SPIDER: A Continuous Speech Light Decoder

Abdelaziz A.Abdelhamid, Waleed H.Abdulla, and Bruce A.MacDonald
Department of Electrical and Computer Engineering, Auckland University, New Zealand
E-mail: aabd127@aucklanduni.ac.nz, {w.abdulla, b.mcdonald}@auckland.ac.nz

***Abstract*—In this paper, we propose a speech decoder, called SPeech lIght decoDER (SPIDER), for extracting the best decoding hypothesis from a search space constructed using weighted finite-state transducers. Despite existence of many speech decoders, these decoders are quite complicated as they take into consideration many design goals, such as extraction of N-best decoding hypotheses and generation of lattices. This makes it difficult to learn these decoders and test new ideas in speech recognition that often require decoder modification. Therefore, we propose in this paper a simple decoder supporting the primitive functions required for achieving real-time speech recognition with state-of-the-art recognition performance. This decoder can be viewed as a seed for further improvements and addition of new functionalities. Experimental results show that the performance of the proposed decoder is quite promising when compared with two other speech decoders, namely HDecode and Sphinx3.**

I. Introduction

Currently existing speech decoders are quite large and complex, and often require some libraries to be defined as a prerequisite to be able to perform speech recognition experiments [1], [2], [3]. Also, it is usually required to modify the speech decoder to test new ideas or to add new functionalities. However, this task takes a large amount of time to understand where to modify. One possible solution to this problem is presented in this chapter through the development of a simple baseline speech decoder supporting the primitive functions necessary for speech recognition. By developing such a decoder, new additions and modifications can be built on top of these primitive functions.

In [4], [5], the authors presented another solution based on functional programming. In that research, the low level procedures are simplified using higher level functions that can be simply called to perform speech decoding. Despite the simplicity of that approach, the real-time factor is still high and a huge memory storage is required. Also, that approach presents speech recognition functionalities as black boxes, which might become a barrier for extending that approach to include more functionalities without the need to do low level modifications.

Speech decoders are usually classified into two broad classes [6]. Speech decoders of the first class are based on static recognition network in which speech knowledge sources, namely lexical models (in the form of a pronunciation dictionary), acoustic models (in the form of hidden Markov models (HMMs)) and language models (in the form of statistical n-gram models), are integrated together in one search space. This integration is commonly performed using weighted finite state transducers (WFSTs) before starting speech decoding [7], [8], [9]. The advantage of this class of decoders is the fast decoding speed and the simple realization. Whereas, speech decoders of the second class are based on a dynamically expanded search space, in which speech knowledge sources are integrated during speech decoding. In other words, lexical and acoustic models are statically integrated and language models are handled explicitly during the search process. The advantage of this latter class of decoders is the requirement of a small amount of decoding memory. However, the decoders in this class are sophisticated and there are no general optimization techniques. Nevertheless, a common characteristic between these two classes of decoders is the search strategy which can take the form of either time synchronous or time asynchronous strategy. In both strategies, only the most promising paths are kept active, resulting in a so called beam search. The most common method for implementing this strategy is the token passing mechanism [10]. In this thesis, we adopted the static construction of the search space using WFST along with time synchronous beam search strategy in the design of SPIDER. There are several reasons for selecting this approach, such as it's conceptual simplicity and the possibility to address several advanced problems, such as early language model pruning and utilization of cross-word tri-phone contexts.

In [1], authors provided great efforts in the development of a useful toolkit supporting several speech processing functions, such as feature extraction, parameter learning, speech decoding, and more. Also, this toolkit supports language models in the form of regular grammar and n-gram models. However, the decoding process is based on dynamic expansion approach which might negatively affect the overall recognition performance. In [2], a similar speech decoding engine is presented. That engine is also based on dynamic expansion of the search space. Despite the improvement achieved by this engine in the recognition performance, it still quite complex to modify or add new functionalities to this engine due to handling speech knowledge sources at runtime. Another speech decoding engine is presented in [3] which has a small footprint when compared to the previous decoding engines. However, this engine is more suitable for small tasks. Also, this engine does not support WFST which recently becomes a standard approach in building speech decoders. On the other hand, authors in [11] presented a WFST-based speech decoder. However, to the best of our knowledge, this decoder does not support Windows operating system, as it is designed mainly for Unix machines. Nevertheless, it requires a set of helping toolkits, such as Torch and Tracter, which makes it a non-trivial task to get it works. Also, this decoder does not support

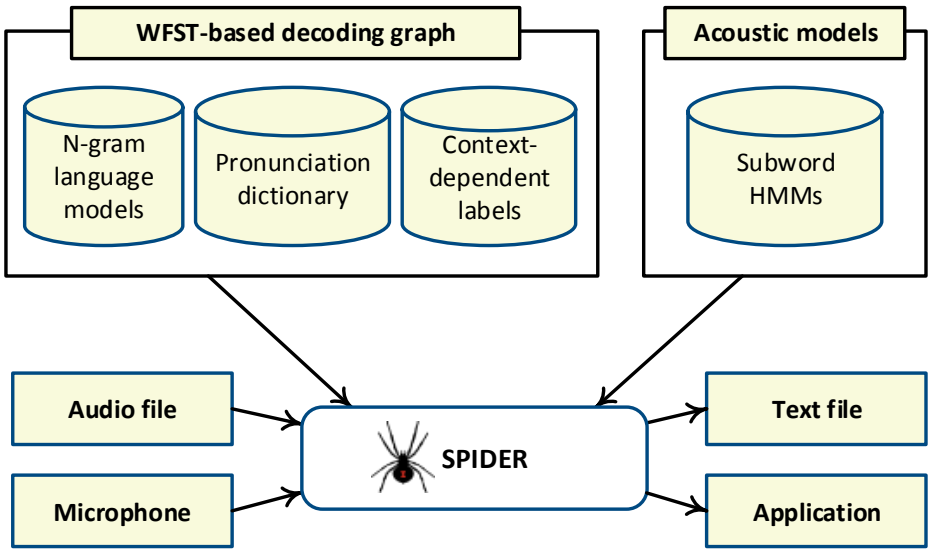

Fig. 1. Overview of SPIDER.

live speech decoding. In addition, for some advances functions, such as discriminative training on WFST decoding graphs, it is necessary to get HMM state level forced alignments. However, this level of forced alignment is not supported by this decoder.

Although the design of SPIDER is similar to that of the above decoders, the data structure and decoding mechanism of SPIDER are provided in a simple and compact form. Therefore, SPIDER can be viewed as a seed for future improvements and addition of new functionalities.

