A study of large vocabulary speech recognition decoding using finite-state graphs[1]

Zhijian OU, Ji XIAO
Department of Electronic Engineering, Tsinghua University, Beijing
Corresponding email: ozj@tsinghua.edu.cn

***Abstract*—The use of weighted finite-state transducers (WFSTs) has become an attractive technique for building large vocabulary continuous speech recognition decoders. Conventionally, the compiled search network is represented as a standard WFST, which is then directly fed into a Viterbi decoder. In this work, we use the standard WFST representations and operations during compiling the search network. The compiled WFST is then equivalently converted to a new graphical representation, which we call finite-state graph (FSG). The resulting FSG is more tailored to Viterbi decoding for speech recognition and more compact in memory. This paper presents our effort to build a state-of-the-art WFST-based speech recognition system, which we call GrpDecoder. Benchmarking of GrpDecoder is carried out separately on two languages - English and Mandarin. The test results show that GrpDecoder which uses the new FSG representation in searching is superior to HTK's HDecode and IDIAP's Juicer for both languages, achieving lower error rates for a given recognition speed.**

Keywords—WFST; finite-state graph; grpdecoder

I. INTRODUCTION

Since the pioneering works at AT&T [1], the use of weighted finite-state transducers (WFST) has become an attractive technique for building large vocabulary continuous speech recognition (LVCSR) decoders. Basically, the task of decoding is to find the most likely state sequence in a search network constrained by various knowledge sources, such as language model, lexicon, phonetic context-dependency, and acoustic HMMs. Various decoding techniques mainly differ in two aspects - the search network expansion (static versus dynamic) and the search algorithm itself (time-synchronous versus asynchronous) [2]. The WFST approach has been shown to yield better performance when compared with traditional decoders using dynamic network expansion [3]. This is mainly attributed to (1) the separation of compiling the search network and doing the search itself, and (2) doing the compilation to fully optimize the search network to reduce its redundancy, through the powerful WFST operations of composition, determinization and minimization. The compilation, namely the static network expansion and optimization, is often executed offline before decoding. The final search network is represented as a single transducer that maps sequences of HMM states to sequences of words, which can be directly used in a Viterbi decoder.

As we know, running a WFST decoder consists of two stages, compiling the search network and performing the actual search. Previous works about the WFST approach are mainly concerned with the first stage. Mohri et al. [1] first introduces how to represent various knowledge sources as WFSTs and how to compile them through the standard WFST operations. Chen [4] proposes memory-efficient implementation of composition and determinization for using large-context phonetic decision trees. Recently, there are some studies that propose interaction between these two stages. Some algorithms that perform on-the-fly composition and optimization are studied [5][6]. These algorithms can be used either to reduce the memory burden during recognition, or to accommodate the use of dynamic knowledge sources.

This paper is concerned with the second stage. Conventionally, the compiled search network is represented as a standard WFST, which is then directly fed into a Viterbi decoder. In this work, we use the standard WFST representations and operations during compiling the search network. The compiled WFST is then equivalently converted to a new graphical representation, which we call finite-state graph (FSG). The resulting FSG is more tailored to Viterbi decoding for speech recognition and more compact in memory. The advantage of memory efficiency is due to the fact that the FSG representation exploits a special property of the final compiled WFST used in speech recognition, when compared to general WFSTs. After a background introduction to the standard WFST-based speech recognition in section II, we will detail the new FSG representation in section III.

In recent years, there have emerged a number of speech recognition systems that use the WFST approach, for example, those developed at AT&T [1], IBM [7], IDIAP (Juicer) [8], Titech (T^3) [6], etc. This paper presents our effort to build a state-of-the-art WFST-based speech recognition system, which we call GrpDecoder (as the abbreviation of graphical decoder). Benchmarking of GrpDecoder is carried out separately on two languages - English and Mandarin. The comparison systems are HDecode as part of HTK [9] (an excellent decoder using dynamic network expansion) and Juicer [8] (a standard WFST-based decoder), both of which are publicly available. The test results show that GrpDecoder which uses the new FSG representation in searching is superior to HDecode and Juicer for both languages, achieving lower error rates for a given recognition speed. The English and Mandarin benchmarking test are described in section IV.

[1] This work is supported by NSFC (61075020) and 863 (2006AA01Z149).

II. Review of the WFST approach to speech recogntion

In this section, we give a background introduction to the standard WFST-based speech recognition. For a more in depth description, the reader is referred to [1].

