IDIAP RESEARCH REPORT

Juicer: A Weighted Finite-State Transducer speech decoder

Darren Moore [a] John Dines [a]
Mathew Magimai Doss [a] Jithendra Vepa [a]
Octavian Cheng [a] Thomas Hain [b]

IDIAP–RR 06-21

April 2006

to appear in
MLMI'06, Washington DC

[a] IDIAP Research Institute and Ecole Polytechnique Federale de Lausanne (EPFL), Martigny, Switzerland
[b] Department of Computer Science, University of Sheffield, UK

IDIAP Research Institute **www.idiap.ch**
Rue du Simplon 4 P.O. Box 592 1920 Martigny – Switzerland
Tel: +41 27 721 77 11 Fax: +41 27 721 77 12 Email: info@idiap.ch

IDIAP Research Report 06-21

Juicer: A Weighted Finite-State Transducer speech decoder

Darren Moore John Dines Mathew Magimai Doss Jithendra Vepa
Octavian Cheng Thomas Hain

April 2006

to appear in
MLMI'06, Washington DC

Abstract. A major component in the development of any speech recognition system is the decoder. As task complexities and, consequently, system complexities have continued to increase the decoding problem has become an increasingly significant component in the overall speech recognition system development effort, with efficient decoder design contributing to significantly improve the trade-off between decoding time and search errors. In this paper we present the "Juicer" (from trans***ducer***) large vocabulary continuous speech recognition (LVCSR) decoder based on weighted finite-State transducer (WFST). We begin with a discussion of the need for open source, state-of-the-art decoding software in LVCSR research and how this lead to the development of Juicer, followed by a brief overview of decoding techniques and major issues in decoder design. We present Juicer and its major features, emphasising its potential not only as a critical component in the development of LVCSR systems, but also as an important research tool in itself, being based around the flexible WFST paradigm. We also provide results of benchmarking tests that have been carried out to date, demonstrating that in many respects Juicer, while still in its early development, is already achieving state-of-the-art. These benchmarking tests serve to not only demonstrate the utility of Juicer in its present state, but are also being used to guide future development, hence, we conclude with a brief discussion of some of the extensions that are currently under way or being considered for Juicer.

1 Introduction

Speech recognition technology draws on a number of sources of knowledge and integrates these in the speech decoder to estimate the most likely word sequence from the given acoustical evidence. Typically these knowledge sources are represented in the form of hidden Markov models (HMM), pronunciation lexica, and N-gram language models. The means for combining these knowledge sources and efficient decoding of the acoustic input is a demanding task and a range of optimisation techniques and heuristics are employed to achieve lower computational and memory requirements with minimal sacrifice to recognition accuracy [1]. In this paper we present the "Juicer" decoding software that has been developed at IDIAP. The decoder is based on weighted finite-state transducer (WFST) theory, permitting simple decoder design through the efficient composition of a static decoding network.

We begin the paper with a short preamble, presenting our motivation for developing the Juicer decoder, followed in Section 3 by a brief overview of decoder technology and the primary design considerations, thus leading to Section 4 in which we present the Juicer system. In Section 5 we then follow-up with some preliminary benchmarking tests that have been carried out to date and in Section 6 an overview of future development directions for Juicer. Section 7 gives some brief concluding remarks concerning the material presented.

2 Why another speech decoder?

Over the years many decoding software packages employing a number of different decoding strategies and sporting varying capabilities have been made available to the research community and public at large, often in open source form. To name a few, there is HVite as part of HTK [4], Sphinx [11], NOWAY [8] not to forget IDIAP's own earlier effort, TODE [15]. One feature that all these decoders have in common is that they employ the acoustic, phonetic, lexical and linguistic knowledge sources in a manner that is hard-wired into the decoder architecture, thus making modifications to the decoder non-trivial. This can make the incorporation of new research into the decoder a significant undertaking (and possibly even infeasible for a given decoder architecture) and, as a result of this, it means that advancements to the state-of-the-art in speech recognition are often not included in the decoder and are rather used for rescoring decoder output, where their impact is likely to be more limited.