The organization of this paper comes as follows. An overview of the developed speech decoder along with the currently supported functions are described in Section II. The internal structure of SPIDER is presented and discussed in Section III. Experimental conditions are then described in Section IV. Finally, the conclusions and future work are given in Section V.

II. Overview

An overview of SPIDER system is depicted in Fig. 1. Given a WFST-based decoding graph and a set of acoustic models, SPIDER functions as a speech recognition system of the given task. SPIDER supports processing of both audio files and a live audio stream. For the file input, SPIDER assumes one sentence utterance per input file. The output from SPIDER can be either written to a file or transmitted to an application. The WFST-based decoding graph supported by SPIDER incorporates lexical (pronunciation) models, context-dependent labels and language models. Whereas, acoustic models are treated separately in the decoding process.

A. Summary of Features

SPIDER is fully implemented in C++ using multi-threads to handle live speech capturing as well as speech decoding. The current version supports the following features.

Functions:

- Live audio input recognition via microphone.
- Output 1-best decoding hypothesis.
- Forced alignment in word, phone or HMM-state level.

Supported models:

- WFST decoding graph generated using OpenFST [12].
- Tri-phone HMM with tied-mixtures in HTK format [1].
- Mel-frequency cepstral coefficients in HTK format [1].

Performance:

- Real time recognition when tested on the Resource Management (RM1) command and control corpus.
- A footprint of 356MB for WFST-based search space (using 20k words and tri-gram language models [13]) and 85.4MB for tri-phone acoustic models represented by HMMs (8000 states, 16 mixtures/state [13]).
- No hard-coded or machine-specific optimization.

III. Inside SPIDER

Speech decoding is the process of searching for the best word sequence, $\hat{W} = w_1 w_2 ... w_m$, that has the maximum posterior probability, $P(W|O)$, for an input acoustic observation sequence, $O = O_1, O_2, ..., O_T$. This process is described based on Bayes' theorem as follows.

$$\hat{W} = \underset{\forall W \in L}{arg\,max}\ P(O|W)P(W)^{LMSF} WIP^{N(W)} \quad (1)$$

where $P(O|W)$ and $P(W)$ are the acoustic and language model probabilities, respectively. The language model scaling factor (LMSF) is an arbitrary constant used to compensate the scale difference between language and acoustic probabilities. Word insertion penalty (WIP) is another arbitrary constant

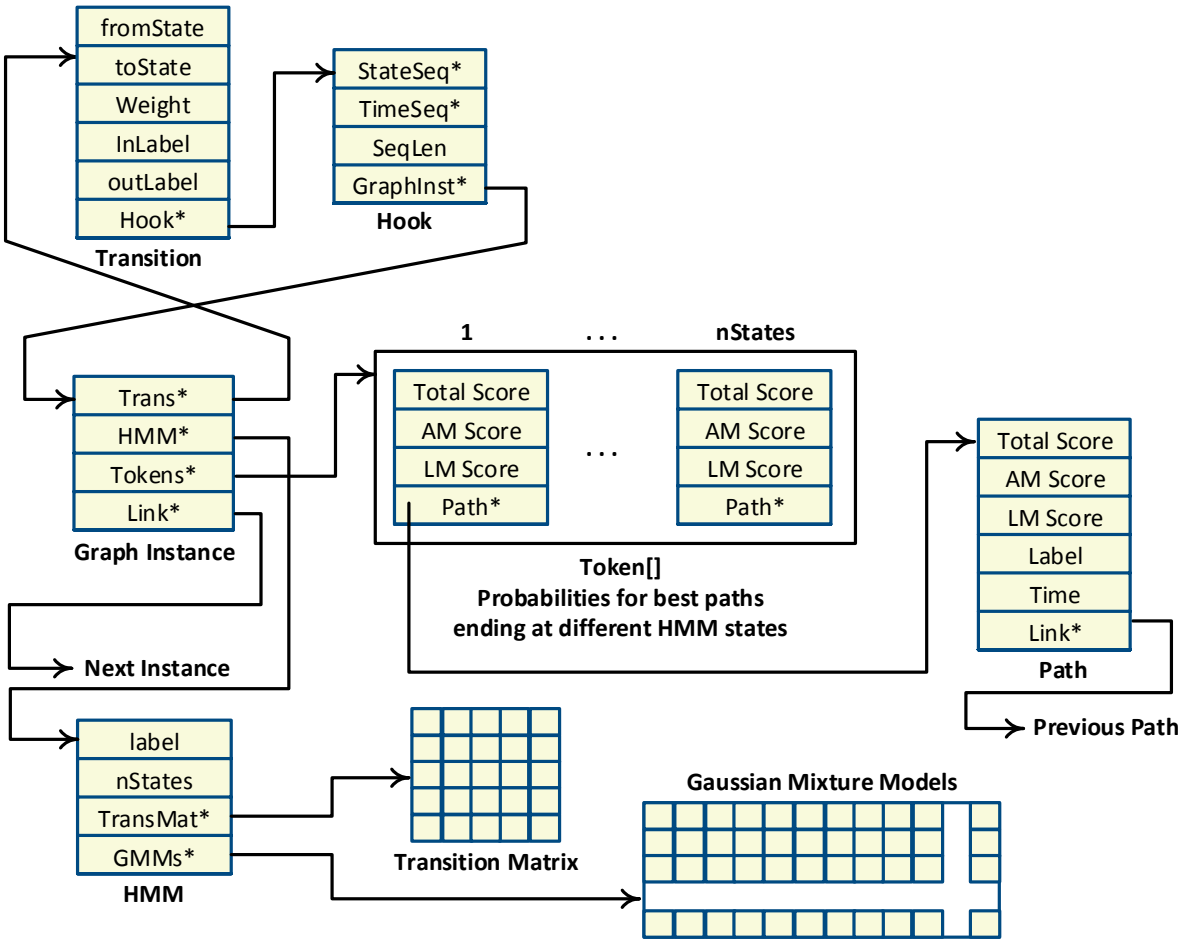

Fig. 2. Data structure incorporated in the implementation of SPIDER.

used to avoid preferring short words over longer words. The value of WIP is usually controlled by the number of words, $N(W)$, of the decoding hypothesis, W. And, L refers to the search space within the limits of a beam width. To find the best word sequence, $\hat{W}$, one obvious way is to search all possible word sequences, and select the one with best posterior probability. However, this way is practically infeasible, especially when the vocabulary contains large number of words. An alternative way is to reduce the search space to make this process more feasible. There are several powerful design principles can be used to reduce the search space such beam pruning [6]. These methods enable speech decoding to be performed with even larger vocabularies and slower machines at near real-time performance. Usually, speeding up the search process does not come without a cost, the decoding accuracy has to be well considered. With a good design, both speed and decoding accuracy can be well combined.