A. *Weighted finite-state transducers (WFSTs)*

Formally, a transducer T is defined as the 8-tuple: $T = (A, X, Q, I, F, E, \lambda, \rho)$, where

- A is a finite input alphabet;
- X is a finite output alphabet;
- Q is a finite set of states;
- I is the set of initial states;
- F is the set of final states;
- $E \subseteq Q \times (A \cup \{\varepsilon\}) \times (X \cup \{\varepsilon\}) \times \mathbf{K} \times Q$ is a finite set of transitions. Each transition is specified by a five-tuple - the source state, the input symbol, the output symbol, the weight from a semiring $\mathbf{K}$ (e.g. the set of real numbers), and the destination state;
- $\lambda : I \mapsto \mathbf{K}$ is the initial state weight assignment;
- $\rho : F \mapsto \mathbf{K}$ is the final state weight assignment.

Theoretically, a WFST represents a weighted relation between sequences of input symbols and sequences of output symbols. The WFST can be easily understood by plotting it as a directed graph, where the nodes represent the states and the arcs represent the transitions. Fig.1 is a WFST example.

B. *Application to speech recognition*

The power of the WFST approach is that various knowledge sources used in speech recognition can all be consistently represented as WFSTs, and then can be combined and optimized, using the general WFST operations of composition, determinization, and minimization.

Specifically, separate transducers are first constructed for the four main knowledge sources - the language model G, the lexicon L, the phonetic context-dependency C, and the acoustic HMMs H. Using the composition operation (denoted as $\circ$), the determinization operation (det), and the minimization operation (min), the four transducers are then composed and optimized into a single integrated transducer,

$$N = min\Big(det\Big(H \circ det\big(C \circ det\big(L \circ G\big)\big)\Big)\Big) \quad (1)$$

that maps sequences of acoustic HMM states to sequences of words, and is known as the H-level transducer. As we can see in (1), the optimization operations are applied in intermediate steps in compiling the final transducer, which helps to improve the efficiency of composition and to reduce the intermediate transducer size.

The final integrated transducer N can be directly used in a Viterbi decoder. Note that the HMM state self-loops are not explicitly represented in N, they are simulated in the run-time decoder.

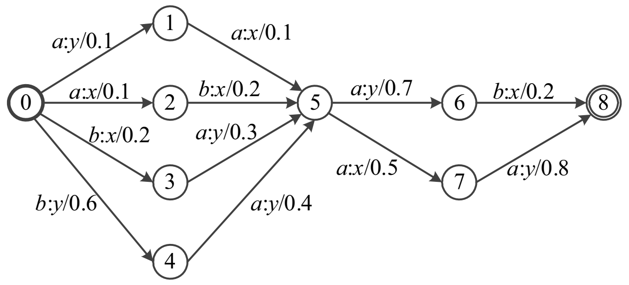

Figure 1. A WFST example. The initial states and final state are plotted as bold circles and double circles respectively. The input symbol *a*, output symbol *x* and weight 0.1 are marked on the corresponding arc by *a*:*x*/0.1.

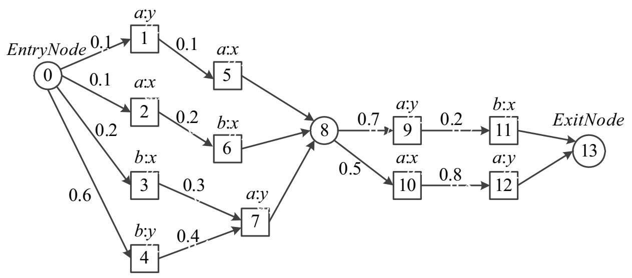

Figure 2. A FSG example. Content-nodes and virtual nodes are plotted as squares and circles respectively. The node content symbol *a* and the link record symbol *x* are marked above the corresponding node by *a*:*x*. The weight 0.1 is marked on the corresponding arc. Zero weights can be omitted for plotting.

III. Finite-state graph representation of the search network

As described in section II-B, when applying the WFST approach to speech recognition, the final compiled search network is represented as a standard WFST, which is then directly fed into the Viterbi decoder. While it is justified to use the WFST representation during compilation due to the fact that the powerful composition and optimization operations are well developed for WFSTs, it seems to take it for granted in previous studies that the representation of the search network during Viterbi searching remains to be the standard WFST.