Not all decoder architectures suffer from such limitations. In recent years considerable effort has been invested in the development of more flexible decoder architectures based upon the theory of weighted finite-state transducers [14, 2] in which the decoding network is compiled independently of the decoder, thus enabling a more flexible approach to the incorporation of the various speech recognition knowledge sources. This approach also has some significant drawbacks, in particular, the memory demands for the compilation of static decoding networks for LVCSR systems can quickly grow beyond the capabilities of most machines, but efforts have also been made to alleviate this problem [10, 2]. While there has been significant efforts made towards developing decoder technology based upon WFST, unfortunately for the research community, to the best of our knowledge the availability of a state-of-the-art, open source decoder based upon WFST is yet to be realised.

There are many research groups around the world that are conducting significant research in LVCSR, many using their own in house recognition engine or relying on cooperation with industry for their decoder technology. In the present research environment, with many institutions and companies partnering up in European and international projects such as AMI and DARPA GALE, there is an increasing motivation for using systems and technologies that can be easily integrated and compared. In this respect there are no 'standard' recognition system configurations and file formats, but maintaining compatibility with widely accepted technologies, using modular system and software design and using open source distribution framework can help engender collaborative speech recognition research environments.

Thus far, we have identified key motivating factors for the development of new speech decoding software:

- The lack of a flexible speech decoder architecture, that not only provides state-of-the-art performance, but also acts as a platform for ongoing research, thus enabling it to maintain state-of-the-art performance in the future.
- There are decoding paradigms and methods for their implementation that offer a promising avenue to obtain the aforementioned flexible architecture, but for which there is not currently a solution freely available to the research community.
- Any new decoding software that is developed should be within an open-source development and distribution framework, to encourage on-going development and facilitate cooperation between research sites.

In the remaining sections we present a brief overview of speech decoding technology and, more specifically, the Juicer decoder which was developed as a response to these three driving forces.

3 LVCSR speech decoding

3.1 The decoding problem

Simply stated, the decoding problem in speech recognition is to find the most likely word sequence, $W_1^n = w_1, w2, \ldots, w_n$, given a sequence of acoustic observation vectors, $O_1^T = o_1, o_2, \ldots, o_T$, derived from the speech signal. This can be expressed by the equation:

$$\hat{W} = \arg\max_{W_1^n} \left\{ P(W_1^n) P(O_1^T | W_1^n) \right\} \tag{1}$$

$$= \arg\max_{W_1^n} \left\{ P(W_1^n) \cdot \sum_{S_1^T} P(O_1^T, S_1^T | W_1^n) \right\} \tag{2}$$

where W_1^n sequence of words from a vocabulary of size N_W, and S_1^T is any state sequence of length T.

Thus, our knowledge sources are incorporated into the decoder architecture by way of an hierarchical organisation:

- $P(W_1^n)$ comprises the language model (LM) which represents our prior **linguistic** knowledge independently of the observed acoustic information. Typically, language modelling is carried out using stochastic N-gram in which word probabilities are only dependent on the N-1 predecessors:
- $P(O_1^T | W_1^n)$ represents our model of the lexical, phonetic, and acoustic knowledge:
  - The **lexical** knowledge comprises the known words along with their pronunciation. Multiple pronunciations may be included, possibly with a prior probability for each pronunciation variant.
  - The **phonetic** knowledge describes the fundamental units in the pronunciation lexicon. These units are usually modelled in the context of their neighbours to account for the systematic, contextual variation that occurs in naturally spoken speech, even across word boundaries.
  - **Acoustic** knowledge is represented by way of the state emission probability density functions associated with each state of each context-dependent phoneme. In practice, various parameter tying schemes are used in emission PDF estimation to improve model of robustness.

A complete search of the solution space is practically infeasible, hence, a number of approaches have been developed to solve the decoding problem in a tractable fashion. In [1] a comprehensive overview of the major decoding strategies is given, with approaches being first grouped according to two main axes:

- Time-synchronous versus asynchronous decoding.
- Static versus dynamic expansion of the search space.

