Efficient WFST-Based One-Pass Decoding With On-The-Fly Hypothesis Rescoring in Extremely Large Vocabulary Continuous Speech Recognition

Takaaki Hori, *Member, IEEE*, Chiori Hori, *Member, IEEE*, Yasuhiro Minami, *Member, IEEE*, and Atsushi Nakamura, *Senior Member, IEEE*

***Abstract*—This paper proposes a novel one-pass search algorithm with on-the-fly composition of weighted finite-state transducers (WFSTs) for large-vocabulary continuous-speech recognition. In the standard search method with on-the-fly composition, two or more WFSTs are composed during decoding, and a Viterbi search is performed based on the composed search space. With this new method, a Viterbi search is performed based on the first of the two WFSTs. The second WFST is only used to rescore the hypotheses generated during the search. Since this rescoring is very efficient, the total amount of computation required by the new method is almost the same as when using only the first WFST. In a 65k-word vocabulary spontaneous lecture speech transcription task, our proposed method significantly outperformed the standard search method. Furthermore, our method was faster than decoding with a single fully composed and optimized WFST, where our method used only 38% of the memory required for decoding with the single WFST. Finally, we have achieved high-accuracy one-pass real-time speech recognition with an extremely large vocabulary of 1.8 million words.**

***Index Terms*—On-the-fly composition, speech recognition, weighted finite-state transducer (WFST).**

I. Introduction

In recent years, large-vocabulary continuous-speech recognition (LVCSR) systems have been incorporated in various speech applications, including dictation systems, spoken dialogue systems, and broadcast news captioning systems. The LVCSR decoding process finds a sequence of words that best matches an input signal from among a large number of hypotheses. Although the above applications work with current technologies, more efficient search algorithms are still needed for very large vocabulary tasks. For example, with the spoken-interactive open-domain question-answering system [1], a user asks a question that is a natural spoken sentence and its domain is not restricted, then the system finds the answer from a large corpus of news texts covering the last 12 years. Since the system cannot know what the user will ask in advance, the speech recognizer has to cover an extensive vocabulary. However, traditional approaches do not assume such an extremely large vocabulary. As a result, there is an enormous computational cost involved in finding the most likely hypothesis because many ambiguous hypotheses are generated during decoding.

The weighted finite-state transducer (WFST) provides a promising approach as an alternative to traditional decoding approaches [2]–[4]. The WFST offers a unified framework that can homogeneously represent various knowledge sources utilized in state-of-the-art LVCSR, e.g., hidden Markov models (HMMs), phonotactic networks, lexical descriptions, and n-gram language models. And, multiple WFSTs, each of which corresponds to one such knowledge source, can be integrated into a fully composed single WFST that organizes an all-in-one search network that represents the whole search space up to the deep details of the HMM-state level. Then, the single WFST is converted into an equivalent and more efficient WFST by optimizing operations, which disambiguate the search space and accelerate decoding.

This general framework is also applicable to other types of spoken language processing, such as text/speech translation [5], [6], and speech summarization [7]. In many applications, the WFST makes it possible to perform several types of processing with a one-pass search algorithm.

However, a fully composed WFST often becomes oversized because the all-in-one search network is composed of a multiplicative combination of states and transitions in component WFSTs. Although the resulting WFST does not always become very large ([2] reported that the size was 1.3 times larger than the language model for a 40k-word vocabulary), it can easily bloat depending on the specifications of the component WFSTs. In fact, when we use a very large lexicon, or a more detailed language model, we obtain a larger WFST (about two times larger than the language model) as a result of combining all the component WFSTs. This increases both the amount of computation and the memory used in decoding even if we optimize the WFST. Another problem with the fully composed WFST is that it offers less flexibility in regards to knowledge updates. For example, the lexicon and the language model need to be updated when we add new words or adapt the system to a new task/domain. However, once a single WFST is composed and

Manuscript received February 17, 2006; revised September 2, 2006. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Bill Byrne.

T. Hori is with NTT Communication Science Laboratories, NTT Corporation, 619-0237 Kyoto, Japan, and and also with the Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA 02139 USA (e-mail: hori@cslab.kecl.ntt.co.jp).

C. Hori was with NTT Communication Science Laboratories, NTT Corporation, 619-0237 Kyoto, Japan. She is now with InterACT, Carnegie Mellon University, Pittsburgh, PA 15213 USA (e-mail: chiori@cs.cmu.edu).

Y. Minami and A. Nakamura are with NTT Communication Science Laboratories, NTT Corporation, 619-0237 Kyoto, Japan (e-mail: minami@cslab.kecl.ntt.co.jp; ats@cslab.kecl.ntt.co.jp).

Digital Object Identifier 10.1109/TASL.2006.889790

optimized, such partial knowledge updates cannot be accomplished easily. It is necessary to update the component WFSTs whenever changes are made to any of the knowledge sources. Then, we must spend a long time reconstructing the fully composed and optimized WFST.

On-the-fly composition (sometimes called on-demand, lazy, or delayed composition) is a practical alternative to approaches that use a fully composed single WFST [4], [8]–[12]. With on-the-fly composition, component WFSTs are divided into groups of two or more, and one sub-WFST is composed for each group. A one-pass decoder drives these sub-WFSTs in sequence, and composes them during decoding as necessary. In previous studies, on-the-fly composition was economical in terms of memory, but made decoding slower than when driving a fully composed single WFST due to the computational overhead generated by composition during decoding.

On the other hand, a multipass search strategy is often chosen in order to reduce computation and memory use [13]. In a general two-pass search method, a decoder uses a less complex language model (usually a bigram model) in the first pass, and generates a word lattice including multiple hypotheses. In the second pass, the decoder rescores the lattice using a more complex language model (usually a trigram or a higher-order n-gram model), and selects the best hypothesis in the lattice. This method can also work in the WFST framework [14]. However, the multipass approach has a drawback for online applications. It has a certain latency after an utterance. Although the first pass can be performed time-synchronously with speech input, the second pass has to await the completion of the first pass which cannot be completed until the end point of the utterance is detected.

In this paper, we propose a novel search algorithm with on-the-fly composition of WFSTs designed to achieve fast and memory-efficient decoding. In the new algorithm, a Viterbi search is performed using one of two sub-WFSTs, and the other sub-WFST is used solely for rescoring hypotheses generated from the Viterbi search process in an on-the-fly fashion. Since this rescoring only needs minimal computation, the total amount of computation for our new algorithm is almost the same as that needed for the Viterbi search itself. This allows the proposed algorithm to be much faster than conventional search algorithms with on-the-fly composition, and possibly faster even than when driving a fully composed single WFST. From another viewpoint, this algorithm can be interpreted as the simultaneous processing of lattice generation and its rescoring in a time-synchronous way, where the algorithm works completely in one pass. In addition, since all knowledge sources including detailed models (i.e., trigram or higher order n-gram models) are available from the beginning of the search, this is effective for both selecting correct paths and pruning hypotheses.

This paper is organized as follows. In Section II, we review WFSTs in speech recognition. In Section III, we present a baseline search algorithm with on-the-fly composition and describe its problems. In Section IV, we propose a fast search algorithm with on-the-fly hypothesis rescoring including its concept, algorithm, implementation, and noteworthy features compared with other algorithms. In Section V, we provide experimental results for a spontaneous lecture speech transcription task and an extremely large vocabulary recognition task in a spoken interactive open domain question-answering system [1]. We also provide experimental results comparing full and on-the-fly composition methods, one-pass and two-pass strategies, etc. Section VI concludes the paper.

