Julius — an Open Source Real-Time Large Vocabulary Recognition Engine

Akinobu Lee, Tatsuya Kawahara**, Kiyohiro Shikano**

* Graduate School of Information Science, Nara Institute of Science and Technology, Japan
** School of Informatics, Kyoto University, Japan
ri@is.aist-nara.ac.jp

Abstract

Julius is a high-performance, two-pass LVCSR decoder for researchers and developers. Based on word 3-gram and context-dependent HMM, it can perform almost real-time decoding on most current PCs in 20k word dictation task. Major search techniques are fully incorporated such as tree lexicon, N-gram factoring, cross-word context dependency handling, enveloped beam search, Gaussian pruning, Gaussian selection, etc. Besides search efficiency, it is also modularized carefully to be independent from model structures, and various HMM types are supported such as shared-state triphones and tied-mixture models, with any number of mixtures, states, or phones. Standard formats are adopted to cope with other free modeling toolkit. The main platform is Linux and other Unix workstations, and partially works on Windows. Julius is distributed with open license together with source codes, and has been used by many researchers and developers in Japan.

1. Introduction

In recent study of large vocabulary continuous speech recognition, we have recognized the necessity of a common platform. By collecting database and sharing tools, the individual components such as language models and acoustic models can be fairly evaluated through actual comparison of recognition result. It benefits both research of various component technologies of and development of recognition systems.

The essential factors of a decoder to serve as a common platform is performance, modularity and availability. Besides accuracy and speed, it has to deal with many model types of any mixtures, states or phones. Also the interface between engine and models needs to be simple and open so that anyone can build recognition system using their own models. In order to modify or improve the engine for a specific purpose, the engine needs to be open to public. Although there are many recognition systems recently available for research purpose and also as commercial products, not all engines satisfy these requirement.

We developed a two-pass, real-time, open-source large vocabulary continuous speech recognition engine named "Julius" for researchers and developers. It incorporates many search techniques to provide high level search performance, and can deal with various types models to be used for their cross evaluation. Standard formats that other popular modeling tools[1][2] use is adopted. On a 20k-word dictation task with word 3-gram and triphone HMM, it realizes almost real-time processing on most current PCs.

Figure 1: Screen shot of Japanese dictation system using Julius.

Julius has been developed and maintained as part of free software toolkit for Japanese LVCSR[4] from 1997 on volunteer basis. The overall works are still continuing to the Continuous Speech Recognition Consortium, Japan[3]. This software is available for free with source codes. A screen shot of Japanese dictation system is shown in Figure 1.

In this paper, we describe its search algorithm, module interface, implementation and performance. It can be downloaded from the URL shown in the last section.

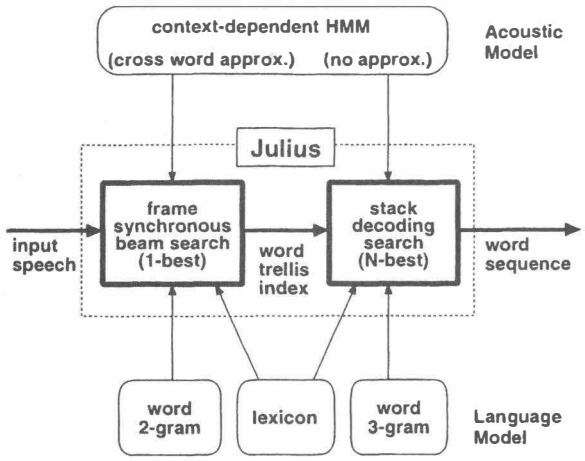

Figure 2: Overview of Julius.

2. Search Algorithm

Julius performs two-pass (forward-backward) search using word 2-gram and 3-gram on the respective passes[5]. The overview is illustrated in Figure 2. Many major search techniques are incorporated. The details are described below.

2.1. First Pass

On the first pass, a tree-structured lexicon assigned with language model probabilities is applied with the frame-synchronous beam search algorithm. It assigns pre-computed 1-gram factoring values to the intermediate nodes, and applies 2-gram probabilities at the word-end nodes. Cross-word context dependency is handled with approximation which applies the best model for the best history. As the 1-gram factoring values are independent of the preceding words, it can be computed statically in a single tree lexicon and thus needs much less work area. Although the values are theoretically not optimal to the true 2-gram probability, these errors can be recovered on the second pass.

We assume one-best approximation rather than word-pair approximation. The degradation by the rough approximation in the first pass is recovered by the tree-trellis search in the second pass. The word trellis index, a set of survived word-end nodes, their scores and their corresponding starting frames per frame, is adopted to efficiently look up predicted word candidates and their scores on the later pass. It allows recovery of word boundary and scoring errors of the preliminary pass on the later pass, thus enables fast approximate search with little loss of accuracy.