A. Data structure

Speech decoders rely mainly on three types of knowledge sources, namely lexical, acoustic and language models. In SPIDER, only lexical and language models along with context-dependent labels are integrated together using a sequence of WFST operations [14], [15], [16]. However, the acoustic models are not included in the integration of knowledge sources to reduce the size of the resulting WFST-based search space. Before running the speech decoder, both search space and acoustic models are loaded in memory. The transitions of the search space are loaded using instances from a class called *Transition*. Whereas, the acoustic models are loaded using instances from the HMM class. The main data structure incorporated in SPIDER is depicted in Fig. 2. The main component of this data structure is called *Graph Instance*, which is responsible of full tracking of the decoding process. Each instance is linked with other instances from the WFST transition and the HMM classes along with an array to hold the tokens passing over the HMM states. The decoding path is preserved using a class called *Path*. Each instance of this class contains total score, acoustic model score, language model score, the decoded label, frame/time and a link to the previously decoded labels. In addition, the *Hook* class is used for HMM state level forced alignments.

B. Decoding algorithm

The decoding strategy in current state-of-the-art automatic speech recognition (ASR) systems is based on dynamic pro-

gramming concept [17], [18] in searching for the most likely word sequence given acoustic, language models constraints, and an input signal. The control structure required to perform this search is conceptually simple, but extremely hard to implement efficiently in software. It involves extensive data structure manipulations, particularly in the case of tri-gram language models and cross-word context-dependent phone models. Consequently, handling the speech decoding process is considered as the most time-consuming part in the design of speech recognition systems. Fortunately, using WFST-based search space, this process can be simply and efficiently realized using Viterbi beam search algorithm [6]. In the following, the Viterbi algorithm and its implementation using token passing mechanism are discussed.

Algorithm 1: Speech decoding algorithm of SPIDER.

Input: A set feature vectors (**Features**) of length T.
Output: 1-best decoding hypothesis.

begin
  $\hat{Q}_1 \leftarrow Q_{words\_start\_state}$
  $token_{q,1} \leftarrow nullToken$ **for all states** q

  for each $o_{t=1:T} \in Features$ **do**
    for each $q \in \hat{Q}_t$ **do**
      $Max\_score \leftarrow max(score(token_{q,t}))$
      $Cache(q) \leftarrow null$
    end for
    $\hat{Q}_{t+1} \leftarrow \{\}$

    for each $q \in \hat{Q}_t$ **do**
      $score(token_{q,t}) \leftarrow score(token_{q,t}) - Max\_score$
      if $score(token_{q,t}) > Prun\_thr$ **then**
        if $(Log\_emis\_prob \leftarrow Cache(q)) = null$ **then**
          $Log\_emis\_prob \leftarrow Calculate\_emission\_prob(o_t, q)$
          $Cache(q) \leftarrow Log\_emis\_prob$
        end if
        $S \leftarrow redirectToken(log\_emis\_prob, q, t)$
        $\hat{Q}_{t+1} \leftarrow \hat{Q}_{t+1} \cup S$
      end if
    end for
  end for
  $\wp \leftarrow \hat{Q}_{T+1} \cap Q_{words\_end\_state}$
  $best\_hypothesis \leftarrow \underset{q \in \wp}{argmax}(score(token_{q,T+1}))$
end

Fig. 3. Speech decoding steps employed in SPIDER.

1) Viterbi beam search: Viterbi search is inherited from a broader class of search techniques referred as breadth-first. In the Viterbi search algorithm, all hypotheses are processed in parallel and pruned gradually to find out the correct hypothesis

Algorithm 2: $redirectToken(Log\_emis\_prob, q, t)$

Input: Log-emission probability, Log_emis_prob, an HMM state, q, and frame index, t.
Output: A set, S, containing the succeeding HMM states.

begin
  $S \leftarrow \{\}$
  for each q_suc **succeeds** q **do**
    $new\_score \leftarrow score(token_{q,t}) + Log\_emis\_prob + trans\_weight(q, q\_suc)$
    if $new\_score > score(token_{q\_suc,t+1})$ **then**
      $score(token_{q\_suc,t+1}) \leftarrow new\_score$
      $path(token_{q\_suc,t+1}) \leftarrow path(token_{q,t})$
      $S \leftarrow S \cup \{q\_suc\}$
    end if
  end for
  return S
end

Fig. 4. Token passing mechanism used in SPIDER.

with maximum score. Therefore, the speech decoding process can be viewed as a recursive process over the transitions of a decoding graph. At any stage of the search process, all the generated partial hypotheses terminate at the same point in time, and thus this search technique is described as a time-synchronous search. All partial hypotheses can be compared to each other as they correspond to the same fragment of the input speech utterance. Practically, the complete Viterbi search is impossible, since the search space becomes very large even in moderate-sized tasks. The beam pruning approach is usually used to reduce the search space, in this case, the algorithm is called Viterbi beam search. In the latter approach, all hypotheses falling outside the beam are punned away, and only the hypotheses whose likelihood falls within the beam are kept for further growth [19], [20], [21]. The estimation of the beam width is usually determined empirically.

As this approach is implemented by the token passing method, a data structure is called *token* is defined to hold log-probability of best path as well as a reference to the path history. This token is passed over each HMM state of the search space as shown in Fig 5. Initially, the token is assigned to the first state of all candidate HMMs. Then, at each time instant, the token is copied along all the possible transitions, and the probability is updated for each transition. At each state, all the received copies of the token are sorted, and only the best token is stored. Also, the information about the states through which the tokens have been passed are stored at each token. While propagating the tokens through the decoding graph, the likelihood of a token is compared to the beam width, and only the tokens with probabilities within the beam width are kept alive, otherwise, the token is pruned away.

a) Description of algorithm: The general outlines of the speech decoding process are presented in the algorithm shown in Fig. 3. The input to the decoding process is a sequence