II. WFSTs in Speech Recognition

Continuous speech recognition can be formulated as a problem that involves finding a word sequence $\hat{W}$ such that

$$\hat{W} = \arg\max_{W} P(W|O) \tag{1}$$

$$= \arg\max_{W} P(O|W)P(W) \tag{2}$$

where $P(O|W)$ is the probability or probability density of speech signal O given a word sequence W, and $P(W)$ is the prior probability that a word sequence W occurs. In many cases, we consider the $P(O|W)$ to be a function of W, and its value to be the likelihood of W being the correct transcription of speech signal O. These values are usually estimated based on models, namely, knowledge sources trained from a large amount of speech data, i.e., $P(O|W)$ is calculated based on acoustic models, typically formulated by the HMM, and $P(W)$ is calculated based on language models, typically formulated by the phoneme-level or word-level n-gram probability model. More practically, we incorporate $P(V|W)$, which is the probability of phoneme sequence V given W, and rewrite (2) as

$$\hat{W} = \arg\max_{W} \sum_{V} P(O|V,W)P(V|W)P(W) \tag{3}$$

$$\approx \arg\max_{W} \left\{ \max_{V} P_H(O|V)P_L(V|W)P_G(W) \right\} \tag{4}$$

where $P_H(O|V)$ is the likelihood calculated based on the linear sequence of phoneme-unit acoustic models, $P_L(V|W)$ indicates that the probability can be calculated using lexical or word-pronunciation models, and $P_G(W)$ is the prior probability calculated based on word-level language models. Mainly for implementation reasons, we use the logarithms of these values in a speech recognition decoder. Namely, the decoder performs the following:

$$\hat{W} \approx \arg\max_{W} \left\{ \max_{V} P_H(O|V)P_L(V|W)P_G(W) \right\} \tag{5}$$

$$= \arg\max_{W} \Big\{ \max_{V} \{\log P_H(O|V) + \log P_L(V|W) + \log P_G(W)\} \Big\} \tag{6}$$

For simplicity, henceforth we refer to the logarithmic likelihood and probabilities as scores, for example, $\log P_H(O|V)$ is referred to as the acoustic score, and $\log P_G(W)$ as the language score. In this paper, we employ the WFST framework to enable us to implement (6) accurately, and we propose a fast and memory-efficient algorithm running on the WFST framework.

A WFST is a kind of finite-state automaton that can describe a conversion or *transduction* from one symbol sequence to another, with input and output symbols, as well as a weight, defined on each of its arcs. In the process of transduction, an intermediate weight value is "multiplied" by a weight, most

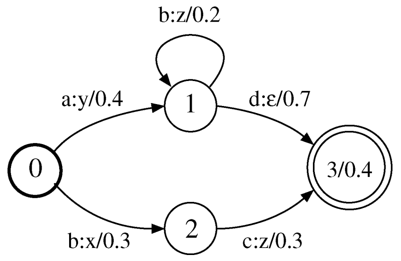

Fig. 1. Weighted finite-state transducer (WFST).

typically at a transition, and two or more intermediate weight values are "summed up" in a state, based on a *semiring* algebraic structure, and this provides the overall weight for the transduction. A semiring consists of a set of arbitrary elements, two formally defined binary operations, i.e., "addition" and "multiplication," over the set, and an identity element in the set for each operation. In this paper, we use the *tropical semiring* for specifically defining the property of real-valued weight, where the "addition" and "multiplication" of two weights are defined as the minimum of the two, and ordinary addition, respectively. See [2] for further details about *semiring*. A WFST T over a *tropical semiring* $\mathcal{K}$ is defined by an 8-tuple as $T = (\Sigma, \Delta, Q, i, F, E, \lambda, \rho)$ where:

1) Σ is a finite set of input symbols;
2) Δ is a finite set of output symbols;
3) Q is a finite set of states;
4) $i \in Q$ is an initial state;
5) $F \subseteq Q$ is a set of final states;
6) $E \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times (\Delta \cup \{\epsilon\}) \times \mathcal{K} \times Q$ is a finite set of transitions;
7) λ is an initial weight;
8) $\rho : F \to \mathcal{K}$ is a final weight function.

ϵ is a meta symbol that indicates there is no symbol to input or output. Fig. 1 shows an example of a WFST. The nodes and arcs correspond to states and transitions of the WFST. The label on each arc denotes "input-symbol: output-symbol/weight," and the final state (state 3) possesses a final weight (0.4). For instance, this WFST transduces a symbol sequence "abbd" into another symbol sequence "yzz" with the overall weight $0.4 + 0.2 + 0.7 + 0.4$ by transition through states 0, 1, 1, and 3.

In the WFST framework, we treat a speech recognition problem as a transduction from input speech signal O to a word sequence W. Each of the models used in speech recognition is, in fact, interpretable as a WFST whose weights are defined as the negatives of scores. Namely, we consider WFSTs H, L, and G corresponding to (6), which transduce O into V with weight $-\log P_H(O|V)$, V into W with weight $-\log P_L(V|W)$, and W into W with weight $-\log P_G(W)$, respectively. Then, the target transduction from O to W can be achieved by a cascade consisting of H, L and G. To make the transduction more efficient, we combine these WFSTs and compose one single WFST N that transduces O into W directly

$$N = H \circ L \circ G \tag{7}$$

where "$\circ$" is a composition operator. Then, the speech recognition is formulated by a search process on N for the word sequence with the minimum overall weight

$$\hat{W} \approx \arg\min_{W} \Big\{ \min_{V} \{(-\log P_H(O|V)) + (-\log P_L(V|W)) + (-\log P_G(W))\} \Big\}. \tag{8}$$

Note that this is obviously equivalent to the word sequence with the maximum overall scores in (6).

When we incorporate triphone models, which are standard models in state-of-the-art speech recognition, we insert an additional WFST C that transduces a triphone sequence into a phoneme sequence according to phonotactic rules

$$N = H \circ C \circ L \circ G. \tag{9}$$

This fully composed WFST organizes an all-in-one search network, and the cross-word triphone contexts are accurately represented there, which is very difficult when we employ traditional formulations. The fully composed WFST can be optimized further by operations commonly employed in WFST frameworks, such as weighted determinization and minimization. These operations achieve global optimization over the whole search space, and can greatly increase the search efficiency while the effects of similar techniques in traditional approaches have been limited to a local part of the search space basically corresponding to each model.

In a common implementation of the WFST-based approach, a decoder pretransduces an input speech signal into a sequence of HMM states, each of which is symbolized by an HMM-state ID, and H in (9) is designed to transduce an HMM-state ID sequence into a triphone sequence [3]. Then, $H \circ C \circ L \circ G$ accepts an HMM-state ID sequence as the input and provides a word sequence as the output. Henceforth, we assume this implementation in our explanations of conventional and proposed algorithms. Also, in preparation for the following sections, we introduce a formal expression of the minimal weight on a WFST N given an input symbol sequence X

$$\Omega(X \to \hat{Y}) = \min_{Y} \Omega_N(X \to Y) \tag{10}$$

where $\Omega_N(X \to Y)$ is the overall weight when WFST N transduces X into Y. Here, we assume that the transition path from the initial to final states is chosen so that the overall weight is the minimum of all the possible candidates. $\hat{Y}$ indicates the output symbol sequence on the minimally weighted path.

III. Search Method With Standard On-the-Fly Composition

When we compose a WFST, we have to consider all the possible combinations of states or transitions from component WFSTs. A single fully composed WFST can easily become huge when one or more of the component WFSTs are large and nondeterministic, and it is often infeasible to store a fully composed WFST in the memory when we deal with a large vocabulary or very detailed linguistic knowledge in speech

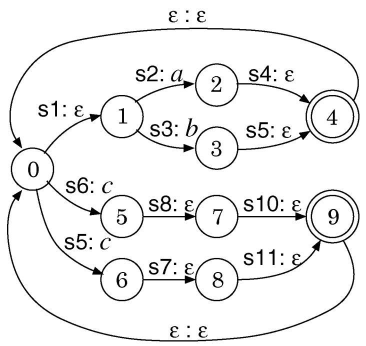

Fig. 2. HMM-state-to-unigram transducer: s1,s2, ..., s11 are HMM-state IDs, and a, b, c represent distinct words.

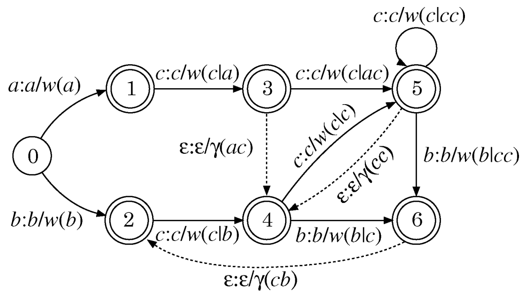

Fig. 3. Trigram model transducer: a, b, c represent distinct words; $w(\cdot)$ on each arc stands for a weight calculated based on minus log-probability, where $w(c|ab) = -\log\{P(c|ab)/P(c)\}$; $\gamma(\cdot)$ stands for a minus log of a back-off weight; the dashed lines indicate back-off transitions.