2.2. Second Pass

In the second pass, 3-gram language model and precise cross-word context dependency is applied for re-scoring. Search is performed in reverse direction, and precise sentence-dependent N-best score is calculated by connecting backward trellis in the result of the first pass. The speech input is again scanned for re-scoring of cross-word context dependency and connecting the backward trellis. There is an option that applies cross-word context dependent model to word-end phones without delay for accurate decoding. We enhanced the stack-decoding search by setting a maximum number of hypotheses of every sentence length (envelope), since the simple best-first search sometimes fails to get any recognition results. The search is not A*-admissible because the second pass may give better scores than the first pass. It means that the first output candidate may not be the best one. Thus, we compute several candidates by continuing the search and sort them for the final output.

2.3. Gaussian Pruning and Gaussian Mixture Selection on Acoustic Computation

For efficient decoding with tied-mixture model that has a large mixture pdfs, Gaussian pruning is implemented[6]. It prunes Gaussian distance (= log likelihood) computation halfway on the full vector dimension if it is not promising. Using the already computed k-best values as a threshold guarantees us to find the optimal ones but does not eliminate computation so much [safe pruning]. We implement more aggressive pruning methods by setting up a beam width in the intermediate dimensions [beam pruning]. We perform safe pruning in the standard decoding and beam pruning in the efficient decoding.

To further reduce the acoustic computation cost on triphone model, a kind of Gaussian selection scheme called Gaussian mixture selection is introduced[7]. Additional context-independent HMM with smaller mixtures are used for pre-selection of triphone states. The state likelihoods of the context-independent models are computed first at each frame, and then only the triphone states whose corresponding monophone states are ranked within the k-best are computed. The unselected states are given the probability of monophone itself. Compared to the normal Gaussian selection that definitely select Gaussian clusters by VQ codebook, the unselected states are reliably backed-off by assigning actual likelihood instead of some constant value, and realize stable recognition with even more tight condition.

2.4. Transparent Word Handling

Toward recognition of spontaneous speech, transparent word handling is also implemented for fillers. The N-gram probabilities of transparent words are given as same as other words, but they will be skipped from the word

context to prevent them from affecting occurrence of neighbor words.

2.5. Alternative algorithms

Besides these algorithms described above, conventional algorithms are also implemented for comparison. The default algorithms described above such as 1-gram factoring, one-best approximation and word trellis index can be replaced to conventional 2-gram factoring, word-pair approximation and word graph interface respectively. They are selectable on compile time and any combination is allowed. Users can choose suitable algorithms for their evaluation and development.

3. Module Interface

In order to act as a module of various speech recognition systems, a recognition engine needs to have simple and trivial interface to other modules. We adopt standard and common format as module interface to keeps generality and modularity to various models. The interfaces, speech input and output of Julius is described in the following.

3.1. Acoustic Model

Monophone and triphone HMM with any number of mixtures, states, phones are supported. It can also handle tied-mixture models and phonetic tied-mixture model[6]. The model types are automatically identified. The HMM definition file should be in HTK format. When tied-mixture model is given (by hmmdefs using directive `<TMix>`), Gaussian pruning is activated for each mixture pdfs. Not all formats in HTK hmmdefs are supported: multi input stream, discrete HMM, covariance matrix other than diagonal and duration parameter are not supported.

3.2. Lexicon

The format of the recognition dictionary is similar to the HTK dictionary format. It is a list of words with their output strings and pronunciations. Each pronunciation is expressed as a sequence of sub-word unit name in the acoustic model. Actually any sub-word unit like syllables can be used. Multiple pronunciations of a word should be written as separate words, each has possible pronunciation pattern. With a triphone model, each pronunciation unit is converted to context-aware form (ex. "a-k+i") on startup. To map possible (logical) triphones to the defined (physical) ones in the hmmdefs, acoustic model should be accompanied with HMM list that specifies the correspondences.

The default maximum vocabulary size is set to 65535 words by default for memory efficiency, but larger size can be allowed if configured so.

3.3. Language Model

Two language model is needed: word 2-gram for the first pass and word 3-gram in reverse direction for the second pass. ARPA-standard format can be directly read. To speed up the startup procedure of recognition system, an original binary format is supported[1].

3.4. Input / Output

Speech input by waveform file (16bit PCM) or pattern vector file (HTK format) is supported. Live microphone input is also supported on Linux, FreeBSD, Sun and SGI workstations. Input can also be sent via TCP/IP network using DAT-Link/netaudio protocol. These input speech stream can be segmented on pause by watching zero-cross and power, and each segment is recognized sequentially in turn.

Decoding of the first pass is done in parallel with the speech input. It starts processing as soon as a input segment starts, and when a long pause is detected, it finishes the first pass and continues to the second pass. As the second pass finishes in very short time, the delay of recognition result output is sufficiently small.

The current speech analysis function of Julius can extract only one coefficients for our acoustic models[2]. CMN (cepstral mean normalization) is activated automatically if acoustic model requires it. In case of file input, cepstral mean is computed first for the file or segment. For live input, average values of last 5 seconds are used.

Output is a recognition result in word sequence. N-best results can be output. Phoneme sequence, log likelihood scores and several search statistics can also be generated. Partial results can be output successively while processing the first pass, though the final result is undetermined till the end of the second pass.