3.1.1 Time-synchronous versus asynchronous decoding

This axis concerns the manner in which the decoder network is searched; that is *breadth-first*, in which all hypotheses are evaluated in parallel in a time-synchronous fashion; and *depth-first*, in which the most promising hypothesis is pursued first until the end of the speech signal is reached, thus, the search is carried out asynchronously with respect to time. Of these approaches, the breadth-first approaches dominate today and it is with these approaches that we are concerned. Breadth-first decoding is more frequently referred to as Viterbi decoding, which approximates the solution to Equation 2 by only searching for the most probable state sequence:

$$\hat{W} \approx \arg\max_{W_1^n} \left\{ P(W_1^n) \cdot \max_{S_1^T} P(O_1^T, S_1^T | W_1^n) \right\} \tag{3}$$

3.1.2 Static versus dynamic expansion of the search space

Static network expansion requires the generation of the entire decoding network prior to decoding. This can enable significant reduction of the decoder search space by taking advantage of the sparsity of knowledge sources and permits a simple decoder design which operates independently of the network expansion stage. For LVCSR systems the static network can be prohibitively large, motivating the development of techniques for dynamic network generation, though the use of WFST approaches in LVCSR has to some extent alleviated these concerns and has seen a shift of interest back to static network generation.

Converse to static network expansion, dynamic network expansion forms an integral part of the speech decoder, enabling the handling of large scale decoding tasks as the decoding network is only ever partially expanded. A consequence of this is the need to incorporate a number of sub-optimal network composition techniques that can be applied on-the-fly; including phonetic prefix trees [6] and language model spreading [18]. Various methods are employed for the dynamic network expansion, each with its own pros and cons. An obvious drawback of the dynamic expansion paradigm is the need for the integration of network expansion and decoder, leading to a more complex and less flexible design.

3.1.3 Heuristics in decoder design

A number of heuristic techniques are employed, generally falling under the moniker of pruning methods. The objective of these techniques is to prevent the evaluation of unpromising decoder hypotheses, thus significantly saving in computational effort. The undesirable side effect of pruning is that sometimes the best path may be pruned, resulting in *search errors*. With the application of appropriate pruning strategies should be able to be limited to just a few percent of the total error. We will briefly describe two of the most common pruning techniques, beam search pruning and histogram pruning.

The basic idea behind beam search pruning is that those hypotheses with likelihood scores sufficiently below that of the highest scoring path can be safely disregarded. Hence, in a time-synchronous decoder, at each time step all hypotheses are evaluated against the highest scoring path and only those that fall within the beam search width are extend to next time step. Since not all knowledge source are evenly distributed across the decoding network it is common to employ multiple pruning thresholds at a number of different stages of decoding.

Histogram pruning applies an upper threshold on the maximal number of active hypotheses. In [18] it was shown that the maximum number active paths exceeded by almost two orders of magnitude the average number of active paths. By taking a histogram of the scores for all of the active paths and discarding all paths that fall below this upper threshold it was observed that computational demand could further reduced from regular beam-form with minimal impact on search accuracy.

It need be noted that there are many more techniques for pruning and for reducing search errors due to pruning, but the application of these approaches is often dependent on the decoder architecture. Furthermore, it is worth observing that different decoding architectures tend to lend themselves to more or less effective pruning, thus making pruning an important feature in decoder design.

3.2 WFST and speech decoding

While the use of static networks in speech decoding is far from being a new idea, the explicit use of weighted finite-state transducers is relatively recent. Pioneered by Mohri and others at AT&T [12], the key advantage behind the use of WFSTs for speech decoding is that it enables the integration and optimisation of all knowledge sources within the same generic representation. This provides a more efficient framework for carrying out speech recognition and also enables greater ease for the integration of new knowledge sources in various stages of the system hierarchy. In this section we briefly describe the key features of WFST theory and its application to speech decoding.