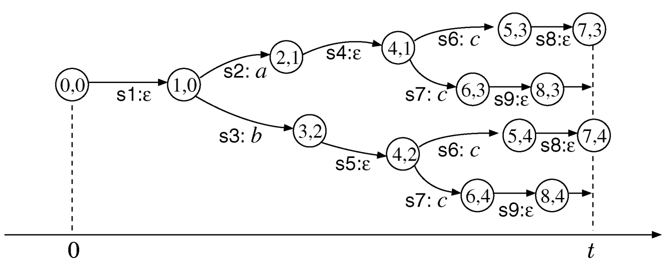

Fig. 4. Hypotheses in a standard search algorithm with on-the-fly composition: each path from the initial state (0,0) represents a hypothesis at the current time frame t during decoding.

recognition. On-the-fly composition using sub-WFSTs is a practical way to avoid this problem.

Suppose that G is weighted based on trigram models. In a standard search with on-the-fly composition, we consider two groups of WFSTs, for example, $\{H, C, L, G_{\text{uni}}\}$ and $\{G_{\text{tri/uni}}\}$ [9], [10], where G is decomposed into G_{uni} and $G_{\text{tri/uni}}$. G_{uni} is weighted based on unigram models, and $G_{\text{tri/uni}}$ possesses trigram-based weights compensated by their corresponding unigram weights so that $G_{\text{uni}} \circ G_{\text{tri/uni}}$ becomes equivalent to G. Then, we compose sub-WFST

$$N_{\text{uni}} = H \circ C \circ L \circ G_{\text{uni}}. \tag{11}$$

Note that the total size of the two sub-WFSTs, N_{uni} and $G_{\text{tri/uni}}$, is smaller than that of the single fully composed WFST N. A decoder drives the two sub-WFSTs in sequence, and successively performs partial composition using pieces from the sub-WFSTs according to the inputs of HMM-state IDs in an on-the-fly fashion.

Figs. 2–4 show examples of N_{uni}, $G_{\text{tri/uni}}$ and the decoding process with on-the-fly composition, respectively. In Fig. 4, the hypotheses are expanded with nodes and arcs along the time axis. Each node on the same time instant is identified by a pair of numbers by which the nodes in Figs. 2 and 3 are identified. Node (2,1), for example, indicates that the node is composed of nodes 2 and 1 in Figs. 2 and 3, respectively. Each arc between the nodes is also composed of two arcs in Figs. 2 and 3. The on-the-fly composition is thus performed only when a decoder has to expand a hypothesis by composing related nodes and arcs with successive inputs of HMM-state IDs taken into account. This reduces the amount of memory needed for storing knowledge sources compared with the prior full composition of the WFSTs.

However, the standard on-the-fly composition can only provide a search space that is unnecessarily larger than that of full composition, which can be fully optimized in advance. Furthermore, the parallel drive of and successive linkage between two sub-WFSTs become an unavoidable overhead and increase the amount of computation required for decoding.

One way to achieve a high search efficiency similar to that of the fully optimized WFST is on-the-fly optimization. In [4], it is shown that on-the-fly determinization improves search efficiency. In [12], on-the-fly pushing and minimization were introduced with some approximations. However, the overhead is basically unavoidable, and the decoding speed is never beyond that of the fully composed WFST.

IV. Search Method With On-the-Fly Rescoring

Two inherent disadvantages of the separate possession of sub-WFSTs in an on-the-fly composition approach are that the provided search space is less optimal since the sub-WFSTs cannot provide a sufficiently detailed view over the search space, and that composition during a search produces an inevitable computational overhead. We propose a new search algorithm to overcome these disadvantages, that can reduce the basic computation amount by rescoring partial hypotheses from the first WFST (not the first WFST itself) with the second WFST sequentially during the search process. We call this mechanism *on-the-fly rescoring*.

The new algorithm can achieve a faster and more memory-efficient search than a search method with standard on-the-fly composition, or even than one that uses a fully composed WFST.

A. Concept

We first define terms for describing the concept of the proposed algorithm. Given a transition e, we denote its input symbol by $i[e]$, its output symbol by $o[e]$, its origin state by $p[e]$, its destination state by $n[e]$, and its weight by $w[e]$. A hypothesis during the search $h = e_1, \ldots, e_k$ is a path along consecutive transitions from the initial state: $p[e_1] = i$, $n[e_{j-1}] = p[e_j]$, $j = 2, \ldots, k$. We extend $n[\cdot]$ and $p[\cdot]$ to paths as $n[h] = n[e_k]$ and $p[h] = p[e_1]$. We also extend $o[\cdot]$ and $w[\cdot]$ to paths as $o[h] = o[e_1] \cdot \ldots \cdot o[e_k]$ and $w[h] = w[e_1] + \ldots + w[e_k]$. We sometimes use a subscript indicating the corresponding WFST for $i[\cdot]$, $o[\cdot]$, $p[\cdot]$, $n[\cdot]$, and $w[\cdot]$ for disambiguation. If

$n[e_k] \in F$ after an input symbol sequence X was accepted (i.e., $i[h] = X$), h is a complete hypothesis. Otherwise, we call h a partial hypothesis or just a hypothesis. In addition, given a WFST T and symbol sequences X and Y, we use $\Pi_T(X \to Y)$ to denote a set of complete paths that accept X and output Y in T.

Suppose there are two WFSTs $A = (\Sigma, \Delta, Q_A, i_A, F_A, E_A, \lambda_A, \rho_A)$ and $B = (\Delta, \Gamma, Q_B, i_B, F_B, E_B, \lambda_B, \rho_B)$, and suppose $A \circ B$ does not become null. In a standard on-the-fly composition, given an input symbol sequence X, the decoder finds $\hat{Z}$ such that

$$\Omega(X \to \hat{Z}) = \min_{Y,Z} \{\Omega_A(X \to Y) + \Omega_B(Y \to Z)\}. \quad (12)$$

In order to find $\hat{Z}$ based on this equation, the decoder is required to examine all the possible combinations of Y and Z. The decoder actually finds the best complete path that outputs $\hat{Z}$ such that

$$\Omega(X \to \hat{Z}) = \min_{h \in \Pi_A(X \to Y), f \in \Pi_B(Y \to Z)} \{w_A[h] + w_B[f]\}. \quad (13)$$

Thus, if the WFSTs have different paths for the same input or the same output symbol sequence, the number of possible combinations becomes even larger.

We note here that (12) can be rewritten as

$$\Omega(X \to \hat{Z}) = \min_{Y} \left\{\Omega_A(X \to Y) + \min_{Z} \Omega_B(Y \to Z)\right\}. \quad (14)$$

This equation means that the algorithm for finding $\hat{Z}$ given X could be reorganized based on an algorithm for finding Y that derives $\min_Y \Omega_A(X \to Y)$ with a slight modification where $\Omega_A(X \to Y)$ is adjusted by the addition of $\min_Z \Omega_B(Y \to Z)$. We can also rewrite this by including paths as

$$\Omega(X \to \hat{Z}) = \min_{Y} \left\{ \min_{h \in \Pi_A(X \to Y)} w_A[h] + \min_{f \in \Pi_B(Y \to Z)} w_B[f] \right\}. \quad (15)$$

Since the decoder finds the minimally weighted path in B only for each Y, it is not necessary to consider all possible path combinations as in (13). This fact reveals the potential for reducing the amount of computation needed for a search with on-the-fly composition.

According to (15), the search procedure consists of two steps: 1) enumeration of all Y on A's paths accepting X, and 2) selection of the minimally weighted path in B's paths accepting Y. In practice, this procedure can be performed with a two-pass strategy, i.e., lattice generation by A and rescoring of the lattice by B, where a WFST is generated by composing the lattice with B, and a shortest path search is applied to the WFST. Since the first pass (lattice generation) uses only A, its computation amount is much smaller than that of $A \circ B$. Furthermore, if pruning techniques are used in the first pass, the size of the lattice can be reduced, and the computation amount for the second pass can also be reduced. Our idea in this paper is to solve (15) with a one-pass search strategy.