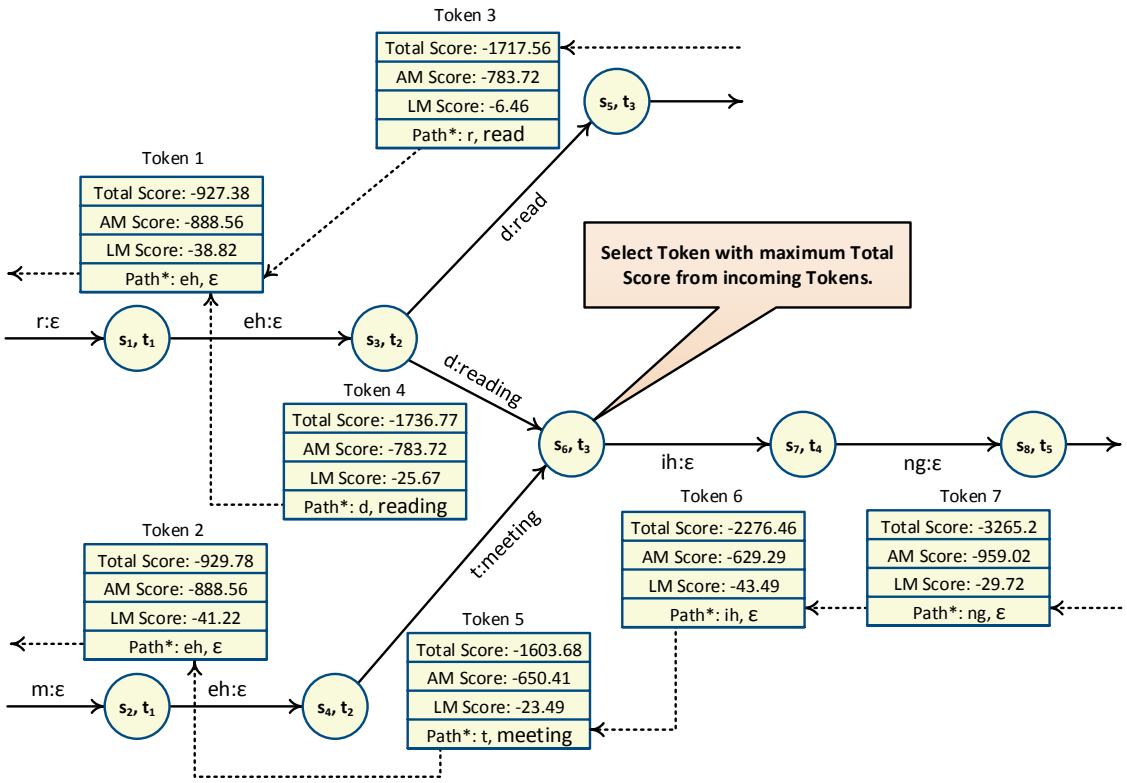

Fig. 5. Procedure of token passing with sample acoustic (AM Score) and language (LM Score) likelihoods. Total Score refers to the accumulated sum of the acoustic and language scores. Path* refers to a backpointer to the previous tokens generated in the decoding process. Dotted arrows refer to backpointer links.

of feature vectors in the form of MFCCs, and the output is the best decoding hypothesis. Initially, the start states of all candidate HMMs, Line 2, are assigned a copy of a null token, Line 3, where all scores are reset. The set of states of the candidate HMMs is denoted by $Q_{word-start}$. Then, a loop is iterated over all the feature vectors, Line 5, to decode each vector. At each iteration of this loop, the maximum score is retrieved from all active tokens, Line 7, to be used to normalize the scores of the active tokens, and the cache, which holds the emission probabilities, is reset, Line 8. Another loop is iterated over all the active tokens at time t, Line 12, to calculate the acoustic and language model probabilities. Inside this loop, the scores of the active tokens are normalized, Line 13. Afterwards, the normalized score is compared with a pruning threshold, Line 14. The value of this threshold is determined empirically, and in our implementation it takes a value in the range from -100 to -300. If the score of the active token is greater than the pruning threshold, Line 14, the emission probability is calculated if it is not stored in the cache, Lines 15-18. Then, the token is redirected to the next state in the search space using a function called *redirectToken*, Line 19. This function returns the new reached states to be processed in the next iteration, Line 20. After processing all the acoustic features, the best decoding hypothesis is retrieved from the last survived token as the token with best total score, Lines 24-25.

Figure 4 illustrates the token passing mechanism. In this figure, for each HMM state, q, an iterative loop is performed on all its succeeding states, Lines 4-11. A new_score is calculated for each succeeding state, Line 5. The calculation of this score is performed in terms of transition weight, which can be either a WFST transition weight for cross-HMM transitions, or an HMM probability, for within-HMM transitions. Using this new_score, the score at state q_suc will be updated only if the new_score is greater than the score of the succeeding state, q_suc, Line 7. Also, the path at state, q, is assigned to the path of the token at the succeeding state, q_suc, Line 8.

b) Emission probability calculation: In current state-of-the-art ASR systems, the emission probability distribution is represented by Gaussian mixture models (GMMs). The function used in calculating the emission probability at an HMM state, j, is defined in terms of a set of Gaussian mixtures as follows.

$$
\begin{aligned}
b_j(o_t) &= \sum_{m=1}^{M} b_{jm}(o_t) = \sum_{m=1}^{M} c_{jm}\mathbb{N}(o_t, \mu_{jm}, \sigma_{jm}) \\
&= \sum_{m=1}^{M} c_{jm} \frac{1}{(2\pi)^{\frac{D}{2}}|\sigma_{jm}|^{\frac{1}{2}}} \\
&\quad exp\left[-\frac{1}{2}(o_t - \mu_{jm})^T \sigma_{jm}^{-1}(o_t - \mu_{jm})\right]
\end{aligned}
\tag{2}
$$

where o_t is an acoustic observation vector, $b_{jm}(o_t)$ is the observation distribution function of the m^{th} Gaussian mixture which is weighted by c_{jm}, and M is the number of mixtures per HMM state. Also, μ_{jm} and σ_{jm} refer to the mean and variance vectors of the Gaussian mixture, respectively. As the coefficients of the acoustic features are assumed independent, the matrix, σ_{jm}, is assumed diagonal. Also, as the multiplication of probabilities becomes addition in the logarithmic domain, it will be more convenient to express $b_{jm}(o_t)$ and $b_j(o_t)$ in their logarithmic forms. Thus, the logarithm of a weighted mixture is defined as follows.

$$log_e\ b_{jm}(o_t) = C_{jm} + g_{jm} + \sum_{d=1}^{D} \left(o_t^{(d)} - \mu_{jm}^{(d)}\right)^2 v_{jm}^{(d)} \quad (3)$$

where $\mu_{jm}^{(d)}$ is the d^{th} dimension of the mean vector, μ_{jm}. The constants C_{jm}, $g_{jm}^{(d)}$ and $v_{jm}^{(d)}$ are defined as follow.