3.2.1 Overview of WFST

A weighted finite-state transducer is a finite-state automaton with state transitions labelled with input and output symbols and each transition having an associated weighting. Sequences of input symbols are thus mapped to sequence of output symbols with a weighting value which is calculated over all valid paths through the transducer, where each path weight is a function of all the state-transition weights associated with that path. An example of a simple WFST is shown in Figure 1. WFST algorithms comprise a number of fundamental operations for composition and optimisation, which are briefly summarised below. Further details of the algebraic notation and algorithms for WFST can be found in [14, 12].

Composition Composition is used to combine transducers of different levels of representation. The operation $C = A \circ B$ specifies the composition of two transducers A and B with input/output symbols x/y and y/z, respectively, into a single transducer, C, with input/output symbols x/z and weights calculated to give the same weighting to all possible input/output sequences as the original separate transducers.

Determinization A transducer is deterministic if and only if each of it's states has at most one transition for any given input label and there are no *epsilon* input labels[1]. Determinization, denoted $\det(C)$, serves to reduce redundancy in the network thus reducing the time taken to match paths with input sequences.

Minimisation A minimised automata, $D = \min(C)$, is equivalent to automata C and has the least number of states and the least number of transitions among all deterministic automata equivalent to C. As the weighting of transitions tends to result in all transitions being distinct classical minimisation techniques tend to be ineffective. In order to alleviate this problem the WFST network first undergoes weight pushing in which all transitions in the transducer are reweighted to facilitate minimisation. Typically this involves a shifting of transition weights to the beginning of the network, but there is no overall effect on the total weights associated with paths through the network.

[1] *epsilon* (ϵ) labels consume no input or produce no output.

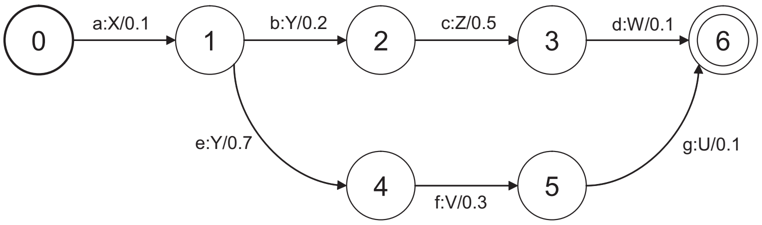

Figure 1: An example of a simple WFST: one path through the network would have input label sequence *abcd*, output label sequence $XYZW$ and weight $f(0.1, 0.2, 0.5, 0.1)$

3.2.2 Application to LVCSR

The application of WFST in LVCSR requires the representation of each of the knowledge sources as weighted finite state automata, which subsequently undergo composition and can then be optimised using the determinization and minimisation to produce a compact and efficient decoding network, as previously described. Typically, separate transducers are constructed for the N-gram language model, G, the lexicon, L, and the context dependency expansion, C. Though not currently supported in Juicer, HMM state level topology, H, and phonological information, P, can also be incorporated into the network structure:

$$N = H \circ C \circ P \circ L \circ G \tag{4}$$

In order to ensure that the entire transducer can be determinized it is necessary to undertake some additional steps:

1. In order to make the lexicon and grammar composition $L \circ G$ determinizable, the addition of an auxiliary phone symbol marking word endings in the lexicon is necessary, giving $\tilde{L}$. This auxiliary symbol must then be repeated in the transducers below lexical level (eg. $\tilde{C}$, $\tilde{P}$ and $\tilde{H}$) which at completion of determinization/minimisation undergo an erasing operation, π_ϵ, which replaces the auxiliary symbols with ϵ-labels.

2. Similarly, the context dependency transducer is generally not deterministic as there may be multiple state transitions with the same input symbol (representing the different contexts in which that symbol can occur). Building of a compact context dependency transducer can be achieved by creating the inverse of the context dependency transducer, which can be simply determinized and then inverting the resultant transducer.

3. Factoring can be employed as a last optional step, which can be used to save storage space of the transducer. This is achieved by taking any long chains of single incoming-single outgoing state transitions and replacing these with a single state in the network. These chains may then be stored separately and incorporated into the network at run-time.