In the proposed algorithm, partial hypotheses are generated from A, but they are weighted based on (15) during a time-synchronous Viterbi search. The main difference from two-pass strategies is that the decoder uses all knowledge sources, i.e., both WFSTs A and B, from the beginning of the search. This is effective for both correct path selection and pruning. On the other hand, while a Viterbi search is performed according to $A \circ B$ in standard search methods with on-the-fly composition, it is performed according to A in our algorithm. Hence, our method essentially requires less computation than the standard methods.

Suppose there is a partial hypothesis h with its weight $w_A[h]$, which is generated from A. In our method, h is linked to a set of cohypotheses that are generated from B by taking the symbol sequence $o_A[h]$ as B's input, where we call the hypotheses produced by B "cohypotheses" to distinguish them from the hypotheses produced by A. The decoder rescores h using the minimally weighted cohypothesis in the set. The rescoring is performed very efficiently by managing each hypothesis h with its link to a list of cohypotheses $g[h]$. During decoding, the following basic procedure is used to generate each hypothesis and rescore it by using its related cohypotheses.

When a new hypothesis h' is generated by adding a transition e originating from $n_A[h]$, the weight of h' can be calculated as $w_A[h'] = w_A[h] + w_A[e]$. If the transition e outputs nothing ($o_A[e] = \epsilon$ and $o_A[h'] = o_A[h]$), no new cohypothesis is generated from B. In this case, the list of cohypotheses is kept as it is, i.e., $g[h'] = g[h]$. Only when the transition e outputs a nonepsilon symbol y ($o_A[e] = y \neq \epsilon$ and $o_A[h'] \neq o_A[h]$), a new cohypothesis f' is generated for each existing cohypothesis f in $g[h]$ by adding a transition r, which originates from the state $n_B[f]$ with an input symbol y. The weight of f' can be calculated as

$$w_B[f'] = w_B[f] + w_B[r]. \quad (16)$$

New cohypotheses generated as above are then stored in $g[h']$. We then use the following modified weight, instead of the pure weight $w_A[h']$, in the Viterbi search on A

$$\alpha(h') = w_A[h'] + \min_{f' \in g[h']} w_B[f']. \quad (17)$$

When the Viterbi search chooses the best hypothesis from different hypotheses that meet at the same state in A, their cohypothesis lists are merged into one list. If different cohypotheses have reached the same state in B, only the best cohypothesis among them is retained in the merged list.

After the input symbol sequence has been processed, the best complete hypothesis and the best complete cohypothesis can be simultaneously derived as

$$\hat{h} = \arg\min_{h: n_A[h] \in F_A} \alpha(h) \quad (18)$$

and

$$\hat{f} = \arg\min_{f \in g[\hat{h}]: n_B[f] \in F_B} w_B[f] \quad (19)$$

respectively. Eventually, the result of the search for the best symbol sequence is given by $o_B[\hat{f}]$, that is the output symbol sequence of the best complete cohypothesis.

Fig. 5 shows the decoding process in the proposed method when the decoder uses the WFSTs shown in Figs. 2 and 3. The upper half of Fig. 5 represents a set of hypotheses generated from the first WFST in Fig. 2. Here, the number of hypotheses

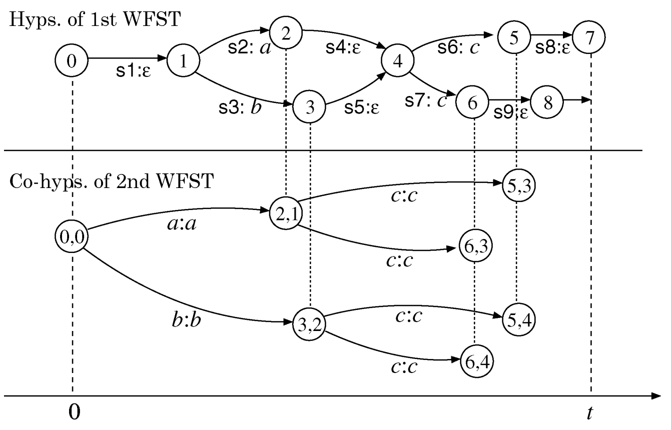

Fig. 5. Hypotheses in the proposed search algorithm.

```
1   $h \leftarrow NewHyp(i)$
2   $\alpha(h) \leftarrow \lambda$
3   $Insert(H, h)$
4   while $EOT(D) = false$ do
5       $x \leftarrow ReadSym(D)$
6       $H' \leftarrow \phi$
7       for each $h \in H$ do
8           for each $e \in E(n[h], x)$ do
9               $h' \leftarrow NewHyp(h \cdot e)$
10              $\alpha(h') \leftarrow \alpha(h) + w[e]$
11              $Insert(H', h')$
12          end for
13      end for
14      $PruneHyp(H')$
15      $H \leftarrow H'$
16  end while
17  for each $h \in H : n[h] \in F$ do
18      $\alpha(h) \leftarrow \alpha(h) + \rho(n[h])$
19  end for
20  $\hat{h} \leftarrow \operatorname{argmin}_{h \in H : n[h] \in F} \alpha(h)$
```

Fig. 6. Search algorithm using a single WFST.

dealt with at each time frame is much smaller than for the standard method shown in Fig. 4. As shown in the lower half of Fig. 5, each hypothesis is linked to a list of cohypotheses generated from the second WFST, and is rescored using cohypotheses in each corresponding set. Since minimal computation is required to update the list of cohypotheses, the total amount of computation is almost the same as when decoding with only the first WFST.

In other words, this algorithm can be interpreted as one-pass processing of lattice generation and rescoring. The set of hypotheses generated by the first WFST at each time frame can be considered a lattice as in Fig. 5. The rescoring is performed as necessary based on the lattice by composing it with the second WFST. Thus, the total computation amount is almost the same as that for a two-pass search with two WFSTs. However, our algorithm can utilize both WFSTs from the beginning of the search and complete the work in one pass.

B. Algorithm

We describe the proposed search algorithm, and compare it with standard algorithms. Before describing the algorithms, we incorporate the following functions.

- $ReadSym(D)$: read a symbol from an input tape D, return the symbol and move the head to the right direction.
- $EOT(D)$: return *true* if the head is located at the end of the input tape D; otherwise return *false*.
- $NewHyp(h)$: generate a hypothesis and initialize it as hypothesis h or initial state i.
- $Insert(H, h)$: insert hypothesis h into a hypothesis list H (If there is a hypothesis j that has reached $n[h]$ (i.e., $n[h] = n[j]$) in H, either h or j is retained, whichever has the smaller weight).
- $PruneHyp(H)$: prune unpromising hypotheses from hypothesis list H based on their accumulated weights.
- $E(q, x)$: return a set of transitions accepting symbol x from state q.

Although the algorithms are written assuming that WFSTs accept a symbol sequence on a single input tape, they can accept an acoustic feature vector sequence for speech recognition by extending the functions, $ReadSym(D)$, $EOT(D)$, and $E(q, x)$. In speech recognition, $ReadSym(D)$ returns an acoustic feature vector for the current time frame, and $EOT(D)$ returns *true* when the end point of the input utterance is detected. For $E(q, x)$, a special procedure is necessary. If x is a feature vector and the WFST has an HMM-state ID on the input side of each transition, $E(q, x)$ returns all nonepsilon transitions originating from q, where a minus log acoustic likelihood of x calculated with the HMM state is added to the weight of each transition dynamically. In addition, we assume that the WFST has transitions for self loops of the HMM although they are omitted in Fig. 2. Self loops of HMMs are usually discarded from the WFST when storing it and when performing composition and optimization, but those transitions are simulated by the run-time decoder.

Several pruning techniques are available for $PruneHyp(H)$. In our case, first weight-based pruning is applied, in which hypotheses with a higher accumulated weight than a given threshold are pruned. The threshold is derived by the minimum accumulated weight in H plus a fixed parameter Φ that is a predefined real number. Next histogram pruning is applied, in which hypotheses are pruned except for the top M hypotheses in H based on their accumulated weights, where M is a predefined fixed number [13].

First, we show the basic search algorithm when using a single WFST $T = (\Sigma, \Delta, Q, i, F, E, \lambda, \rho)$ that accepts an input symbol sequence read from a tape D. Fig. 6 describes the algorithm. The initial hypothesis h is generated on line 1. On lines 2 and 3, h is weighted with the initial weight λ and stored in the hypothesis list H. On lines 4–16, the main search step is performed until the final input symbol is processed. The main step iterates the generation of hypotheses by $NewHyp()$ and their recombination by $Insert()$.