$$C_{jm} = log_e\ c_{jm} \quad (4)$$

$$g_{jm} = -\frac{1}{2}\left(D\ log_e\ (2\pi) + \sum_{d=1}^{D} log_e\ \left(\sigma_{jm}^{(d)}\right)^2\right) \quad (5)$$

$$v_{jm}^{(d)} = \frac{-1}{2\left(\sigma_{jm}^{(d)}\right)^2} \quad (6)$$

where $\left(\sigma_{jm}^{(d)}\right)^2$ is the d^{th} element of the diagonal co-variance matrix. Using recursively procedure, the natural log of observation distribution probability, $log_e\ b_j(o_t)$, can be estimated follows.

$$\begin{aligned} log_e\ b_j(o_t) = (((&log_e\ b_{j1}(o_t) \oplus log_e\ b_{j2}(o_t)) \\ &\oplus log_e\ b_{j3}(o_t)) \oplus ... \oplus log_e\ b_{jM}(o_t)) \end{aligned} \quad (7)$$

where $\oplus$ refers to the log-add operator which is defined as follows.

$$x \oplus y = log_e\ (e^x + e^y) \quad (8)$$

C. Design for fast decoding

From the HRI perspective, fast speech decoding is an essential requirement to make the interaction with robots as natural as possible. In addition, fast speech decoding is required in parameter learning especially when a large training corpus is employed. Therefore, we developed two techniques for performing this task. The first technique is the use of caching for fast likelihood access, whereas the second is the implementation of histogram beam pruning along with conventional state-level beam pruning. These techniques are discussed in the following sections.

1) Likelihood access and caching: To speed up the speech decoding process, we have developed a method which reduces the time required for recalculating the acoustic likelihoods, especially when the Gaussian mixtures are shared among several HMMs. In this method, we used a global cache to store the acoustic probabilities to be used by decoding hypotheses. At each point of the decoding process, the likelihood is calculated only once by the first decoding hypothesis uses the Gaussian mixture, then the other decoding hypotheses sharing the same Gaussian mixture can access the calculated likelihood from the cache. This process is depicted in Fig. 6.

2) Beam pruning: The decoding mechanism incorporated in the developed decoder is based on the token passing algorithm [10]. As discussed earlier in this chapter, the token passing algorithm is based on a key object called token. This object is viewed as an information holder carrying the acoustic, language and accumulated scores, as well as a reference to the previous tokens along the path created while decoding a speech utterance. To make the search more efficient, at each point in the search process, only the most promising tokens are kept alive for further navigation of the search space. This process is usually realized by the means of pruning techniques. Two popular pruning techniques are described in the following.

- Global beam pruning: In which, the tokens carrying an accumulated score above a certain pruning threshold are kept alive, while other tokens are eliminated. This technique is based on a pruning threshold that should be estimated carefully to reach a high decoding accuracy with a small real-time factor.
- Histogram pruning: In this technique, only a certain number of tokens are kept alive at each frame, while other tokens are eliminated. This technique is based on a number of tokens threshold which also should be determined carefully from the training corpus.

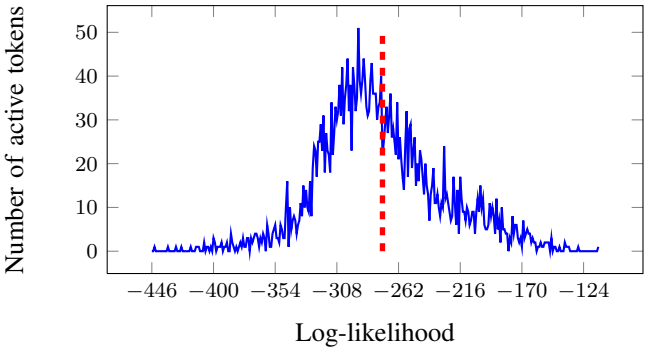

Fig. 7. Histogram of active tokens after processing a speech frame. The red line points to the log-likelihood at which the active tokens are compared to determine the tokens that will be kept alive. Tokens carrying log-likelihoods on the right-hand-side of the red line are kept alive, whereas tokens carrying log-likelihoods on the left-hand-side of the red line are eliminated.

In our implementation, we employed both beam and histogram pruning to achieve a fast speech decoding through using beam pruning when the number of active tokens is less than the maximum number of tokens threshold and using

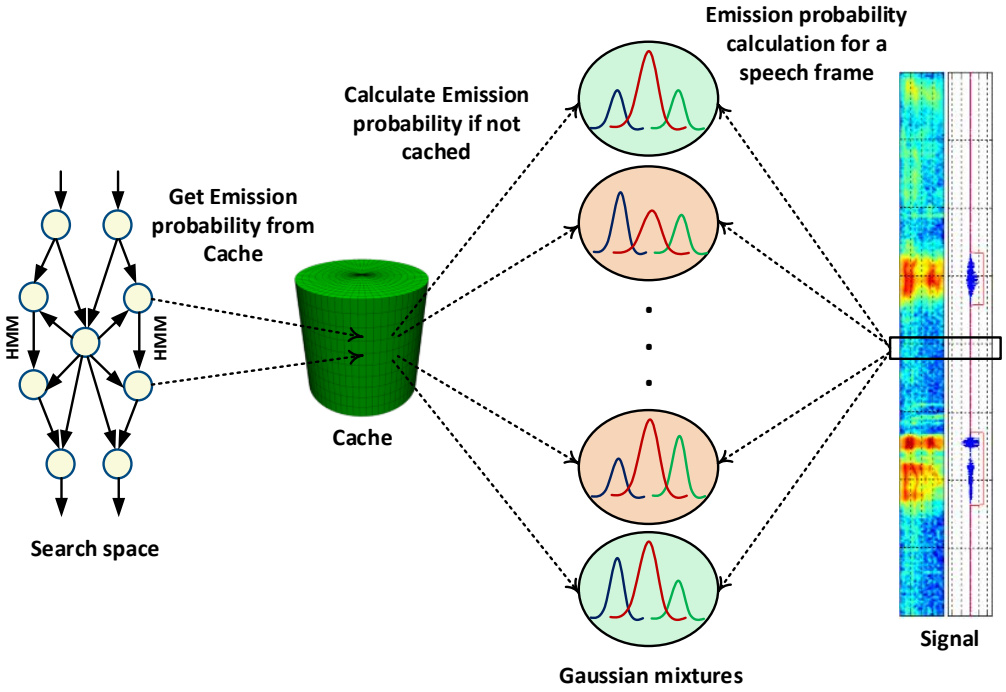

Fig. 6. Emission probability calculation using caching. For clarity, each mixture contains 3 Gaussian distributions, but practically each mixture contains 39 Gaussian distributions. Also, in our implementation, number of mixtures per HMM state is 32.

histogram pruning otherwise. An example on the histogram of active tokens after processing a speech frame is shown in Fig. 7. In this figure, the number of active tokens carrying an accumulated log-likelihood is plotted for a range of log-likelihoods from -446 to -113. When taking the value of the maximum number of active tokens as 2000, we can this number is reached at the log-likelihood of -274, at which the red line is plotted. Consequently, the tokens carrying log-likelihoods above -274 (the right-hand-side of the red line) will be kept alive for further navigation of the search space, whereas, the tokens carrying log-likelihoods below -274 (the left-hand-side of the red line) will be eliminated. Speech decoding using SPIDER keeps only the 2K tokens at each time frame in case of having more than 2K active tokens. However, if the number of active tokens is less than 2K tokens, a global beam is used where its value is taken in the range from -100 to -250.