Thus, the composition and optimisation of the entire static network can be expressed as follows:

$$N = \pi_\epsilon(\min(\det(\tilde{H} \circ \det(\tilde{C} \circ \det(\tilde{P} \circ \det(\tilde{L} \circ G)))))) \tag{5}$$

Further to this, additional steps may need to be taken when dealing with large vocabularies, $N_W \gtrsim 50k$, and long span language models, $N \geq 3$. Several approaches to this end have been investigated by researchers, including language model pruning, finite-state language model approximation, "on-the-fly" composition techniques and dynamic transducer composition, similar to that employed in traditional dynamic network generation based decoders, though still employing the general WFST framework. These issues will be further touched upon in the benchmarking and future development sections in this paper.

4 An overview of Juicer

The Juicer decoder uses a time-synchronous Viterbi search based on the token-passing algorithm with beam-search and histogram pruning, as previously described in this paper. At run time the decoder dynamically expands the model-level transducer network into a state-level network that is suitable for finding the best state-level path subject to knowledge source constraints, hence, optimisation is not yet carried out to take advantage of further state-level redundancies arising from HMM parameter sharing. The package consists of a number of command line utilities in addition to the Juicer decoder itself; more specifically, a number of tools are provided for the generation, composition and optimisation of the ASR knowledge sources (language model, pronunciation dictionary, acoustic models) into a single WFST that is input to the decoder. For the composition and optimisation of WFST resources Juicer relies on the functionality of the AT&T Finite State Machine Library [13] and/or MIT FST toolkit [7]. Figure 2 illustrates the modular organisation of the Juicer utilities.

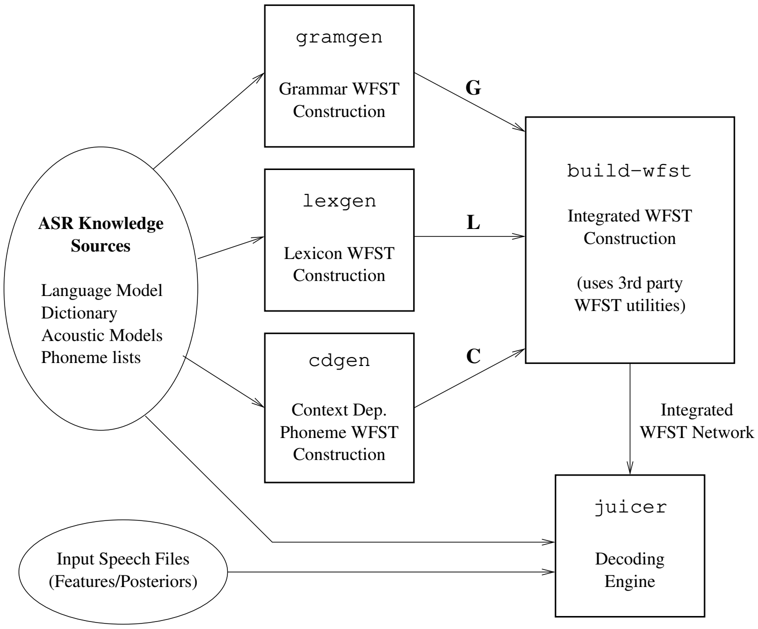

Figure 2: High level architecture of the Juicer decoding package.

The major features of Juicer and its utilities are summarised as follows; further details can be found in the user manual [16]:

- `juicer`: decoding search engine
  - Flexible WFST-based Viterbi decoder (decoding network fully independent of decoding engine implementation)
  - Beam-search (global, model-end) and histogram pruning
  - Lattice generation in AT&T FSM format
  - Word-level or model-level output with timing information
- `gramgen`: language model WFST generation
  - Simple word-loop with start/end silence
  - ARPA Naval Resource Management style word-pair grammar