On line 5, an input symbol x is set by $ReadSym(D)$. The new hypothesis list H' is initialized on line 6. On lines 7–13, new hypotheses h' are generated for each hypothesis in the current hypothesis list H by adding possible transitions e from $n[h]$, the hypotheses are weighted, and inserted into H'. $h \cdot e$ on line 9 indicates a path extension realized by adding a transition e to h. On line 14, unpromising hypotheses are pruned from

```
 1  $h \leftarrow NewHyp((i_A, i_B))$
 2  $\alpha(h) \leftarrow \lambda_A + \lambda_B$
 3  $Insert(H, h)$
 4  while $EOT(D) = false$ do
 5      $x \leftarrow ReadSym(D)$
 6      $H' \leftarrow \phi$
 7      for each $h \in H$ do
 8          for each $e \in E(n_A[h], x)$ do
 9              if $o_A[e] \neq \epsilon$ then
10                  for each $r \in E(n_B[h], o_A[e])$ do
11                      $h' \leftarrow NewHyp(h \cdot (e \circ r))$
12                      $\alpha(h') \leftarrow \alpha(h) + w_A[e] + w_B[r]$
13                      $Insert(H', h')$
14                  end for
15              else
16                  $h' \leftarrow NewHyp(h \cdot e)$
17                  $\alpha(h') \leftarrow \alpha(h) + w_A[e]$
18                  $Insert(H', h')$
19              end if
20          end for
21      end for
22      $PruneHyp(H')$
23      $H \leftarrow H'$
24  end while
25  for each $h \in H : n_A[h] \in F_A$ and $n_B[h] \in F_B$ do
26      $\alpha(h) \leftarrow \alpha(h) + \rho_A(n_A[h]) + \rho_B(n_B[h])$
27  end for
28  $\hat{h} \leftarrow \operatorname{argmin}_{h \in H : n_A[h] \in F_A \& n_B[h] \in F_B} \alpha(h)$
```

Fig. 7. Search algorithm with standard on-the-fly composition.

H' by $PruneHyp()$. On line 15, H is replaced with H' for the next input.

After the main step, hypotheses reaching one of the final states in H are weighted with the corresponding final weight on lines 17–19. On line 20, the best complete hypothesis $\hat{h}$ is chosen from among the hypotheses, and the output symbol sequence $o[\hat{h}]$ becomes the recognition result. For simplicity's sake, no ϵ-transitions are assumed in the algorithm.

Next, we describe the search algorithm with the standard on-the-fly composition of two WFSTs. In general, when using on-the-fly composition, a composite WFST is constructed virtually, i.e., its states and transitions are composed as necessary. In this case, the decoding process can be performed by the algorithm in Fig. 6 by assuming the two WFSTs as a single composite WFST. However, since it is difficult to clarify the actual computation for the search, we describe a search algorithm including on-the-fly composition. Fig. 7 shows the algorithm.

Given two WFSTs A and B, the algorithm searches for the minimally weighted path in the composed search space of $A \circ B$. On line 1, initial hypothesis h is generated based on a composite initial state (i_A, i_B). On lines 2 and 3, h is weighted with the initial weights λ_A and λ_B, and stored in the hypothesis list H. On lines 4–24, the main search step is performed until the final input symbol is processed. Unlike the algorithm for a single WFST, a new hypothesis h' is generated based on a composite transition $e \circ r$ leaving from the composite state $n[h]$ where $n[h]$ consists of $n_A[h]$ and $n_B[h]$, i.e., $n[h] = (n_A[h], n_B[h])$; During the main step, the number of hypotheses in H increases according to the number of combinations of transitions leaving from the original two states $n_A[h]$ and $n_B[h]$.

After the main step, hypotheses reaching one of the final composite states in $F_A \times F_B$ are weighted with the corresponding final weights on lines 25–27. On line 28, the best complete hypothesis $\hat{h}$ is chosen from among the hypotheses.

```
 1  $h \leftarrow NewHyp(i_A)$
 2  $f \leftarrow NewHyp(i_B)$
 3  $\tilde{\alpha}(f) \leftarrow \lambda_A + \lambda_B$
 4  $Insert(g[h], f)$
 5  $\alpha(h) \leftarrow \tilde{\alpha}(f)$
 6  $Insert(H, h)$
 7  while $EOT(D) = false$ do
 8      $x \leftarrow ReadSym(D)$
 9      $H' \leftarrow \phi$
10      for each $h \in H$ do
11          for each $e \in E(n_A[h], x)$ do
12              $h' \leftarrow NewHyp(h \cdot e)$
13              if $o_A[e] \neq \epsilon$ then
14                  $\beta \leftarrow \alpha(h) + w_A[e] - \min_{f \in g[h]} \tilde{\alpha}(f)$
15                  for each $f \in g[h]$ do
16                      for each $r \in E(n_B[f], o_A[e])$ do
17                          $f' \leftarrow NewHyp(f \cdot r)$
18                          $\tilde{\alpha}(f') \leftarrow \tilde{\alpha}(f) + \beta + w_B[r]$
19                          $Insert(g[h'], f')$
20                      end for
21                  end for
22                  $\alpha(h') \leftarrow \min_{f' \in g[h']} \tilde{\alpha}(f')$
23              else
24                  $g[h'] \leftarrow g[h]$
25                  $\alpha(h') \leftarrow \alpha(h) + w_A[e]$
26              end if
27              $Insert(H', h')$
28          end for
29      end for
30      $PruneHyp(H')$
31      $H \leftarrow H'$
32  end while
33  for each $h \in H : n_A[h] \in F_A$ do
34      $\beta \leftarrow \alpha(h) + \rho_A(n_A[h]) - \min_{f' \in g[h]} \tilde{\alpha}(f')$
35      for each $f \in g[h] : n_B[f] \in F_B$ do
36          $\tilde{\alpha}(f) \leftarrow \tilde{\alpha}(f) + \beta + \rho_B(n_B[f])$
37      end for
38      $\alpha(h) \leftarrow \min_{f' \in g[h]} \tilde{\alpha}(f')$
39  end for
40  $\hat{h} \leftarrow \operatorname{argmin}_{h \in H : n_A[h] \in F_A} \alpha(h)$
41  $\hat{f} \leftarrow \operatorname{argmin}_{f \in g[\hat{h}] : n_B[f] \in F_B} \tilde{\alpha}(f)$
```

Fig. 8. Search algorithm with on-the-fly rescoring.

Finally, we present the search algorithm with on-the-fly rescoring shown in Fig. 8. In the algorithm, we introduce a quantity

$$\tilde{\alpha}(f) = w_A[h] + w_B[f] \tag{20}$$

where $f \in g[h]$. We use $\tilde{\alpha}(f)$ to weight cohypothesis f instead of $w_B[f]$ since this simplifies the description of the algorithm. Thus, we rewrite (17) as

$$\alpha(h') = \min_{f' \in g[h']} \tilde{\alpha}(f'). \tag{21}$$

Initial hypothesis h for A is generated on line 1. On lines 2 and 3, initial cohypothesis f is generated for B and weighted with $\lambda_A + \lambda_B$. On line 4, f is inserted into a cohypothesis list of h, $g[h]$. On line 5, h is weighted with $\tilde{\alpha}(f)$ according to (17), where $\min_{f \in g[h]}$ is omitted since $g[h]$ contains only f. Then, h is inserted into H on line 6.

The main search step is performed on lines 7–32. The main step is similar to the search with a single WFST (Fig. 6), except for lines 13–23, where on-the-fly rescoring is performed only when transition e has a nonepsilon output symbol. A compensated weight and a new cohypothesis list are obtained in this part. If this part can be ignored, the complexity of this algorithm is the same as that of the algorithm in Fig. 6.

An accumulated weight $\tilde{\alpha}(f')$ for a new cohypothesis f' derived from a cohypothesis f in $g[h]$ can be computed as a sum of the following three elements:

- $\tilde{\alpha}(f)$: the accumulated weight through f;
- β: a weight accumulated through consecutive transitions with epsilon plus the weight of the last transition e with a nonepsilon output symbol in A;
- $w_B[r]$: the weight of the transition r leaving from state $n_B[f]$ by $o_A[e]$ in B.