3.5. Search Parameters

Various search parameters can be determined in both compile time and run time. The parameters of language model weight and insertion penalty as well as the beam width can be adjusted for the respective passes. Two default decoding options are also set up: Standard decoding strictly handles context dependency of acoustic models for accurate recognition. Efficient decoding uses a smaller beam width by default and terminated the search when the first candidate is obtained.

4. Implementation and Distribution

Main platform is Linux, but it also works on other Unix workstations: FreeBSD, Solaris2, IRIX6.3 and Digital Unix. Live microphone input is supported on most OS. It

[1] A conversion tool "`mkbingram`" is included in the distribution package.

[2] 26 dimension `MFCC_E_D_Z` coefficients only.

Table 1: Performance of 20k Japanese dictation system

system	efficient	accurate
acoustic model	PTM 129x64	triphone 2000x16
Julius conf.	fast	standard
CPU time	1.3 xRT	5.0 xRT
Word acc. (male)	89.0	94.4
Word acc. (female)	91.8	95.6
Word acc. (GD avg.)	90.4	95.0
Word acc. (GID)	89.7	93.6

Julius-3.2, CPU: Pentium III 850MHz
(without Gaussian Mixture selection)

can also run on Microsoft Windows. We are now working on the Windows version to be fully functioned, but currently the features are limited.

The whole source code is distributed freely. Preparing acoustic model and language model, one can construct a speech recognition system for their tasks. Users can modify the source or part of them for their applications without explicit permission of the authors, both in research purpose, and even in commercial purpose.

Julius has been developed since 1996. The recent revision is 3.2 and development is still continuing on volunteer basis. The source codes are written in C language, and its total amount is about 22,000 lines and 604 Kbytes. Although it has been developed and tested on Japanese environment, it should work on other language with little modification.

5. Evaluation on Japanese Dictation task

Performance of the total Japanese dictation system with Julius and typical models provided by the IPA Japanese dictation toolkit[4] is summarized in Table 1 for 20k system. Two typical configuration are listed: efficiency-oriented and accuracy-oriented. Note that the Gaussian mixture selection and transparent word handling is not included in this experiment.

The accurate version with triphone model and standard decoding achieves a word accuracy of 95%. The efficient version using the PTM model keeps the accuracy above 90% and runs almost in real-time at a standard PC. The required memory is about 100Mbytes for the efficient version and about 200Mbytes for the accurate version. This difference comes mainly from acoustic probability cache, as all state probabilities of all frame is cached in the first pass to be accessed on the second pass.

The total system performance integrating Gaussian mixture selection is shown in Table 2. Real-time factor of 1.06 is achieved even with standard setting, and the word accuracy reaches 92.1%.

Table 2: Total Performance

system	efficient + GMS
acoustic model	PTM 129x64
Julius conf.	standard
CPU time	1.0 xRT
Word acc. (male)	90.7
Word acc. (female)	93.5
Word acc. (GD avg.)	92.1

Julius-3.2, CPU: Pentium III 850MHz

6. Conclusions

A two-pass, open-source large vocabulary continuous speech recognition engine Julius has been introduced. It has an ability to achieve word accuracy of 95% in accurate setting, and over 90% in real-time processing in 20k-word dictation. It is well modularized with simple and popular interface to be used as an assessment platform. It provides total recognition facility with the current state-of-the-art search techniques open to all researchers and developers engaging in speech-related works.

It has been used by many researchers and developers in Japan as a standard system. Future work will be dedicated to further refinement of performance (especially in memory usage), stability and more documentation. The main WWW page is on the URL below:
`http://winnie.kuis.kyoto-u.ac.jp/pub/julius/`

7. Acknowledgment

Part of the work is sponsored by CREST (Core Research for Evolutional Science and Technology), Japan.

8. References

[1] P.R. Clarkson and R. Rosenfeld: Statistical Language Modeling Using the CMU-Cambridge Toolkit, In *Proc. of ESCA Eurospeech'97*, vol.5, pages 2707–2710, 1997.

[2] S.Young, J.Jansen, J.Odell, D.Ollason and P.Woodland: The HTK Book, In Entropic Cambridge Research Lab., 1995.

[3] `http://www.lang.astem.or.jp/CSRC/`

[4] T.Kawahara, A.Lee, T.Kobayashi, K.Takeda, N.Minematsu, S.Sagayama, K.Itou, A.Ito, M.Yamamoto, A.Yamada, T.Utsuro, and K.Shikano: Free Software Toolkit for Japanese Large Vocabulary Continuous Speech Recognition, In *Proc. ICSLP*, Vol.4, pages 476–479, 2000.

[5] A.Lee, T.Kawahara and S.Doshita: An Efficient Two-Pass Search Algorithm using Word Trellis Index, In *Proc. ICSLP*, pages 1831–1834, 1998.

[6] A.Lee, T.Kawahara, K.Takeda and K.Shikano: A New Phonetic Tied-Mixture Model for Efficient Decoding, In *Proc. IEEE-ICASSP*, pages 1269–1272, 2000.

[7] A.Lee, T.Kawahara and K.Shikano: Gaussian Mixture Selection using Context-independent HMM, In *Proc. IEEE-ICASSP*, 2001.