In this work, we use the standard WFST representations and operations during compiling the search network. The compiled WFST is then equivalently converted to a new graphical representation, which we call finite-state graph (FSG) and is described as follows. The resulting FSG is more tailored to Viterbi searching for speech recognition and more compact in memory. The advantage of memory efficiency is due to the fact that the FSG representation exploits a special property of the final compiled WFST used in speech recognition, when compared to general WFSTs.

A. *Finite-state graphs (FSGs)*

One of the design considerations for FSGs is to make the representation of the search network in decoding more tailored to the Viterbi algorithm. The Viterbi algorithm is to find the most likely state sequence for the given observation sequence in a HMM. Note that the recognition transducer N essentially represents the state-space of a large flattened HMM, which is created by successive expanding the hierarchical knowledge sources - language model, lexicon, phonetic context-dependency, and acoustic HMMs[2]. So compared with the WFST, the proposed FSG is designed to be closer to the classic state-transition graph used in HMM studies [10].

Formally, a finite-state graph R is defined as the 6-tuple: $R = \left(V, NodeContentSet, LinkRecordSet, EntryNode, ExitNode, E\right)$, where

- V is a finite set of nodes. Each node has a node content symbol $a \subseteq NodeContentSet \cup \{\varepsilon\}$ and a link record symbol $x \subseteq LinkRecordSet \cup \{\varepsilon\}$.
- $NodeContentSet$ is a finite alphabet for node content;
- $LinkRecordSet$ is a finite alphabet for link record;
- $EntryNode$ is the unique entry node;
- $ExitNode$ is the unique exit node;
- $E \subseteq V \times \mathbf{R} \times V$ is a finite set of arcs. Each arc is specified by a triple - the source node, the weight in real numbers, and the destination node.

It is straightforward to plot a FSG as a directed graph. Fig.2 is a FSG example. A node is called a content-node and plotted as a square node, if the node content symbol is not the null symbol ε , otherwise the node is called a virtual-node and plotted as a circle node.

B. Conversions between WFSTs and FSGs

It is worthwhile to compare the FSG and the WFST representations. First, there is a common property for both FSGs and WFSTs. Theoretically, a FSG also represents a weighted relation between two sequences of symbols. Any path from the $EntryNode$ to the $ExitNode$ in a FSG is also associated with two sequences of symbols - the sequence of node content symbols and the sequence of link record symbols. This is much like a path from the initial states to the final states in a WFST, which is also associated with two sequences of symbols. In this analog, the node content symbols and the link record symbols in the FSG correspond to the input symbols and output symbols in the WFST respectively. The difference is that the symbols are associated with the nodes in FSGs, while in WFSTs, the symbols are associated with the arcs.

Second, note that it is always possible to convert a FSG to an equivalent[3] WFST by moving the symbols associated with each node either forward to every outgoing arcs, or backward to every incoming arcs. It is also possible to convert a WFST to an equivalent FSG, which, however, may require the introduction of extra nodes.

Consider the following symbol pushing operation in converting a WFST to a FSG, which is to push the symbols associated with each WFST-arc forward to the destination node, using the WFST-arc's input symbol and output symbol to be the FSG-node's content symbol and the link record symbol respectively. We cannot directly do the symbol pushing operation for converting a WFST to a FSG, since arcs with different pairs of input symbols and output symbols may point to the same destination node in the WFST. An example is to consider the four arcs pointing to node 5 in Fig.1, where there are three different pairs of input symbols and output symbols, $(a:x)$, $(b:x)$, and $(a:y)$. In this case, we introduce the following extra node generating operation. Several extra nodes are generated, each having incoming arcs labeled by the same pair of input symbol and output symbol. After applying the extra node generating operation, we can use the symbol pushing operation to create a valid FSG. Using the operations of extra node generating and symbol pushing, the equivalent FSG for the example WFST in Fig.1 can be created, as shown in Fig.2.

Recall that in the recognition transducer N , the input alphabet A is the set of all acoustic HMM-states, and the output alphabet X is the set of all possible words (i.e. the vocabulary). In our work, after we create the recognition transducer N , it is converted to an equivalent FSG N^{fsg} , in which the $NodeContentSet$ and the $LinkRecordSet$ are correspondingly the set of all acoustic HMM-states and the set of all possible words.