β is calculated on line 14. β can be considered the accumulated weight of the partial path along h from the point at which the cohypothesis list $g[h]$ was updated. Each transition in the partial path has an epsilon output symbol since $g[h]$ is not updated until a nonepsilon output symbol appears. Let $\bar{h}$ be the hypothesis when $g[h]$ has just been updated. Since $\alpha(\bar{h})$ is equal to $\min_{f\in g[h]} \tilde{\alpha}(f)$ if we consider $g[h] = g[\bar{h}]$, β can be computed as line 14.

On lines 15–21, each cohypothesis f in $g[h]$ is expanded according to the second WFST B, and $g[h']$ including new cohypotheses f' is generated. $\tilde{\alpha}(f')$ can be computed on line 18 as stated above. $\alpha(h)$ is derived as (21) on line 22.

On lines 33–39, final weights are added to each hypothesis and the corresponding cohypotheses using β as on lines 14–22. The best complete hypothesis $\hat{h}$ is chosen on line 40, and the best complete cohypothesis $\hat{f}$ is chosen on line 41. The output symbol sequence $o_B[\hat{f}]$ is the recognition result.

As stated in the concept of our search algorithm, when one hypothesis encounters another at a state in A, only the better one with a smaller accumulated weight survives and then the two cohypothesis lists are merged. The merged list is attached to the surviving hypothesis. This procedure is performed in the function $Insert()$ on line 27, in which the accumulated weight of each cohypothesis needs to be updated before merging as

$$\tilde{\alpha}(f) = \tilde{\alpha}(f) + \left\{ \alpha(h) - \min_{f'\in g[h]} \tilde{\alpha}(f') \right\} \tag{22}$$

where $f \in g[h]$. In the list-merging process, only the best cohypothesis survives among the cohypotheses that arrive at the same state in B. Since the accumulated weight of each cohypothesis is not updated until h is expanded by a transition with a nonepsilon output symbol, the weights of the cohypotheses must be updated before their accumulated weights are compared. Although this process needs a certain overhead, it can be skipped when the two cohypothesis lists are identical. Furthermore, the frequency of merging can be reduced by introducing the approximation explained in Section IV-C.

Our algorithm generates fewer hypotheses than the standard on-the-fly composition algorithm during the search since it generates hypotheses based on the first WFST while the standard algorithm generates hypotheses based on the composite WFST. Thus, the basic Viterbi search is performed based only on the first WFST in our algorithm. Additional computation for updating the cohypothesis lists is only necessary when a hypothesis is expanded by a transition with a nonepsilon output symbol. If the first and second WFSTs are designed as N_{uni} and $G_{\mathrm{tri/uni}}$ in Section III, most transitions in the first WFST have an epsilon output symbol. Accordingly, very little computation is needed for handling the lists.

C. Approximation in Decoding

The proposed algorithm shown in Fig. 8 ensures that the best hypothesis is found if no pruning is performed but does not necessarily ensure that this is true in a practical implementation for speech recognition.

In our implementation, the decoder uses two WFSTs, the first of which has a self loop at each HMM state, but the transition is not explicitly written in the WFST. After composition of H, C, L, and G, each transition of the composed WFST has a single HMM-state ID or an epsilon as its input symbol. In addition, a linear HMM-state sequence is padded in one transition by the factorization operation [2]. The WFST usually includes many chains, where a chain means a path whose states other than the first and last have at most one outgoing and one incoming transition. The factorization operation replaces each chain with one transition, where a new input symbol corresponding to the HMM-state sequence on the chain is assigned to the new transition. This operation helps to reduce the size of the WFST. As a result of factorization, the input symbol of each transition is an HMM-state ID, an HMM-state ID sequence, or an epsilon.

The HMM-level state transitions padded in one transition are handled by the decoder in addition to the WFST-level state transitions. In the Viterbi computation for the HMM-level state transitions, we avoid merging the cohypothesis lists when one hypothesis encounters another in an HMM state because frequent list merging potentially results in a certain overhead, although each merging process involves little computation. However, this simplification requires an assumption to ensure that the best hypothesis is found.

Fig. 9 shows an example of an error that occurs when the assumption is not satisfied. A trellis space formed by a part of a WFST and a time axis can be seen in the figure. The WFST-like graph shown on the Y-axis is a network actually searched by the decoder, where it comprises four WFST states (the numbered large nodes) and 11 HMM states (the small nodes labeled by HMM-state IDs as "sK" where K is an integer). This graph is prepared on demand by the decoder using each input symbol, i.e., HMM-state ID sequence, associated with each transition of the original (factorized) WFST. The graph also has input and output symbols (e.g., "$\epsilon : a$") inherited from the original WFST.

There are three Viterbi paths representing hypotheses in the trellis space. At time t_1, one hypothesis encounters another, and only the better hypothesis survives and proceeds following t_1. Then the two cohypothesis lists attached to the hypotheses are merged into one list that is delivered to the surviving hypothesis. However, at time t_2, the cohypothesis list attached to the worse hypothesis represented with a dashed line is lost since we avoid merging the cohypothesis lists when a hypothesis encounters another in an HMM state. Accordingly, search errors will occur as a result of the loss of the cohypothesis lists in an HMM state if

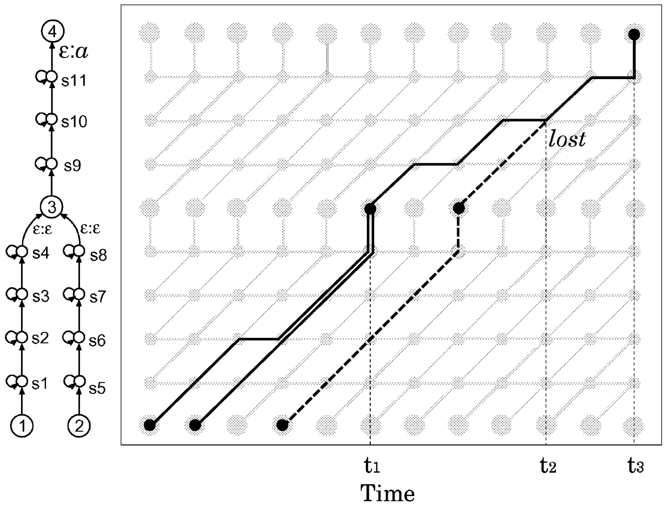

Fig. 9. Example of approximation error. There are three hypotheses in a trellis space organized with time on the X-axis and a search network on the Y-axis. This example shows that one of the hypotheses can be lost with its cohypothesis list due to the mismatch in the time alignment from the other hypotheses.

the cohypothesis that will become the best complete cohypothesis is included in the lost list. The loss of such cohypotheses also affects the weights of the succeeding hypotheses. To ensure both the best hypothesis and cohypothesis are retained, the boundary time between s4 and s9 (i.e., WFST state 3) must be equal to that between s8 and s9 for all the hypotheses.

However, as mentioned in [15] triphones yield a good assumption. If cross-word triphones are used, all transitions to a WFST state come from HMM states associated with a unique preceding phone. In that case, when hypotheses ending with the same phone meet in a WFST state during decoding, the boundary time between the phone and the succeeding phone tends to be equal on the Viterbi paths. This kind of assumption is also used to generate word lattices in the WFST framework [14].

In addition, even if the boundary times are not the same, critical errors rarely occur because the same or similar cohypotheses with the same output symbol sequence are usually included in both survived and lost lists. However, the accumulated weight may change slightly due to a declination of the time alignment. Since the time alignment is decided based on the first WFST as well as the two pass search [14], it can be different from that derived by the composed WFST. In Fig. 5, for example, the boundary times between s4 and s6 and between s5 and s6 necessarily share a common time decided by the first WFST, while they can be different from each other in Fig. 4. Hence, when using the proposed method, a declination of the time alignment can occur even if the correct output sequence is derived.

Although the preceding phone is not necessarily unique since the WFST is actually optimized up to the shared HMM states, we can say that *phone-pair approximation* is roughly assumed. The phone-pair approximation assumes that the best starting time for a phone depends solely on the preceding phone rather than on the entire preceding phone sequence. In the example in Fig. 9, since HMM states s4 and s8 belong to the same center phone or similar phones, the boundary times for s9 will be the same or close in terms of the time alignments of the hypotheses. If this assumption is satisfied, our method ensures that the best hypothesis is found. Even if the assumption is not satisfied, the error rarely militates against the recognition result since the accumulated weight does not change significantly with small differences in time alignment.