D. The full picture

The full picture of the speech decoding process is depicted in Fig. 8. In this figure, two structures are loaded into memory, 1) search space which is represented by an integrated WFST-based decoding graph, and 2) context-dependent models represented by HMMs. After loading these data structures, speech decoding process starts decoding the incoming acoustic features. The token passing mechanism starts navigating the search space looking for the best word sequence. As it is surfing the search space, if it passes through a transition carrying context-dependent label, the corresponding acoustic model is retrieved from memory to calculate the probability of the acoustic feature with respect to that acoustic model if this probability does not exist in the cache. Whereas, if it passes through a transition carrying a language model probability, this probability will be added to the accumulated score of the token at this point of the search process.

IV. Experiments

For comparison purposes, two speech decoders were included in the conducted experiments. The first decoder is HDecode, which is developed by Cambridge University [1]. The other decoder is Sphinx3, which is developed by Carnegie Mellon University [2]. These decoders were tested in terms of a full decoding graph. In all experiments, the LMSF and WIP were set 13 and 0, respectively. All the experiments were conducted on a machine having a CPU running at 2160 MHz with 2GB of memory. The current supporting operating system is WindowsXP.

A. Speech corpus

The speech database incorporated in this research is the resource management (RM1) command and control speech corpus [22]. The test set of this corpus consists of 1200 commands based on 1000 words vocabulary. The speech utterances are recorded by 80 speakers. All the spoken commands are sampled at 16kHz and 16 bits quantization. The MFCC feature vectors extracted from each spoken command consist of 39

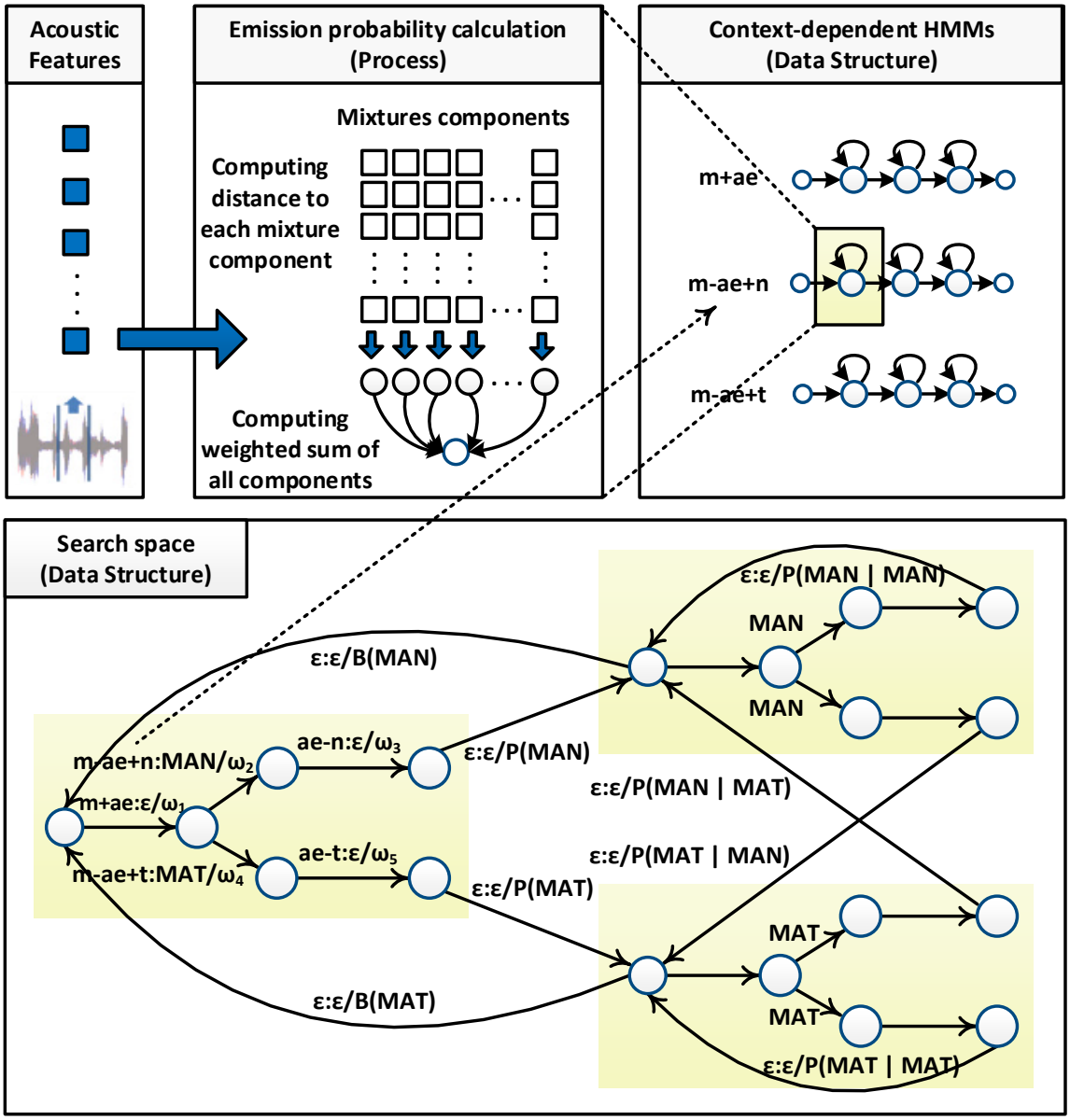

Fig. 8. The full picture of speech decoding using SPIDER. Dotted arrow refers to accessing corresponding acoustic model.

dimensions (12 static, 1 energy, 13 delta and 13 acceleration) for each vector.