  - ARPA MIT-LL text format N-gram (arbitrary N, subject to memory limitations)
- `lexgen`: dictionary WFST generation
  - Multiple pronunciations, with optional pronunciation probabilities
  - Tee models are handled via optional silence/short pause in the dictionary
- `cdgen`: acoustic model/context dependency WFST generation
  - Monophone, word-internal n-phones (tri/quin/...), cross-word triphones
  - HTK MMF file format support
  - Hybrid HMM/ANN decoding supported (using LNA-format posterior files)
- `build-wfst`: WFST composition and optimisation
  - Calls to AT&T and MIT FST routines
  - Supports optional determinization and minimisation of the final transducer (the most memory demanding step)

5 Benchmarking experiments

Benchmarking of Juicer was carried out with two main aims; the first was to assess its performance purely from the word error rate versus pruning efficiency standpoint, and the second was to investigate its capabilities in the context of a very large vocabulary task with long span language models in which the size of the network was going to be a limiting factor.

For the first step of experiments, a system developed using the WSJ1 continuous speech recognition corpus [17]. Three-state, cross-word triphone, decision tree state-clustered CDHMM models were trained using HTK on the "si_tr_s" set of 38,275 utterances. Models were trained from 39 dimensional MF-PLPs including delta and delta-delta features, with speaker side-based cepstral mean and variance normalisation. The pronunciation dictionary was based off that used for AMI RT05s system [5]. The standard MIT bigram and trigram backed-off language models were used with the 20k development test set "si_dt_20" from WSJ1 database, consisting of 503 utterances. Figure 3 shows the results for the various systems tested.

We can see that HDecode is achieving better performance in terms of beam-width versus word-error rate, but quickly converge within a fraction percent by a beam width of 200. We postulate that this result derives from HDecode's use of multiple tokens per state,[2] which can be beneficial to performance by enabling the generation of more active hypothesis for the given beam width. A comparison of real-time factors could also give more insight into the differences between the two decoders, but remained outside the scope of our initial benchmarking tests as decoding experiments were conducted at different sites.

For the second set of experiments the AMI RT05s system was used. First-pass decoding in this system uses three-state cross-word triphone models and 50k lexicon with backed-off trigram language model comprising some 29 million bigrams and 40 million trigrams, ensuring that static composition of these resources was going to be a formidable task. In order to compare the practicality of constructing a decoding network for such a system, pruned versions of the AMI language model were produced and compiled along side the full LM. Table 1 shows the outcome of the composition experiments.

We see that the size of the network grows significantly with more relaxed pruning, and in the unpruned case the final composition stage failed! In light of the size of the language model transducer this was not at all surprising and this behaviour has also been reflected in the use of relatively aggressively pruned language models in some of the published literature [14]. Despite this, we were interested in evaluating the performance of Juicer against HDecode on the RT05s recognition task

[2]We were unable to disable this functionality.

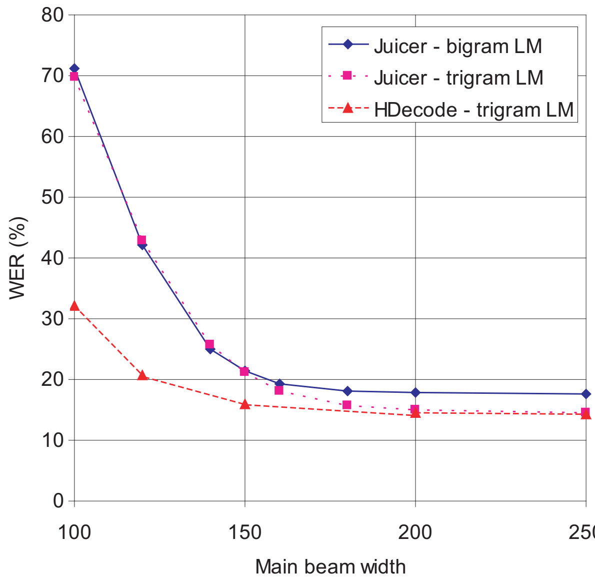

Figure 3: Results for WSJ 20k task, showing WER versus main beam-width. Sizes of trigram transducer with Juicer (thousands of arcs): $C = 9929$, $L = 50$, $G = 15619$ and $C \circ L \circ G = 33378$

