THE TITECH LARGE VOCABULARY WFST SPEECH RECOGNITION SYSTEM

Paul R. Dixon[1], Diamantino A. Caseiro[2], Tasuku Oonishi[1], Sadaoki Furui[1]

[1]Department of Computer Science, Tokyo Institute of Technology, Japan
[2]INESC-ID Lisboa, Spoken Language Systems Lab, Portugal
{dixonp,oonishi,furui}@furui.cs.titech.ac.jp dcaseiro@l2f.inesc-id.pt

ABSTRACT

In this paper we present evaluations on the large vocabulary speech decoder we are currently developing at Tokyo Institute of Technology. Our goal is to build a fast, scalable, flexible decoder to operate on *weighted finite state transducer* (WFST) search spaces. Even though the development of the decoder is still in its infancy we have already implemented a impressive feature set and are achieving good accuracy and speed on a large vocabulary spontaneous speech task. We have developed a technique to allow parts of the decoder to be run on the graphics processor, this can lead to a very significant speed up.

Index Terms— Speech recognition, Parallel programming

1. INTRODUCTION

We are currently developing a large vocabulary speech recognition system at Tokyo Institute of Technology (Titech), our goal is to build a decoder to operate on *weighted finite state transducer* (WFST) search spaces. The main design criteria are flexibility, speed and resource usage.

The paper is structured as follows: Section 2 of the paper is a very brief overview of WFST speech decoders. Section 3 first covers the basic operation of the search algorithm, then describes the decoder features in more detail and is loosely broken down according to our design criteria. To address speed we describe the parallelisation extensions that exploit multiprocessor systems. We describe a novel and unique feature of our decoder where we have extended the multi-processor paradigm to allow the acoustic computations to be offloaded onto the graphics card. This is important work, which shows how another type of application can make use of the graphics processing power already available in many machines.

There is a description of flexibility options available in the decoder, these include an overview of the highly modular frontend which allows the system to run in batch mode or streaming mode accompanied by details of the live decoding extensions and lattice generation features.

We next describe how we have reduced the memory usage by incorporating a disk based WFST option which also allows the system to fast-load the models. Section 4 of the paper provides a set of benchmarking results on a difficult Japanese spontaneous speech task. The experiments aim to illustrate the decoder's behavior in terms of accuracy, speed and memory usage when operating in various configurations.

2. BACKGROUND

The basic speech recognition problem can be expressed as:

$$\hat{\mathbf{W}} = \arg\max_{\mathbf{W}} \{P(\mathbf{X} \mid \mathbf{W})P(\mathbf{W})\} \quad (1)$$

The task of the decoder is to find the most likely word sequence $\hat{\mathbf{W}}$, given an observed sequence of speech feature vectors $\mathbf{X}$. The language model probability is denoted by $P(\mathbf{W})$, whilst $P(\mathbf{X} \mid \mathbf{W})$ is the score from the other knowledge sources such as the acoustic models.

Within the WFST paradigm all the knowledge sources in the search space are combined together to form a static search network [1]. The composition often happens off-line before decoding and there exist powerful operations to manipulate and optimise the search networks [1]. Within this framework the decoder becomes agnostic to various knowledge sources, thus changes made to the information sources before the network compilation stage should not require modifications to the decoder.

However there is a caveat with the WFST approach. The fully composed networks can often be very large and therefore at both composition and decode time large amounts of memory can be required. To mitigate the memory burden on the decoder approaches such as *on-the-fly* composition of the network [2, 3], disk based search networks [4] and efficient in-memory representations [5] have been developed by others.

3. DECODER DESCRIPTION

Our current implementation is a single pass time synchronous Viterbi beam search decoder using a token passing scheme [6]. The decoder has been designed specifically with speech tasks in mind. We are primarily considering a recogniser cascade which performs a transduction from context dependent phone arcs to word sequences, that is $C \circ L \circ G$ (CLevel)

where C is the context dependency, L is the lexicon, G is the language model and $\circ$ denotes the composition operator. During the search the decoder dynamically expands the hidden Markov model (HMM) arcs into state sequences. It is also possible to decode state level networks by composing acoustic models H into the recogniser cascade $H \circ C \circ L \circ G$ (HLevel), and then simulating the self-transitions via an appropriate arc definition that contains a single HMM state.

The decoder's search algorithm keeps a list of active search states and a list of active arcs. States and arcs are considered active when they contain tokens. The algorithm consists of creating a token for the empty hypotheses, associating it with the initial state of the search network and then iterating the three following steps for each parameter vector in the utterance.