B. Acoustic and language models

The experiments conducted in this research were based on the freely available acoustic and language models provided in [13]. The acoustic models are represented by a set of HMMs based on 8000 states and 32 Gaussian mixtures per state. Also, the language models are based on tri-gram models consisting of 19998 uni-grams, 492269 bi-gram and 247647 tri-grams.

C. System accuracy

The three speech decoders were tested on the RM1 command and control corpus. The recognition accuracy is shown in Fig. 9. In this figure, the recognition accuracy of the developed decoder outperforms that of HDecode and Sphinx3. This can be interpreted as the proposed system is based on WFST where the transition weights are distributed in a way that better guides the token passing mechanism to the best decoding hypothesis. The recognition results shown in Fig. 9 are recorded in terms of different beam widths for all the speech decoders. The system accuracy is defined as follows.

$$Word\ accuracy\ rate = \frac{n-s-d-i}{n} \times 100\% \qquad (9)$$

where n is the total number of words. s is the number of word substitutions (incorrectly recognized words). d and i are the

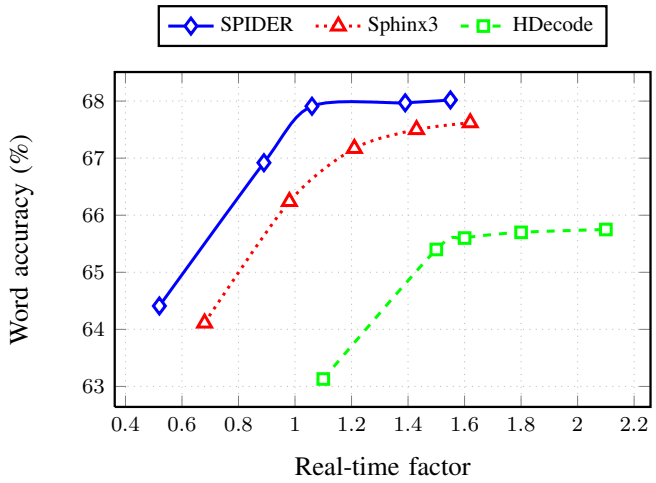

Fig. 9. Decoding performance of three speech decoders (HDecode, Sphinx3, and SPIDER) in terms the RM1 evaluation set.

numbers of word deletions and word insertions, respectively.

D. Timing profile

A set of experiments were conducted to examine the timing performance using the RM1 command and control corpus. In these experiments, the entire test set of the RM1 corpus was tested using the three speech decoders. The real-time factor of each speech utterance was recorded. The left-hand-side figures of Fig. 10 show the timing profile of the three decoders. From this figure, it can be seen that the real-time factor of most of the speech utterances is below 1 when using Sphinx3 and SPIDER, that indicates the similar response time of the developed system with the well-known speech decoder, Sphinx3. Also, the response time of the developed system outperforms that of HDecode system. The average real-time factor of SPIDER reached is 0.88, whereas it reached 0.92 and 1.4 when using Sphinx3 and HDecode, respectively. Another criterion used to further analyse the experimental results is called time delay. The criteria used to measure real-time factor and time delay are defined as follows.

$$Real-time\ factor = \frac{Decoding\ time}{Speech\ duration} \quad (10)$$

$$Time\ delay = Decoding\ time - Speech\ duration \quad (11)$$

When the value of this time-delay criterion is positive, it means that the decoding time is longer than the speech duration. This values gives us an indication about the amount of time needed the used needs to wait until the system recognizes the spoken utterance. Practically, this value of the time delay should be closer to or less than zero to make a fast a natural interaction speed with the decoding engine. However, the negative value of this criterion indicates that the decoding time is shorter than the speech duration, which is highly appreciated in developing real-time speech recognition systems, as it suggests that the spoken command can be recognized instantly after the speech is uttered. This criterion was measured for all the three speech decoders and the results are shown in the right-hand-side figures of Fig. 10. In this figure, we can notice that both Sphinx3 and SPIDER have a negative time delay, which means that both of them can be used in practical and real-time large scale applications.

V. Conclusion

In this paper, we presented a simple speech decoder, called SPIDER. This decoder is fully implemented in C++ and supports the primitive functions necessary for achieving state-of-the-art recognition performance, which can be further extended to include more complex functionalities. Experimental results show that the proposed decoder achieves state-of-the-art recognition performance when compared with two common speech decoder, namely HDecode and Sphinx3. Future plan includes improving the computational efficiency and extending the system to incorporate model training and adaptation.

VI. Acknowledgement

This work is supported by the R&D program of the Korea Ministry of Knowledge and Economy (MKE) and Korea Evaluation Institute of Industrial Technology (KEIT). [KI001836, Development of Mediated Interface Technology for HRI].

References

[1] S. Young, G. Evermann, M. Gales, T. Hain, D. Kershaw, X. Liu, G. Moore, J. Odell, D. Ollason, D. Povey, V. Valtchev, and P. Woodland, *The HTK book*. Cambridge University, 2009.

[2] D. Huggins, M. Kumar, A. Chan, A. Black, M. Ravishankar, and A. Rudnicky, "PocketSphinx: A free, real-time continuous speech recognition system for handheld devices," in *Proceedings of International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Toulouse, May 2006, pp. 185–188.

[3] A. Lee and T. Kawahara, "Recent development of open-source speech recognition engine Julius," in *Proceedings of Asia-Pacific Signal and Information Processing Association (APSIPA)*, 2009, pp. 131–137.

[4] T. Shinozaki, M. Sekijima, S. Hagihara, and S. Furui, "Compact speech decoder based on pure functional programming," in *Proceedings of Asia-Pacific Signal and Information Processing Association (APSIPA)*, 2011.

[5] T. Shinozaki, S. Furui, Y. Horiuchi, and S. Kuroiwa, "Pipeline decomposition of speech decoders and their implementation based on delayed evaluation," in *Proceedings of Asia-Pacific Signal and Information Processing Association (APSIPA)*, 2012.

[6] X. Aubert, "An overview of decoding techniques for large vocabulary continuous speech recognition," *Transactions on Computer Speech and Language*, vol. 16, pp. 89–114, 2002.

[7] M. Mohri, "On some applications of finite-state automata theory to natural language processingy," *Transactions on Natural Language Engineering*, vol. 2, no. 1, pp. 61–80, 1996.

[8] ——, "Finite state transducers in language and speech processing," *Transactions on Computational Linguistics*, vol. 23, pp. 269–311, 1997.

[9] M. Mohri and M. Riley, "Network optimizations for large vocabulary speech recognition," *Transactions on Speech Communication*, vol. 28, pp. 1–12, 1998.

[10] S. Young, N. Russel, and J. Thornton, "Token passing: A simple conceptual model for connected speech recognition systems," Tech. Rep., 1989.