Language Model	FSM Software	Number of arcs (thousands)					Time (hrs)
		G	L	C	$L \circ G$	$C \circ L \circ G$	
Pruned-08	AT&T + MIT	4,145	127	1,065	7,008	14,945	0:30
Pruned-09	MIT	13,692	127	1,065	23,160	50,654	1:44
Pruned-10	MIT	35,895	127	1,065	59,626	120,060	5:38
Unpruned	MIT	98,288	127	1,065	DNF	DNF	10:33

Table 1: Network composition experiments for AMI RT05s system. DNF – did not finish. Pruned-XX – language model pruning factor XX, where all N-grams are pruned that reduce language model perplexity on the training data by less than 10^{-XX} relative. The AT&T toolkit could only be used for the smallest language model as the library is not available with a 64-bit compilation.

using a heavily pruned LM. The results are shown in Table 2. We see that despite the heavy pruning of the LM (in fact, this LM is more heavily pruned than all those shown in Table 1) the results are still resectable, with only 5% relative increase in WER. Future benchmarking experiments will look into profiling the relationship between WER and language model pruning, including the effect that this has on decoding speed and lattice generation and rescoring accuracy.

6 Future development

The results of early benchmarking experiments indicate that Juicer is currently severely hampered when used for large vocabulary tasks with large, high-order N-gram language models. Hence, a priority of future development is to extend its ability with higher-order language models, however, the problem of meeting memory requirement of such tasks through the brute force approach is seemingly unsurmountable. This is a consequence of the fact that, during composition, the size of the resultant transducer can be as big as the product of its constituents [9]. As we have demonstrated, for cases of higher-order language models, the composition algorithm, as well as the following optimisation

System	TOT	Sub	Del	Ins
P1.HDecode	41.1	21.1	14.7	5.3
P1.Juicer	43.5	23.0	13.7	6.8
P2.HDecode	33.1	15.9	13.4	3.9
P2.Juicer	34.5	16.9	13.6	4.0

Table 2: % WER results on RT05s individual headset microphone task for HDecode (full LM) and Juicer (Pruned-07 LM). The P1 system uses ML trained models, the P2 system includes VTLN, MPE trained models, and HSLDA feature transform. Further details of the evaluation system can be found in [5]

procedure, can easily fail due to lack of memory. Even if the final transducer could be successfully generated, the size may still be too large for decoding to be carried out on a conventional PC.

One of the possible solutions to this problem is to perform on-the-fly transducer composition during decoding. Acoustical, phonetic and lexical resources may still be composed and optimised off-line, while the language model transducer is locally, dynamically composed at run time [3, 19, 9]. By using this approach, we can avoid composing part of the search space which is not traversed by any hypotheses. In addition, the total size of the constituent transducers will be much smaller than the integrated transducer. This approach carries certain disadvantages in terms of introducing extra overheads during decoding and transducer optimisation operations can not be performed on the full transducer possibly leading to sacrifice to performance.

Future development of Juicer will aim to assess dynamic transducer composition along side alternative schemes, including the investigation of improved static composition techniques developed as part of the FSA toolkit, which have been demonstrated to achieve much more memory efficient composition [10] and multiple-pass decoding strategies that enable more sparse language models to be used on the first pass. Furthermore, the implementation of on-the-fly transducer composition still permits a flexible decoder architecture and need not be necessary in all applications.

7 Concluding remarks

In this paper we have presented the Juicer speech recognition decoder developed at IDIAP. The decoder employs a statically built decoding network based upon weighted finite-state transducer theory. In benchmarking experiments we have demonstrated some of the capabilities of the decoder, in particular, we have shown that on a medium vocabulary task performance with HDecode compares favourably with moderate to wide pruning settings, while on a large vocabulary task some of the drawbacks of the current system were identified, although in spite of this, respectable WER was still able to be achieved. We have also described some of our future plans for Juicer development, more specifically, those aimed at addressing the issues raised during benchmarking. Presently, the Juicer decoder and utilities, including source code, are only available to AMI partners, but we envisage that the decoder and utilities will soon be made available to the wider research community.