1. In the first step, expand active states, tokens are propagated from each state to the initial state of every arc leaving the state.

2. In the second step, expand active arcs, the tokens held in each arc are advanced to the next frame. This step uses a specialised time synchronous Viterbi algorithm optimised to the arc topology. When a token reaches the last state in the arc, it is propagated to the following search state, and activating it if necessary.

3. Finally, in the third step, epsilon propagation, tokens are propagated across epsilon input edges.

The best solution is recovered as the best hypothesis contained in a final state after processing the last frame of the utterance.

Two pruning strategies are currently used: *Beam* pruning where at each frame hypotheses are culled if they have a score greater than the combined current best cost plus beam width. *Histogram* pruning caps the maximum active hypotheses allowed to a fixed band width.

As a large amount of time is spent performing the acoustic computations, the Gaussian mixture calculations are written to take advantage of the Streaming Single Instruction Multiple Data (SIMD) extensions when they are available.

3.1. Multiprocessing

It is common for commodity processors to support some form of multiprocessing, either via multiple cores or as multi threads. To fully exploit these architectures, our decoder can also divide the work into various threads, using an approach similar to [7]. Each thread has its own active state and arc lists, and its own memory management. States are assigned to a particular thread using a hash function of its numerical ID and arcs are assigned to the same thread as its source state. Interaction between threads occurs when a token is propagated from the last state of an arc to the following search state, because this state can belong to a different thread. A pending activation list is used to delay this propagation so that it will be later done by the thread that owns the state. Synchronisation between threads is achieved using barriers at various points in the algorithm: before propagating pending lists, and during epsilon propagation. Epsilon propagation is still single threaded.

3.2. Acoustic Computation using Graphics Hardware

Acoustic computations often consume the greatest amount of CPU time during speech decoding, others have proposed algorithmic methods for reducing this cost [8, 9]. We are currently investigating new techniques to utilise the graphics processor unit (GPU) as a co-processor for offloading the acoustic calculations. Modern GPUs are massively parallel processors with very large memory bandwidth. The field of research concerned with performing general purpose computations on graphics hardware [10] is known as General Purpose GPU (GPGPU). In our GPGPU scheme all of the acoustic models are represented as a matrix $\mathbf{A}$ in which each row is a log weighted Gaussian component. The i^{th} component of J dimensional mixture model is represented as a vector according to:

$$\left\{ K, \frac{\mu_{i1}}{\sigma_{i1}^2}, \ldots, \frac{\mu_{ij}}{\sigma_{ij}^2}, -\frac{1}{2\sigma_{i1}^2}, \ldots, -\frac{1}{2\sigma_{ij}^2} \right\} \tag{2}$$

Where K is:

$$\log w_i - \frac{J}{2}\log 2\pi - \frac{1}{2}\sum_j \log \sigma_{ij}^2 - \frac{1}{2}\sum_j \frac{\mu_{ij}^2}{\sigma_{ij}^2} \tag{3}$$

The score of every Gaussian can be calculated simultaneously by expanding the feature vector $\mathbf{x}$ to:

$$\mathbf{z}^T = \left\{ 1, \mathbf{x}_1, \ldots, \mathbf{x}_J, \mathbf{x}_1^2, \ldots, \mathbf{x}_J^2 \right\} \tag{4}$$

and performing the matrix vector multiplication $\mathbf{y} = \mathbf{A}\mathbf{z}$. $\mathbf{y}$ is vector containing log weighted scores of the feature vector for every Gaussian component. When the decoder is started the acoustic models are expanded and moved into video memory. Next in the main decoding loop before searching each frame the feature vector is sent to the graphics card where every Gaussian score is computed. The scores are moved back from the graphics card into acoustic model score cache, where the scores are then used during the search for the current frame.

3.3. Disk Based Network

The decoder can also operate with the search network left on the disk. The other model components including the acoustic models, arcs definitions and word lists are typically much smaller than the fully composed WFST and are therefore loaded into memory.

In order to perform disk based decoding the WFST file is composed of three separate blocks and arranged in the following manner. A *header* contains information on the size of the network such as the state and arc count. A *state offset*

dictionary, a record for each state indicates the arc count and the position on the file where the arcs are located. The *arcs* for each state are written as blocks in a sequential manner.