[11] D. Moore, J. Dines, M. Magimai, J. Vepa, O. Cheng, and T. Hain, "Juicer: A weighted finite-state transducer speech decoder," in *Proceedings of International Joint Workshop on Multimodal Interaction and Related Machine LEarning Algorithms*, 2006.

[12] C. Allauzen, M. Riley, J. Schalkwyk, W. Skut, and M. Mohri, "OpenFST: A general and efficient weighted finite-state transducer library," in *Proceedings of International Conference on Implementation and Application of Automata*, vol. 4783, Prague, Czech Republic, July 2007, pp. 11–23.

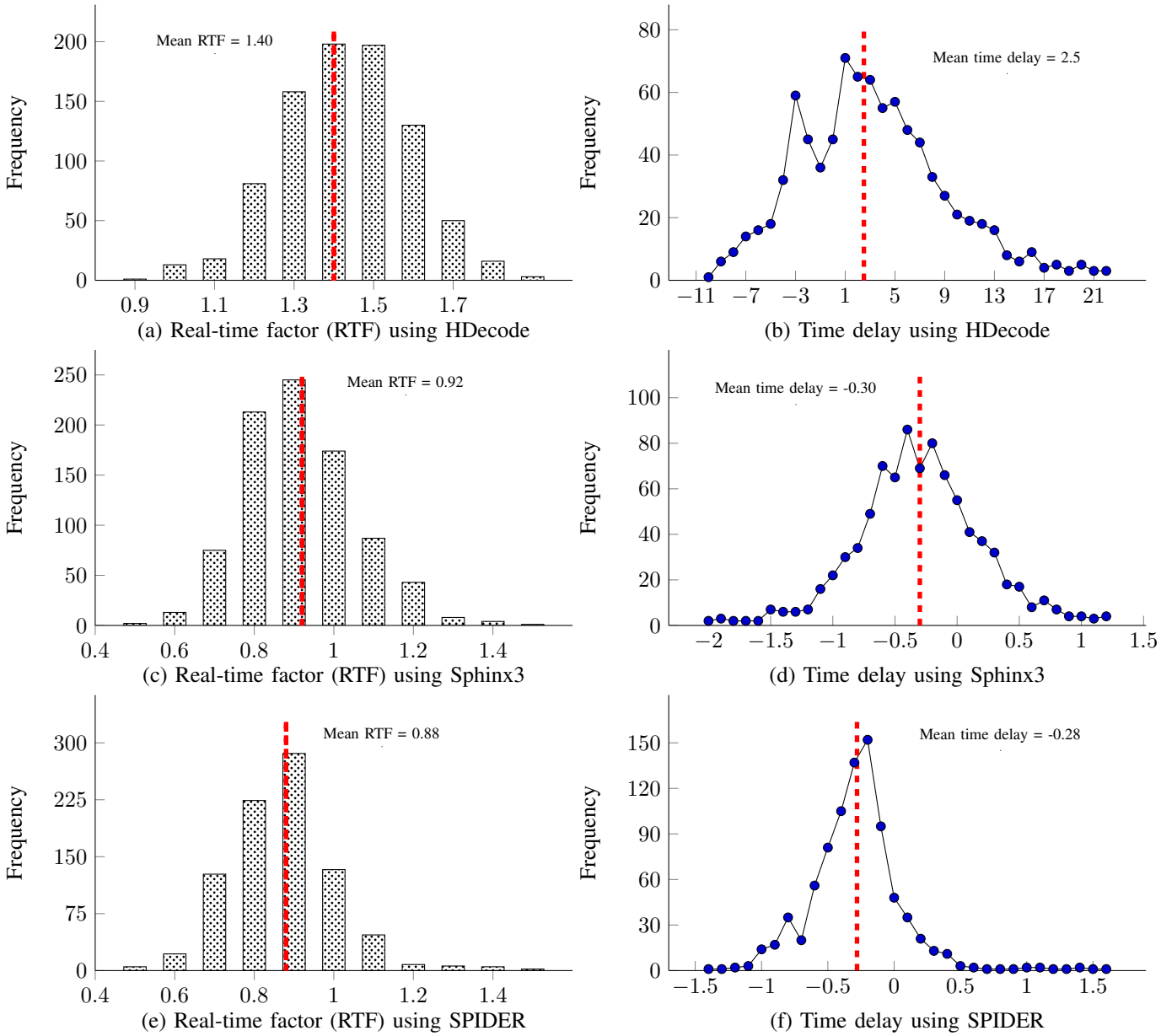

Fig. 10. Histograms of the speech decoding process of three decoders (HDecode, Sphinx3 and SPIDER) expressed in terms of the real-time factor on the left hand side figures and the time delay on the right hand side figures.

[13] K. Vertanen, "Baseline WSJ acoustic models for HTK and Sphinx: Training recipes and recognition experiments," Cambridge University, Tech. Rep., 2006.

[14] M. Mohri, F. Pereira, and M. Riley, "Weighted finite state transducers in speech recognition," *Transactions on Computer Speech and Language*, vol. 16, pp. 69–88, 2002.

[15] M. Mohri, "Weighted finite-state transducer algorithms: An overview," *Transactions on Formal Language and Applications*, vol. 148, pp. 551–564, 2004.

[16] J. Novak, N. Minemaysu, and K. Hirose, "Painless WFST cascade construction for LVCSR-Transducersaurus," in *Proceedings of International Speech Communication Association (InterSpeech)*, 2011, pp. 1537–1540.

[17] H. Ney and S. Ortmanns, "Dynamic programming search for continuous speech recognition," *IEEE Signal Processing Magazine*, vol. 16, no. 5, pp. 64–83, 1999.

[18] R. Bellman, *Dynamic Programming*. Princeton University Press, 1957.

[19] H. Ney, R. Umbach, B. Tran, and M. Oerder, "Improvements in beam search for 10000-word continuous speech recognition," in *Proceedings of International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, San Francisco, CA, USA, 1992, pp. 9–12.

[20] S. Ortmanns, A. Eiden, H. Ney, and N. Coenen, "Look-ahead techniques for fast beam search," in *Proceedings of International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 1997, pp. 1783–1786.

[21] V. Steinbiss, B. Tran, and H. Ney, "Improvements in beam search," in *Proceedings of International Conference on Spoken Language Processing*, 1994, pp. 2143–2146.

[22] P. Price, W. Fisher, J. Bernstein, and D. Pallett, "The DARPA 1000-word resource management database for continuous speech recognition," in *Proceedings of International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 1988, pp. 651–654.