C. GrpDecoder description

Our GrpDecoder is a general-purpose, time-synchronous Viterbi beam search decoder, using the FSG representation of the search network. It can decode not only using H-level recognition FSG N^{fsg} , but also using the C-level FSG N^{fsg}_{CLevel} , which is converted from the C-level transducer N_{CLevel}

$$N_{CLevel} = C \circ min\left(det\left(L \circ G\right)\right)^{4} \tag{2}$$

In the latter case, the FSG-nodes corresponding to the context-dependent phones are dynamically expanded as the corresponding chain of the acoustic-HMM states. In both cases, the HMM state self-loops are simulated internally in the run-time decoder.

The decoder's search algorithm maintains a list of active states. At a high level, the search is a token passing algorithm [11], holding a token for each active state. It can be basically written as a loop over time frames and an inner loop over sets of active states. Every time we leave a node labeled by a non-null link record symbol, we create a new link record structure,

[2] This expansion is equivalent to the WFST composition of creating the recognition transducer N .

[3] A FSG and a WFST are equivalent if the weighted relations they represent are the same.

[4] It is also possible to do further minimization like in $min\left(det\left(C \circ min\left(det\left(L \circ G\right)\right)\right)\right)$, but this leads to a larger transducer and slower decoding speed in our experiments. There is similar result reported in [12].

containing the link record symbol, the end time for that symbol, and a backpointer to the previous link record structure. This is much similar to create the word link record structure in the classic token passing algorithm, except the definition of the link symbols is more flexible in the GrpDecoder. The link symbols in the GrpDecoder could be the words, the HMM states, or whatever desirable symbols for the user. After processing the last frame of the observation sequence, the best matching sequence of link record symbols can be obtained by backtracking from the token contained in the $\mathit{ExitNode}$.

Two types of pruning are currently supported in the GrpDecoder, the beam pruning and histogram pruning. The beam pruning is to retain only those tokens with a score close to the best token using a threshold called the beam-width. The histogram pruning is to limit the number of surviving tokens to a maximum number.

D. Advantages of using the FSGs

Now it is ready to explain the advantages of using the FSG representation of the search network during Viterbi searching over using the WFST representation.

First, note that the HMM-states which are matched against the observation sequence in Viterbi decoding, are contained in the nodes in FSGs, while they are contained in the arcs in WFSTs. Therefore, the Viterbi decoder using WFSTs has to maintain a list of active nodes *and* a list of active arcs. The Viterbi decoder using FSGs (like GrpDecoder) only needs to maintain a list of active nodes. This makes the token propagation in the GrpDecoder much simpler and more efficient. That is why we say that the FSG representation is more tailored to Viterbi searching for speech recognition.

Second, using the FSG representation reduces the memory storage of the search network for speech recognition. At first thought, in converting a WFST to an equivalent FSG, the number of nodes and arcs in the equivalent FSG (denoted as $|\text{equiv-FSG-nodes}|$ and $|\text{equiv-FSG-arcs}|$) are increased compared with the number of nodes and arcs in the original WFST (denoted as $|\text{WFST-nodes}|$ and $|\text{WFST-arcs}|$), due to the extra node generating operations. However, we find that the extra node generating operations are not frequently applied in practice. In the following experiments of converting the H-level and C-level WFSTs for two languages - English and Mandarin, it is observed that $|\text{equiv-FSG-nodes}|$ and $|\text{equiv-FSG-arcs}|$ are about 1.12 ~ 1.58 and 1.06 ~ 1.20 times of $|\text{WFST-nodes}|$ and $|\text{WFST-arcs}|$ respectively. This means that the percentage of the nodes in the final compiled WFST which have incoming arcs labeled by the same pair of input symbol and output symbol is pretty large, especially for the H-level WFST. This property is special for the final compiled WFST used in speech recognition, when compared to general WFSTs. In the following, we will explain how the FSG representation of the search network is more compact in memory than the WFST representation, due to this special property.

