from the two aspects so that the speed increase more obvious.

As the expectation in Section 4.1, the AsyncBigLM decoder achieves the benefit from saving searching operations. We count the effective propagation times on both BigLM decoder and AsyncBigLM decoder. The propagation statistics comparison further shows the reasons of acceleration: 1) the propagation times of asynchronous decoder is significantly less than that of the BigLM decoder on the exploration front; 2) the propagation times on the backfill front is limited and the total times of AsyncBigLM decoder is still less than the BigLM decoder's. They prove the validity of our proposed asynchronous method.

Table 4. The effective propagation times (million)

	Exploration	Backfill
BigLM	5.97	0
AsyncBigLM	3.84	0.3

5. CONCLUSION

In this paper, we proposed a smart AsyncBigLM decoder with A* method to speedup the one-pass on-the-fly composition decoding. The proposed algorithm can achieve up to 20.17% speedup rate. More importantly, the speedup would be more prominent and stable as the complexity of data increases.

6. REFERENCES

[1] Vijayaditya Peddinti, Yiming Wang, Daniel Povey, and Sanjeev Khudanpur, "Low latency acoustic modeling using temporal convolution and lstms," *IEEE Signal Processing Letters*, vol. 25, no. 3, pp. 373–377, 2017.

[2] Golan Pundak and Tara Sainath, "Lower frame rate neural network acoustic models," 2016.

[3] Ian McGraw, Rohit Prabhavalkar, Raziel Alvarez, Montse Gonzalez Arenas, Kanishka Rao, David Rybach, Ouais Alsharif, Haşim Sak, Alexander Gruenstein, Françoise Beaufays, et al., "Personalized speech recognition on mobile devices," in *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2016, pp. 5955–5959.

[4] Xu Xiang, Yanmin Qian, and Kai Yu, "Binary deep neural networks for speech recognition.," in *INTERSPEECH*, 2017, pp. 533–537.

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

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

[7] Andrej Ljolje, Fernando Pereira, and Michael Riley, "Efficient general lattice generation and rescoring," in *Sixth European Conference on Speech Communication and Technology*, 1999.

[8] Daniel Povey, Mirko Hannemann, Gilles Boulianne, Lukáš Burget, Arnab Ghoshal, Miloš Janda, Martin Karafiát, Stefan Kombrink, Petr Motlíček, Yanmin 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.

[9] Andreas Stolcke, Yochai Konig, and Mitchel Weintraub, "Explicit word error minimization in n-best list rescoring," in *Fifth European Conference on Speech Communication and Technology*, 1997.

[10] Hainan Xu, Tongfei Chen, Dongji Gao, Yiming Wang, Ke Li, Nagendra Goel, Yishay Carmiel, Daniel Povey, and Sanjeev 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.

[11] Takaaki Hori, Chiori Hori, and Yasuhiro 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.

[12] David Nolden, Ralf Schlüter, and Hermann Ney, "Search space pruning based on anticipated path recombination in lvcsr," in *Thirteenth Annual Conference of the International Speech Communication Association*, 2012.

[13] Hagen Soltau, Florian Metze, Christian Fügen, and Alex Waibel, "Efficient language model lookahead through polymorphic linguistic context assignment," in *2002 IEEE International Conference on Acoustics, Speech, and Signal Processing*. IEEE, 2002, vol. 1, pp. I–709.

[14] Hagen Soltau and George Saon, "Dynamic network decoding revisited," in *2009 IEEE Workshop on Automatic Speech Recognition & Understanding*. IEEE, 2009, pp. 276–281.

[15] Jung-Gi Baek, Sang-Hun Yoon, and Jong-Wha Chong, "Memory efficient pipelined viterbi decoder with look-ahead trace back," in *ICECS 2001. 8th IEEE International Conference on Electronics, Circuits and Systems (Cat. No. 01EX483)*. IEEE, 2001, vol. 2, pp. 769–772.

[16] Takaaki Hori, Yoshiaki Noda, and Shoichi Matsunaga, "Improved phoneme-history-dependent search method for large-vocabulary continuous-speech recognition," *IEICE TRANSACTIONS on Information and Systems*, vol. 86, no. 6, pp. 1059–1067, 2003.

[17] Daniel Povey, Arnab Ghoshal, Gilles Boulianne, Lukas Burget, Ondrej Glembek, Nagendra Goel, Mirko Hannemann, Petr Motlicek, Yanmin Qian, Petr Schwarz, et al., "The kaldi speech recognition toolkit," in *IEEE 2011 workshop on automatic speech recognition and understanding*. IEEE Signal Processing Society, 2011, number CONF.

[18] Stephen John Young, NH Russell, and JHS Thornton, *Token passing: a simple conceptual model for connected speech recognition systems*, Citeseer, 1989.

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

[20] Vassil Panayotov, Guoguo Chen, Daniel Povey, and Sanjeev 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.

[21] Vijayaditya Peddinti, Daniel Povey, and Sanjeev 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.

[22] Daniel Povey, Vijayaditya Peddinti, Daniel Galvez, Pegah Ghahremani, Vimal Manohar, Xingyu Na, Yiming Wang, and Sanjeev Khudanpur, "Purely sequence-trained neural networks for asr based on lattice-free mmi.," 2016.