During search when a set of arcs is required, an in-memory copy of the state dictionary is used to quickly find the file position. The arcs are then loaded from file and used for the search. Under this scheme it is also necessary for the decoder to explicitly release arcs that are no longer needed to avoid memory leaks. A further benefit of the disk based approach is that instances of the decoder can be started very quickly.

3.4. Lattice Generation

The decoder also has the ability to additionally output lattices as well as a single best path. There is support for both word and phone lattices, these are both written out as a WFST. Lattice generation is based on the phone pair assumption [11].

3.5. Frontend Description

The decoder has a highly customizable frontend. The design was influenced by the design of the Sphinx4 [12] frontend. Chains of processing blocks are connected together and data packets propagated through to the decoding engine, each one of the processors is built to perform a specific operation such as windowing or a Fourier transform. The frontend was designed with the following goals in mind:

- To operate in streaming or batch modes.
- To enable arbitrarily long chains of processors to be configured in any order at runtime.
- To facilitate ease of extension in which new types of processor blocks can be written and seamlessly integrated into the frontend.

The current implementation includes all the required processors to extract MFCC features with delta, delta-delta and energy terms.

3.6. Live Decoding

The streaming frontend also allows for the decoder to operate in a live mode. The stream could be from an arbitrarily long data source such as a news feed or a microphone or simply to allow the decoder to operate on large files whilst consuming less memory. In the live mode the decoder operates slightly differently, after garbage collection if a common prefix is found in the traceback the partial hypothesis up to that point is output and the associated memory is released. To allow the decoder to run indefinitely the WFST network is closed so that the final state can feed back into the initial state. This is similar to the scheme described in [13]. The cepstral mean normalisation (CMN) in the live frontend uses a fixed mean vector calculated from the training data.

4. EVALUATIONS

Our evaluations were carried out using the Corpus of Spontaneous Japanese (CSJ) [14]. The corpus contains a total of 228 hours of training data from 953 lectures.

The raw speech was first converted to a sequence of 38 dimensional feature vectors with a 10ms frame rate and 25ms window size. Each feature vector was composed of 12 Mel-frequency cepstral coefficients with delta and delta–deltas, augmented with delta and delta–delta energy terms.

The acoustic models were three state left-to-right tri-phone models. The state output densities were 16 component Gaussian mixture models with diagonal covariances. The language model was back-off trigram with a vocabulary of 25k words.

The test set used for evaluations was composed of 2328 utterances which spanned 10 lectures. This yielded a total of 116 minutes of speech. On this testing data the language model perplexity was 57.8 and the out of vocabulary rate was 0.75%.

The experiments were conducted on a 2.40GHz Intel Core2 machines with 2GB of memory and an Nvidia 8800GTX graphics processor running Linux, the decoder was compiled using the GCC compiler. All of the evaluations utilised the decoder operating without multiprocessor support with the exception of that described in section 4.2. In this evaluation the multi-threaded decoder was compared to the decoder operating without threading support.

4.1. HLevel and CLevel WFST Evaluations

We first evaluated our decoder operating on both CLevel and HLevel networks and compared the performance to the Julius decoder [15] (`http://julius.sourceforge.jp/`).

Several recognition experiments were run with the beam width varied from 100 to 200, the maximum number of hypotheses allowed at any one time during the decoding was capped to the best 10000. Figure 1 shows the recognition accuracy and its relationship to the real time factor (RTF). The WFST decoder was consistently able to achieve higher accuracy for the same RTF when compared with Julius (version 3.5.3).

At the CLevel the compiled network had approximately 2.1M states and 4.3M arcs, and the decoder required between approximately 150MBs and 170MBs of memory during search. As expected the HLevel network had substantially more states and arcs at approximately 6.2M and 7.7M respectively, this translated to between 330MBs $\sim$ 400MBs of memory usage in the decoder. Julius for comparison required 60MBs $\sim$ 100MBs of memory.

For narrow beams the HLevel decoder was slightly faster and achieved a marginally higher accuracy, showing the better optimized HLevel networks can be used with small overhead using singleton arcs. The decoder operating on the CLevel network required substantially less memory than when using

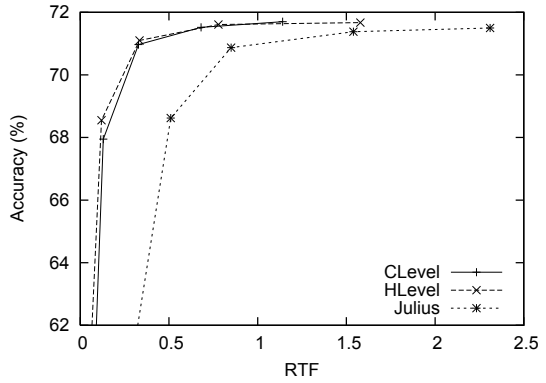

Fig. 1. Recognition accuracy versus RTF comparing the WFST decoder using CLevel and HLevel networks with the Julius decoder.