8 Acknowledgements

The authors would like to thank all those involved in the development of the AMI ASR system, which formed the basis of some of the benchmarking evaluations that were carried out. We would also like to thanks Cambridge University Engineering Department for the right to use Gunnar EvermannŠs HDecode at the University of Sheffield; and Adam Janin and Chuck Wooters from ICSI, who allowed us to carry out the large network composition on one of their recently acquired 64-bit AMD Opteron Processors. This work was supported by the European Union 6th FWP IST Integrated Project AMI

(Augmented Multi-party Interaction, FP6-506811) and the Swiss National Center of Competence in Research (NCCR) on Interactive Multi-modal Information Management (IM)2.

References

[1] X. Aubert. An overview of decoding techniques for large vocabulary continuous speech recognition. *Computer Speech and Language*, 16(1):89–114, January 2002.

[2] D. A. Caseiro. *Finite-state methods in automatic speech recognition*. PhD thesis, Instituto Superior Téchnico, Universidade Técnica de Lisboa, December 2003.

[3] H. Dolfing and I. Hetherington. Incremental language models for speech recognition using finite-state transducers. In *Proc. IEEE ASRU2001*, 2001.

[4] S. Young et. al. *The HTK Book*. Cambridge University Engineering Department, December 2002. For HTK Version 3.2.1.

[5] T. Hain et. al. The 2005 AMI system for the transcription of speech in meetings. In *Proc. NIST RT05 Workshop*, Edinburgh, July 2005.

[6] R. Haeb-Umbach and H. Ney. Improvements in beam search for 10000-word continuous speech recognition. *IEEE Trans. on Speech and Audio Processing*, 2(2):353–356, April 1994.

[7] L. Hetherington. The MIT FST toolkit. MIT Computer Science and Artificial Intelligence Laboratory: *http://people.csail.mit.edu/ilh//fst/*, May 2005.

[8] M. Hochberg, S. Renals, A. Robinson, and D. Kershaw. Large vocabulary continuous speech recognition using a hybrid connectionist-HMM system. In *Proc. ICSLP*, pages 1499–1502, Yokohama, Japan, 1994.

[9] 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 *Proc. Interspeech (ICSLP)*, volume 1, pages 289–292, 10 2004.

[10] S. Kanthak and H. Ney. FSA: An efficient and flexible C++ toolkit for finite state automata using on demand computation. In *Proc. ACL*, pages 510–517, Barcelona, Spain, July 2004.

[11] K. F. Lee. *Automatic Speech Recognition – The Development of the Sphinx System*. Kluwer Academic Publishers, Norwll, Mass., 1989.

[12] M. Mohri. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(2), 1997.

[13] M. Mohri, F. Pereira, and M. Riley. General-purpose finite-state machine software tools. AT&T Labs – Research: *http://www.research.att.com/sw/tools/fsm*, 1997.

[14] M. Mohri, F. Pereira, and M.l Riley. Weighted finite-state transducers in speech recognition. *Computer Speech and Language*, 16(1):69–88, January 2002.

[15] D. Moore. *TODE: A Decoder for Continuous Speech Recognition*. IDIAP Research Institute, Martigny, Switzerland, 2002.

[16] D. Moore. *The Juicer LVCSR decoder - user manual*. IDIAP Research Institute, Martigny, Switzerland, August 2005. for Juicer version 0.5.0.

[17] D. B. Paul and J. M. Baker. The design for the wall stree journal-based CSR corpus. In *Proc. ICSLP*, 1992.

[18] V. Steinbiss, B.-H. Tran, and H. Ney. Improvements in beam search. In *Proc. ICSLP*, pages 2143–2146, Yokohama, Japan, September 1994.

[19] D. Willett and S. Katagiri. Recent advances in efficient decoding combining on-line transducer composition and smoothed language model incorporation. In *Proc. ICASSP*, volume 1, pages 713–716, 5 2002.