Typically, the main data structure for a WFST [13] is a linear array of all the arcs, sorted by source node. Each arc is a 5-tuple, containing the source node, the destination node, the input symbol, the output symbol, and the weight (20 bytes/arc). Each node has a pointer to the beginning of the outgoing arcs for that node (4 byte/node). Thus the memory required by a WFST is $4\times|\text{WFST-nodes}|+20\times|\text{WFST-arcs}|$. Note that the source node is not necessarily stored for each arc. Thus the memory storage for a WFST can be further reduced to

$$4\times|\text{WFST-nodes}|+16\times|\text{WFST-arcs}|\,. \qquad (3)$$

We use a similar data structure for a FSG. First, there is a linear array for all the arcs, sorted by source node. Each arc is represented by the destination node and the weight (8 bytes/arc). Second, there is a linear array for all the nodes. Each node contains the node content symbol, the link record symbol, and a pointer to the first arc leaving that node (12 bytes/node). Thus the memory required by an equivalent FSG to a WFST is

$$12\times|\text{equiv-FSG-nodes}|+8\times|\text{equiv-FSG-arcs}|\,. \qquad (4)$$

If we use (3) and (4) as the measures of the memory size for the WFST and FSG, it can be seen from Table I that using the FSG representation reduces the memory storage for the search network by 6.98% ~ 14.31%.

Finally, it is worthwhile to compare the FSG representation with some related researches. In [13], a new compact representation for WFSTs is proposed, which uses a variable length encoding for arcs and a two-level index for nodes. Similar techniques can also be applied to the representation of FSGs to further reduce the in-memory size. The work in [7] uses the weighted acceptor to represent the search network. A phenomenon similar to the above mentioned special property of the final compiled WFST is also observed for that acceptor representation. The state splitting used there is similar to our extra node generating operation. Despite these, the new FSG representation is essentially different from the weighted acceptor representation in [7].

IV. Benchmarking experiments

In this section, we present the performance evaluations of the three decoders, HDecode in HTK3.4 [9], Juicer-0.11.0 [8] and our own GrpDecoder on two languages - English and Mandarin. HDecode is an excellent decoder using dynamic network expansion. Juicer is a WFST-based decoder recently developed at IDIAP, which currently can only operate on C-level recognition transducers.

All the three decoders use the same acoustic models and language models as described below, and all are in single-threaded execution. The experiments are conducted on a 2.4GHz Intel Core2 machines with 2GB of memory.

A. Experimental setup for English

The benchmarking experiments on English are carried out using the WSJ0 continuous speech recognition corpus. The training data contain 7138 utterances. The test data are the November 1992 ARPA WSJ test set (Nov'92) which contains 330 sentences. The acoustic feature is 39-dimensional, formed by 12 Mel-frequency cepstral coefficients (MFCCs) with normalized log-energy and their first and second order differentials. Cepstral mean normalization (CMN) is applied for each utterance.

The phone set consists of 39 phones defined in the CMU pronunciation dictionary, augmented by a 3-state silence model and a single-state short pause model. The acoustic model is trained using HTK3.4, and is based on cross-word triphones modeled by 3-state left-to-right HMMs. A decision-tree based state tying is applied resulting in a total of 2011 triphone states. The state output densities are 16-component Gaussian mixture models with diagonal covariances. Nov'92 data is evaluated using the WSJ 5K closed vocabulary (non-verbalized punctuation). The language model is the WSJ 5K standard closed bigram. The perplexity of this bigram on Nov'92 data is 129.67.

B. Experimental setup for Mandarin

The benchmarking experiments on Mandarin are carried out using male speech data. The training data are a total of around 90 hours from 250 male speakers, including 863 database and data collected by our lab. A total of 600 utterances (0.9 hours) from other 5 male speakers are used for testing. In the front-end, a 45-dimensional feature vector is first extracted, including 14-dimensional MFCCs with normalized log-energy and their first and second order differentials. Considering that Mandarin is a tonal language, a 3-dimensional tone feature vector is appended to the spectral features, resulting in a final feature vector of 49-dimension. Cepstral mean and variance normalization (CVN) is applied for each utterance.

Considering the special syllable structure of Mandarin, the basic Mandarin phonetic units in our system are 100 consonant units each with 2 states (called initials), 164 vowel units each with 4 states (called finals), plus one single-state silence model. Using Mandarin phonetic classification for state tying, context-dependent triphones are created from the above basic phonetic units [14], resulting in a total of 2862 triphone states. The state output densities are 32-component Gaussian mixture models with diagonal covariances.

The modified Kneser-Ney smoothed bigram language model is trained from the People's Daily newspaper corpus (1994-2003). The perplexity of this bigram on the Mandarin test data (600 utterances) is 619.34.

Both the acoustic model and the language model in Mandarin experiments are trained using our own toolbox.

C. Sizes of search networks

In the experiments, we use the AT&T FSM library for WFST construction. Both C-level and H-level transducers are compiled using the above mentioned knowledge sources - the language model G , the lexicon L , the phonetic context-dependency C , and the acoustic HMMs H .