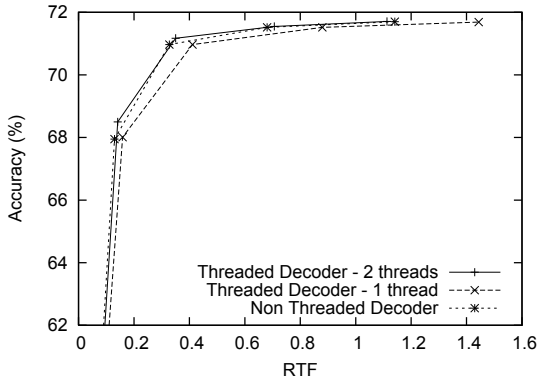

Fig. 2. Recognition Accuracy versus RTF comparing the multi-threaded decoder using 1 and 2 cores to the non-threaded decoder.

the HLevel equivalent. However, using the HLevel network has the advantage of being able to represent totally arbitrary networks.

4.2. Multiprocessor Evaluations

The decoder was additionally run in multi-threaded mode using one and two threads to take advantages of both of the cores in the processor. Figure 2 shows the performance of the multi-threaded decoder using one or two threads in comparison to the non-threaded decoder. The multi-threaded decoder with two threads was faster than running in single threaded mode. However, when decoding in parallel mode neither core fully saturated. Profiling indicated this is due to synchronisation points where the threads communicate best costs and epsilon propagation. The multi-threaded decoder using two threads is able to achieve higher accuracy for the same beam when compared to a single-threaded decoder. This is because there are parts of the decoding where each thread uses a local best cost for pruning and not the absolute best cost at that point in time. Therefore hypotheses which would normally be pruned in the single threaded systems are allowed to exist for longer.

4.3. GPGPU Evaluations

In this set of evaluations the GPGPU enabled decoder is compared to the SSE vectorized Gaussian scheme. The RTF increase in Table 1 shows that for wide beam we see a speed increase; using the GPU for a beam of 175 results in over a 25% reduction in RTF. However, for narrower beams the GPGPU decoder runs slower. This is as expected because for narrow beams the average number of Gaussians needed per frame is small even though with GPGPU we are calculated every Gaussian and communicating with the graphics card over a slower bus.

Beam	SSE RTF	GPGPU RTF	RTF Reduction (%)
100	0.04	0.1	-150
125	0.13	0.14	-7.69
150	0.32	0.25	21.88
175	0.67	0.5	25.37
200	1.12	0.88	21.43

Table 1. RTF reduction when performing acoustic computations using the graphics card.

4.4. Disk Based Network Evaluations

In this round of evaluations we consider the speed impact and memory reduction by switching to a disk based WFST. The evaluations were run using the non-threaded decoder operating in disk and memory based WFST modes. Figure 3 shows the accuracy versus RTF and Figure 4 shows how the memory usage varies with beam width. Memory measurements were approximations based on the *resident* memory reading from the *top* command. The results show that by using a disk based WFST it is possible to substantially reduce memory requirements whilst only increasing the RTF by a small amount.

A further caching scheme was also implemented in which once a set of arcs had been read from the disk they were held in an in-memory cache permanently. However, our experiments showed only a negligible increase in speed, possibly because the operating system or hard disk is already caching frequently accessed data.

4.5. Lattice Generation Evaluations

In these evaluations we study the behavior of the decoder when generating phone lattices. The motivation behind these experiments was to quantify how the extra load on the decoder would affect the decoding speed. These experiments represent the slowest case for measuring the additional load for generating lattices. This is because we not only consider

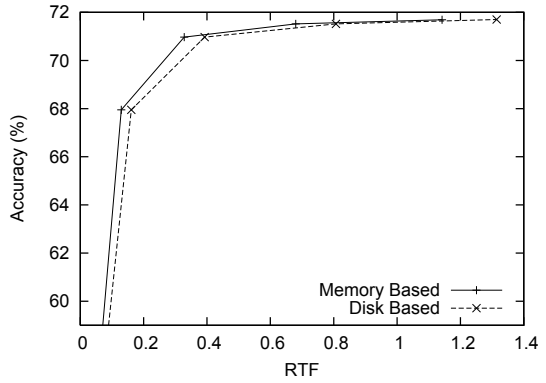

Fig. 3. Recognition Accuracy versus RTF for the memory and disk systems operating using the non-threaded decoder.