There are some similar approaches to our method in the conventional framework of speech recognition. In [16] and [17], a Viterbi search is performed based on a single lexical tree unlike the search with word-dependent tree copies [13]. When using the single tree and a bigram language model, since hypotheses are recombined before bigram probabilities are applied, the most probable preceding word needs to be reselected at each leaf node of the tree, where the identity of the current word is known. However, the most probable starting time of the current word cannot be reselected because the hypothesis knows just one time alignment. This sometimes results in search errors when the decoder is finding the best complete path or generating a lattice. To avoid such errors, in [13] and [18], hypotheses are generated depending on the preceding word. In [15] and [19], phone-dependent or phone-sequence-dependent hypotheses are used. With these methods, it is assumed that the most probable starting time of the current word is solely dependent on the preceding phone, short phone sequence, or word.

We consider our implementation to be a search with an optimized single tree, although it is no longer a tree after optimization. As stated above, search errors can be almost completely avoided by using crossword triphones. Furthermore, since our algorithm is generalized for WFSTs, we can use a one-pass search with any language model in the WFST framework.

D. Hypothesis Data Management

All algorithms described in this section use a list structure for managing hypotheses (and cohypotheses). In the algorithms, a set of (partial or complete) hypotheses are stored as a lattice that contains nodes and arcs as in Figs. 4 and 5. During decoding, each partial hypothesis generated by the current symbol input is represented as a token that indicates the forefront of the partial hypothesis. A token corresponding to a hypothesis h has the accumulated weight $\alpha(h)$, a pointer to the current state $n[h]$, and a back pointer to a lattice node as its history. It also has a cohypothesis list $g[h]$ in the proposed method. The lattice nodes and arcs are generated when tokens visit the states of the WFST being searched. The tokens are stored in a simple chained list H.

In the proposed algorithm, each token has a cohypothesis list. We also have tokens representing cohypotheses. A token for a cohypothesis f has $\tilde{\alpha}(f)$, a pointer to $n_B[f]$, and a back pointer to a lattice node. The lattice nodes and arcs are generated based on the second WFST independently of the first WFST. The best (minimally weighted) cohypothesis in each cohypothesis list, is always placed at the first position of the list to obtain $\min_{f\in g[h]} \tilde{\alpha}(f)$ quickly. In addition, when the size of each cohypothesis list increases, the amount of computation required for inserting new cohypotheses and merging lists also increases. Therefore, we set a maximum number for the cohypotheses stored in the list. Only this fixed number of minimally weighted cohypotheses is kept in the list. This is a kind of pruning for cohypotheses. In our preliminary experiment, 15 cohypotheses were sufficient to avoid any increase in the number of word errors in several tasks, and this reduced the total amount of computation needed for decoding by 5 to 10%. We used this number in all our experiments.

V. Experiments

We evaluated our search method with a 65k-word spontaneous speech transcription task and an extremely large vocabulary spoken question recognition task. In the first task, we investigated the basic performance of our decoding method using a normal vocabulary size of 65 000 words. In the second task, we evaluated our method after increasing the vocabulary size to 1.8 million words. We also compared our one-pass algorithm with a general two-pass method in this task.

In all the experiments, we built the WFSTs as follows:

$$N = \min\left(\mathrm{fact}\left(\pi_\epsilon\left(\mathrm{opt}\left(\tilde{H}\circ\mathrm{opt}\left(\tilde{C}\circ\mathrm{opt}(\tilde{L}\circ G)\right)\right)\right)\right)\right) \tag{23}$$

where $\mathrm{opt}(\cdot)$ means a set of operations, determinization, pushing, and minimization, and $\mathrm{fact}(\cdot)$ means factorization operation. As in [2], auxiliary symbols are inserted into H, C, L, which are written as $\tilde{H}$, $\tilde{C}$, $\tilde{L}$ in (23), to make the WFST at each optimization step determinizable. $\pi_\epsilon(\cdot)$ represents an epsilon removal operation in which the auxiliary symbols are also removed. N_{uni} for on-the-fly composition was also constructed with the same steps. WFSTs G and $G_{\mathrm{tri/uni}}$ for trigram language models were designed to be a compact representation with back-off transitions.

A. Evaluation With CSJ Task

The first task is based on a corpus of spontaneous Japanese (CSJ) [20], mostly comprising monologues such as lectures, presentations, and news commentaries.

The evaluation data were limited to presentations in academic fields. The speeches were digitized with 16-kHz sampling and 16-bit quantization. The feature vectors had 39 elements consisting of 12 MFCCs and a log-energy, their delta and delta–delta components. Tied-state triphone HMMs with 2000 states and 32 Gaussians per state were made by using 787 presentations in the corpus uttered by male speakers (approximately 187 h). A trigram language model was estimated using the manually transcribed text data of 2672 presentations. It was constructed by using the Witten–Bell back-off method with a frequency cutoff of 1 for bigrams and trigrams. Benchmark test 1 was used for the evaluation, which consists of ten academic talks presented by male speakers. The test-set perplexity is 74.8, and the out-of-vocabulary rate is 1.7%.

We prepared a fully composed WFST and a pair of WFSTs for on-the-fly composition. The size of each WFST is shown in Table I. The size (#transition) of the first WFST for on-the-fly composition is much smaller than that of a fully composed WFST. Even when we add that of the second WFST, the total number of transitions is only about half that of the fully composed WFST.

Fig. 10 shows the relationship between word accuracy (WACC) and decoding time in real time factor (RTF) for each decoding method when we change the parameters of the decoder for pruning. To obtain each pair of WACC and RTF, we first fixed the threshold for weight-based pruning, and then we set the beam width for histogram pruning so that the width was as small as possible under the condition that the accuracy change by histogram pruning was less than 0.1%. We used a speech recognizer *SOLON* [21] developed at NTT Communication Science Laboratories, which performs a one-pass Viterbi search based on a single WFST or two WFSTs that can be composed. A standard PC (with a Pentium4 processor of 3.4 GHz and 2 GB of memory) was used to measure the speed of the decoder.

TABLE I
Size of WFSTs in CSJ Task

	1st WFST		2nd WFST	
	#state	#transition	#state	#transition
Full comp.	885,571	2,600,947	-	-
On-the-fly comp.	40,227	122,837	397,938	1,412,550

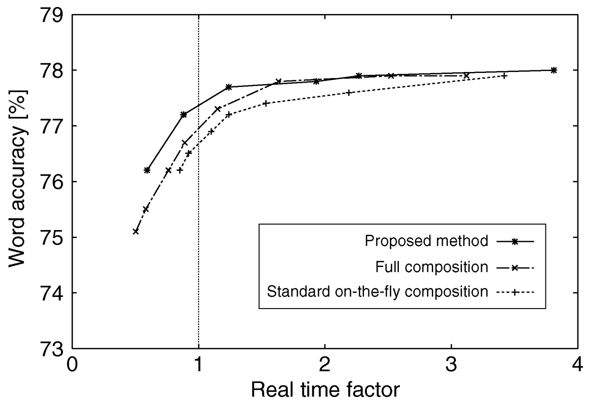

Fig. 10. Relationship between word accuracy and real time factor for the full composition, standard on-the-fly composition and proposed methods in CSJ task.

The decoding time is represented by an RTF that indicates the ratio of decoding time to utterance time. Our proposed method outperformed the full composition method in terms of speed and accuracy. In this case, our method required 55% of the total memory needed by the full composition method, where 27% was used for the acoustic model. Thus, the memory required for storing WFSTs and hypotheses in decoding was 38% of that consumed when using a fully composed WFST. Our method achieves the same accuracy as standard on-the-fly composition, but is about 1.5 to 2 times faster.

B. Evaluation With ODQA Task

We conducted another set of experiments on a task for a spoken interactive open-domain question-answering (ODQA) system developed at NTT Communication Science Laboratories [1].