Table I gives the comparison of different representations of the search networks using the WFSTs and the equivalent FSGs separately for English and Mandarin. It can be seen from Table I that using the FSG representation consistently reduces the memory storage for the search networks for both languages. The memory reduction is more obvious for H-level networks than for C-level networks.

D. Decoding performance comparison for English

Fig.3 shows the performance curves of word error rates (WERs) versus real-time factors (RTFs) for English, using HDecode, Juicer, GrpDecoder operating on C-level and H-level networks.

It can be seen from Fig. 3 that the performances of HDecode and Juicer are close to each other. The GrpDecoder operating on the C-level network consistently achieves lower WER for a given RTF, when compared with HDecode and Juicer. This is a fair comparison between GrpDecoder and Juicer, since both decoders are tested operating on the same C-level network and using the same acoustic-HMM models in this comparison. The recognition performance can be further improved when using the GrpDecoder operating on the H-level network, which clearly shows the benefit of the fully optimization of the search network.

Finally, it is worthwhile to remark the run-time memory usage for different decoders. For the experiments above, Juicer requires 150MBs ~ 200MBs of memory during decoding, HDecode requires 50MBs ~ 100MBs of memory, C-level GrpDecoder requires 120MBs ~ 150MBs of memory, and H-level GrpDecoder requires only 90MBs ~ 110MBs of memory.

E. Decoding performance comparison for Mandarin

Fig.4 gives the performance curves of character error rates (CERs) versus real-time factors (RTFs) for Mandarin, using Juicer, GrpDecoder operating over C-level and H-level networks. Note that the Mandarin triphones used in our system is created using Mandarin phonetic classification for state tying, which is different from the decision-tree based state tying commonly used in HTK. So the triphone HMMs trained using our own toolbox are not compatible with the requirement of HDecode. Therefore, HDecode is not tested for Mandarin speech recognition.

The same observations can be drawn from Fig.4 as from Fig. 3. The performance advantages of C-level GrpDecoder over Juicer and H-level GrpDecoder over C-level GrpDecoder are clear for both English and Mandarin. The performance advantage of C-level GrpDecoder over Juicer in the Mandarin experiments appears to be more obvious than in the English experiments. Presumably this is because that the search networks in the Mandarin experiments are much larger and thus much heavier burden is placed on the run-time decoder. At this time, the power of the GrpDecoder which uses the more compact FSG representation and more tailored to Viterbi decoding, is fully demonstrated.

V. CONCLUSION AND FUTURE WORKS

This paper presents our effort to build a state-of-the-art WFST-based speech recognition system - GrpDecoder. We use the standard WFST representations and operations during compiling the search network. The compiled WFST is then equivalently converted to a new graphical representation - finite-state graph (FSG). The resulting FSG is more tailored to Viterbi decoding for speech recognition and more compact in memory.

Benchmarking of GrpDecoder is carried out separately on two languages - English and Mandarin. The test results show

TABLE I. Comparison of different representations of the search networks using the WFSTs and the equivalent FSGs for English and Mandarin. The columns "WFST-size" and "equiv-FSG-size" are the memory sizes for the WFSTs and FSGs respectively, measured using Equ. (3) and (4). The last column is the relative size reduction by using the FSG respresentation. The C-level and H-level search networks are defined in Equ. (1) and Equ. (2). The numbers in the parentheses in the column "$|\text{equiv-FSG-nodes}|$" and "$|\text{equiv-FSG-arcs}|$" are the ratios $|\text{equiv-FSG-nodes}|/|\text{WFST-nodes}|$ and $|\text{equiv-FSG-arcs}|/|\text{WFST-arcs}|$ respectively.

Networks		|WFST-nodes|	|WFST-arcs|	|equiv-FSG-nodes|	|equiv-FSG-arcs|	WFST-size	equiv-FSG-size	Size reduction
English	C-level	804,497	2,450,496	1,272,322 (1.58)	2,918,319 (1.19)	42,425,924	38,614,416	8.98%
	H-level	1,933,412	3,726,900	2,168,732 (1.12)	3,962,221 (1.06)	67,364,048	57,722,552	14.31%
Mandarin	C-level	1,810,354	5,037,233	2,794,761 (1.54)	6,021,640 (1.20)	87,837,144	81,710,252	6.98%
	H-level	4,931,736	9,382,167	5,692,914 (1.15)	10,143,345 (1.08)	169,841,616	149,461,728	12.00%