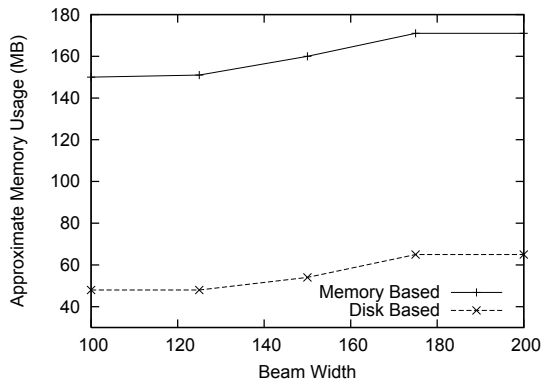

Fig. 4. Memory usage versus beam width for the non-threaded decoder using the memory and disk based networks.

the additional overhead the decoder has in generating the lattices but also include the time required to write the lattices out to disk. The evaluations used the non-threaded decoder configured to generate phone level lattices. Figure 5 shows there is a small reduction in speed when generating lattices.

4.6. Frontend Evaluations

In these evaluations we consider the performance of the recogniser when carrying out feature extraction inside the decoder. In these experiments the frontend was operated in a batch-like mode where for each utterance all of the frames are computed before applying CMN. The feature vectors are then streamed into the decoder. The baseline system reads pre-computed binary cepstra files from disk and loading times are included in the baseline system measurements.

As expected there was no reduction in accuracy when decoding from raw speech. Table 2 shows there is an RTF increase when using the frontend. For moderate to large beams the frontend accounts for only a small portion of the total CPU time.

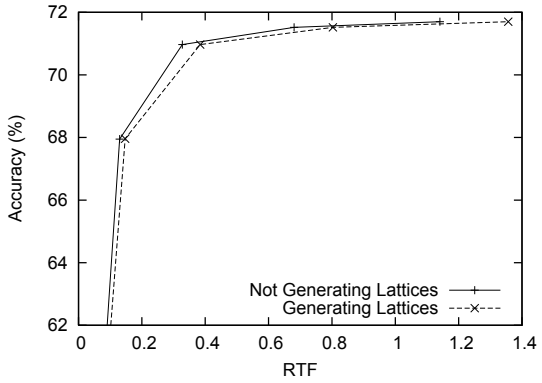

Fig. 5. Performance of the decoder with and without phone lattice generation enabled.

Beam	RTF Increase (%)
100	17.23
125	6.78
150	2.74
175	1.15
200	0.649

Table 2. RTF increase when performing feature extraction.

4.7. Live Decoding Evaluations

When operating in the live mode the decoder can stream data via the frontend and output text during the decoding process. In these evaluations we consider the decoders performance by transcribing entire unsegmented lectures from the CSJ test set. On arbitrarily long data and large files a static mean value computed from the training data is used for CMN. Figure 6 shows the performance of the live decoder operating on lectures compared to the batch decoder recognising cepstra files. For the same beam width the live decoder performs slightly worse than the batch utterance decoder. Applying dithering to batch CMN utterances did not reduce the accuracy. Applying batch CMN across an entire lecture and then live decoding gave nearly identical maximum accuracy to the baseline system referred to in Figure 6. Furthermore, applying static CMN to the utterance data also lead to around a 1% drop in maximum accuracy. These results indicate mismatches in the CMN conditions for testing and training are slightly reducing the accuracy for the live lecture decoder.

5. CONCLUSIONS AND FUTURE WORK

In this paper we have presented preliminary evaluations of our new large vocabulary speech decoder. We described the performance in various configurations.

We have described a new technique for performing the acoustic calculations using the graphics processor that can yield over a 25% reduction in the RTF. By using an improved

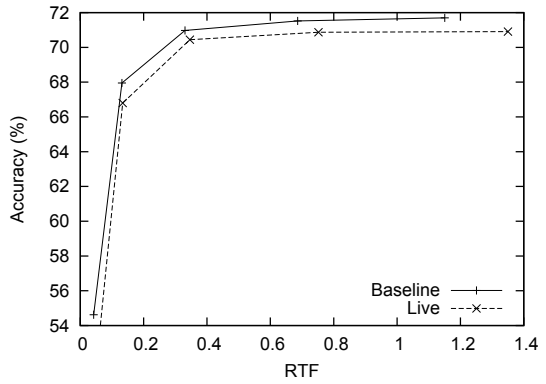

Fig. 6. Performance of the decoder operating in live mode.

GPGPU scheme it should be possible to make better use of computational power available. For example a multi-threaded decoder could use one core to perform the search and another core to co-ordinate the GPU to compute the acoustic scores for future frames.

Future work will finalize and evaluate our on-the-fly composition algorithms and continue improving the performance and speed of our decoder.

6. ACKNOWLEDGMENTS

The authors wish to thank Koji Iwano and Josef Novak for helpful comments on this manuscript. This work was supported by the Japanese government 21st century COE programme "Framework for Systematization and Application of Large-Scale Knowledge Resources" and the METI Project "Development of Fundamental Speech Recognition Technology". Diamantino Caseiro was partially funded by the Portuguese FCT project WFST (POSI/PLP/47175/2002) and INESC ID Lisboa has support from the POSI Program of the "Quadro Comunitário de Apoio III"'.

7. REFERENCES

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

[2] Takaaki Hori and Atsushi Nakamura, "Generalized fast on-the-fly composition algorithm for WFST-based speech recognition," in *Proc. of INTERSPEECH*, 2005, pp. 847–850.

[3] Diamantino A. Caseiro and Isabel Trancoso, "A specialized on-the-fly algorithm for lexicon and language model composition," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 14, no. 4, pp. 1281–1291, 2006.

[4] Daniel Willett, Erik McDermott, Yasuhiro Minami, and Shigeru Katagiri, "Time and memory efficient Viterbi decoding for LVCSR using a precompiled search network," in *Proc. of EUROSPEECH*, 2001, pp. 847–850.

[5] Diamantino A. Caseiro and Isabel Trancoso, "Using dynamic WFST composition for recognizing broadcast news," in *Proc. ICSLP*, 2002, pp. 1301–1304.

[6] S.J. Young, N.H. Russell, and J.H.S. Thornton, "Token passing: A simple conceptual model for connected speech recognition systems," Tech. Rep., Cambridge University Engineering Department, 1989.

[7] Steven Phillips and Anne Rogers, "Parallel speech recognition," *Int. J. Parallel Program.*, vol. 27, no. 4, pp. 257–288, 1999.

[8] George Saon, Geoffrey Zweig, Brian Kingsbury, Lidia Mangu, and Upendra Chaudhar, "An architecture for rapid decoding of large vocabulary conversational speech," in *Proc. of EUROSPEECH*, 2003, pp. 1977–1980.

[9] George Saon, Daniel Povey, and Geoffrey Zweig, "Anatomy of an extremely fast LVCSR decoder," in *Proc. of INTERSPEECH*, 2005, pp. 549–552.

[10] John D. Owens, David Luebke, Naga Govindaraju, Mark Harris, Jens Kru"ger, Aaron E. Lefohn, and Timothy J. Purcell, "A survey of general-purpose computation on graphics hardware," in *Eurographics 2005, State of the Art Reports*, aug 2005, pp. 21–51.

[11] Andrej Ljolje, Fernando Pereira, and Michael Riley, "Efficient general lattice generation and rescoring," in *Proc. of EUROSPEECH*, 1999, pp. 1251–1254.

[12] Paul Lamere, Philip Kwok, William Walker, Evandro Gouvea, Rita Singh, Bhiksha Raj, and Peter Wolf, "Design of the CMU sphinx–4 decoder," in *Proc. ICSLP*, 2003, pp. 1181–1184.

[13] Murat Saraclar, Michael Riley, Enrico Bocchieri, and Vincent Goffin, "Towards automatic closed captioning : Low latency real time broadcast news transcription," in *Proc. ICSLP*, 2002, pp. 1741–1744.

[14] Kikuo Maekawa, "Corpus of spontaneous Japanese: Its design and evaluation," in *Proc. ISCA and IEEE Workshop on Spontaneous Speech Processing and Recognition*, 2003, pp. 7–12.

[15] Akinobu Lee, Tatsuya Kawahara, and Kiyohiro Shikano, "Julius an open source real-time large vocabulary recognition engine," in *Proc. of EUROSPEECH*, 2001, pp. 1691–1694.