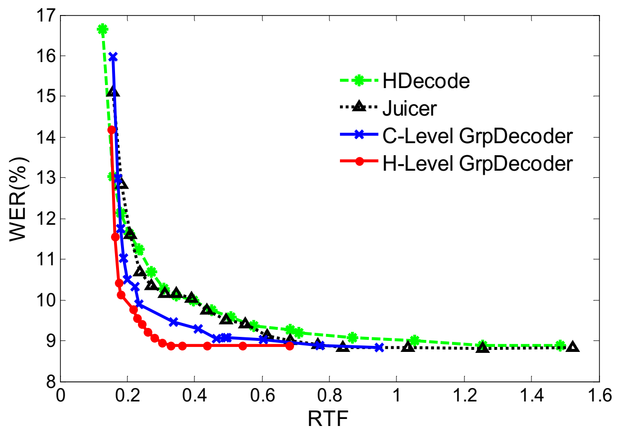

Figure 3. The performance curves of WERs versus RTFs for English using different decoders.

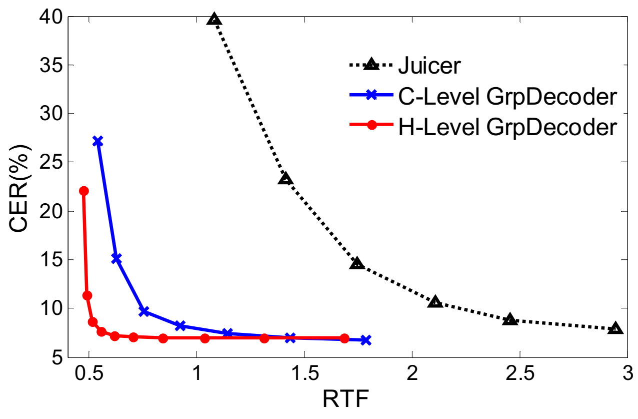

Figure 4. The performance curves of CERs versus RTFs for Mandarin using different decoders.

that GrpDecoder which uses the new FSG representation in searching is superior to HDecode and Juicer for both languages, achieving lower error rates for a given recognition speed. In the future, we plan to augment GrpDecoder with the functionality of on-the-fly composition and optimization.

References

[1] M. Mohri, F. Pereira, and M. Riley, "Speech Recognition with Weighted Finite-State Transducers," Handbook on Speech Processing and Speech Communication, Part E: Speech recognition, 2008.

[2] X. L. Aubert, "A Brief Overview of Decoding Techniques for Large Vocabulary Continuous Speech Recognition," In ISCA Automatic Speech Recognition workshop 2000, pp.91-97, Paris, France, September 2000.

[3] S. Kanthak, H. Ney, M. Riley, and M. Mohri, "A comparison of two LVR search optimization techniques," in Proc. ICSLP, 2002.

[4] S. Chen, "Compiling large-context phonetic decision trees into finite-state transducers". In Proc. Eurospeech 2003.

[5] D. A. Caseiro, I. Trancoso, "A specialized on-the-fly algorithm for lexicon and language model composition. IEEE Transactions on Audio, Speech, and Language rocessing, 14(4):1281-1291, 2006.

[6] Paul R.Dixon, Tasku Oonishi, Koji Iwano, Sadaoki Furui, "Recent Development of WFST-Based Speech Recognition Decoder" in APSIPA Annual Summit and Conference 2009.

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

[8] D. Moore, J. Dines, M.M. Doss, J. Vepa, O. Cheng and T. Hain, "Juicer: A weighted finite-state transducer speech decoder," In Processings of the 3rd Joint WorkShop on Multimodal Interaction and Related Machine Learning Algorithms, 2006.

[9] S.J. Young et al., "The HTK Book (for HTK Versinon 3.4)", Cambridge University Engineering Department, March 2009.

[10] L. Rabiner, "A tutorial on hidden Markov models and selected applications in speech recognition", Proceedings of the IEEE, 1989.

[11] 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.

[12] P. Garner, "Silence models in weighted finite-state transducers", in Proc. Interspeech 2008.

[13] D. Caseiro and I. Trancoso, "Using dynamic WFST composition for recognizing broadcast news", in Proc. ICSLP 2002.

[14] Chun Li, Zuoying Wang, "Phonetic classification-based triphone for continuous mandarin speech recognition," Journal of Tsinghua University, 2003, 43(1):16-19.