We developed five sets of lexicon and trigram language models that corresponded to 65k, 200k, 1M, and 1.8M vocabularies. The language models were trained using newspaper articles covering the last 12 years and about 14 000 interrogative sentences. The interrogative sentences were multiplied by 120 in counting frequencies to make language models. Named entities were extracted [22] for all the training data, and the extracted entities were dealt with as regular words. As a

TABLE II
SIZE OF WFSTs IN ODQA TASK

Vocabulary size	LM shrinking	Composition	1st WFST		2nd WFST	
			#state	#transition	#state	#transition
65K	no	on the fly	45,566	121,516	2,767,284	12,812,718
200K	no	on the fly	136,627	350,910	3,625,767	14,764,035
1M	no	on the fly	711,518	1,737,112	4,433,470	16,252,863
1.8M	no	on the fly	1,328,614	3,221,735	5,281,507	17,911,618
65K	yes	on the fly	45,566	121,516	1,810,340	5,166,403
1.8M	yes	on the fly	1,328,614	3,221,735	3,799,891	9,132,457
65K	yes	full	3,820,743	12,691,687	-	-
1.8M	yes	full	5,602,343	17,262,868	-	-

TABLE III
OUT-OF-VOCABULARY RATE AND TEST-SET PERPLEXITY IN ODQA TASK

Vocabulary size	LM shrinking	OOV rate[%]	Perplexity
65K	no	3.4	69.2
200K	no	1.3	83.9
1M	no	0.7	91.5
1.8M	no	0.4	98.1
65K	yes	3.4	71.6
1.8M	yes	0.4	101.2

result, the total number of different words in the data increased from 500 thousand to 1.8 million. It is important to include the named entities in the vocabulary in this task because the speech recognition accuracy for the named entities improved significantly in a preliminary experiment. The probabilities of the language models were estimated by using the Witten–Bell back-off method with a frequency cutoff of 3 for bigrams and trigrams.

Tied-state triphone HMMs with 3000 states and 16 Gaussians per state were trained by using read speech data uttered by about 300 speakers (approximately 50 h). The speeches were processed under the same conditions as the experiment in the CSJ task.

For each vocabulary size, we built a pair of WFSTs for on-the-fly composition. We used a 64-bit computer with 16 GB of memory to compose and optimize the WFSTs. Although we also tried to build full-composite transducers for this task, it was impossible because of insufficient memory. Therefore, we built small language models using a relative entropy-based technique for shrinking n-gram models [23]. We set the threshold of 5×10^{-8}, which represents the upper limit of the relative perplexity gain due to removing each n-gram, i.e., all n-grams that raise the perplexity by less than the threshold are removed. If the threshold becomes larger, the language model becomes smaller with a larger information loss. We chose the value so that it was as small as possible to the extent that the memory size for building full-composite WFSTs did not exceed the limitation of the computer. We applied the shrinking method to the 64k- and 1.8M-word vocabulary trigram models, and then constructed WFSTs for them. The sizes of the WFSTs we built are summarized in Table II.

We used a set of evaluation data consisting of 500 utterances composed freely by 25 male and 25 female subjects and read by the same subjects. Each subject composed and uttered ten sentences. Table III shows the complexity of the task.

Fig. 11 shows the relationship between word accuracy and decoding time when using cutoffs of 3 for bigrams and trigrams.

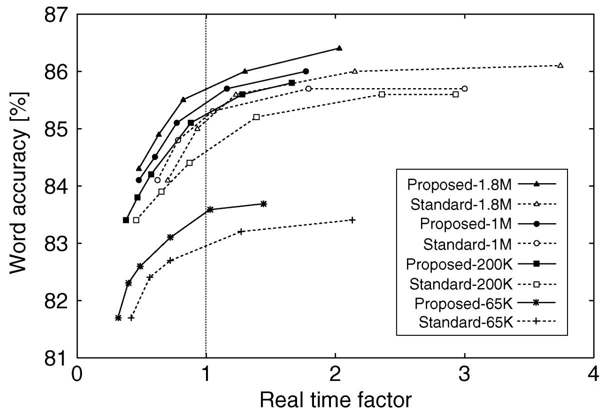

Fig. 11. Relationship between word accuracy and real time factor for the standard on-the-fly composition and proposed methods with different vocabulary sizes in ODQA task.

We used the same computer as that used in the CSJ task to measure the speed of the decoder (SOLON). Word accuracy increases as the vocabulary size increases. Thus an extremely large vocabulary is effective in this task. The 200k, 1M, and 1.8M vocabularies significantly outperformed the 65k vocabulary. For every vocabulary, the proposed method is about 1.5 to more than 3 times faster than the standard method with equal accuracy. These results are similar to those for the CSJ task.

Fig. 12 shows the relationship between word accuracy and decoding time when using the small language models. Compared with the results for the 65k- and 1.8M-word vocabularies in Fig. 11, the word accuracies decreased slightly. In addition, the speed of the proposed and standard methods became closer to each other. This indicates that the search space realized by the standard methods became smaller by composing a smaller language model. In contrast, the search efficiency of the proposed method is not easily affected by the second WFST because it depends on the first WFST rather than the fully composed WFST.

C. Comparison of One-Pass and Two-Pass Search Strategies

We compared our one-pass search method with a standard two-pass approach [14] in an ODQA task. Although our decoder can directly generate a word lattice unlike [14] in which a phone-to-word transducer lattice is first generated and then translated into a word lattice, the basic mechanism for lattice generation is the same as [14]. In our decoder, translation into the word lattice is performed every 1 s (100 frames) together with garbage collection for removing dispensable lattice arcs.

We designed a two-pass decoder in which a bigram-based fully composed WFST is first used to generate a word lattice,

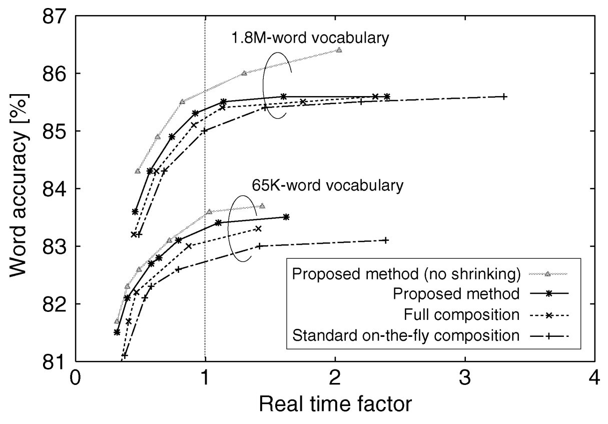

Fig. 12. Relationship between word accuracy and real time factor for the full composition, standard on-the-fly composition, and proposed methods when using shrunk language models in ODQA task. The figure also includes the result when using the original (no shrinking) language models for reference.

TABLE IV
SIZE OF WFSTs USED IN THE TWO-PASS SEARCH METHOD

Vocabulary	WFST for 1st pass		WFST for 2nd pass	
size	#state	#transition	#state	#transition
65K	2.079,290	6,473,159	2,767,284	12,812,718
1.8M	4,083,822	11,249,187	5,281,507	17,911,618

and then a trigram language model is composed with the lattice for rescoring. The conventional shortest path search algorithm was used to find the best path in the rescored lattice.

We built bigram models for 65k- and 1.8M-word vocabularies, which were shrunk with a threshold of 10^{-8} to compile fully composed WFSTs. The test-set perplexity was 88.3 for 65k and 121.1 for 1.8M. We used the original trigram models for rescoring, which were not shrunk. The size of the WFSTs used in the two-pass search is shown in Table IV.

Fig. 13 shows the relationship between word accuracy and decoding time when using the proposed one-pass and standard two-pass methods. For reference, the results of the first pass search in the two-pass method are also plotted in the figure.

For both vocabularies, our one-pass method outperformed the standard two-pass method. With the 1.8M-word vocabulary, the difference between the one-pass and two-pass methods is larger than that with the 65k-word vocabulary. We assume there are two reasons. One is that since the size of word lattices tends to become larger as the vocabulary size increases, the overhead to generate and rescore lattices results in an increase in CPU time. The word lattices in the 1.8M-word vocabulary were actually more than 1.5 times larger than those in the 65k-word vocabulary when using the same beam setting. The other reason is that the bigram model is insufficiently accurate to retain correct sentences in word lattices when using an extremely large vocabulary. This is suggested by the fact that the accuracy gain by the trigram model is larger with the 1.8M-word vocabulary. Since our one-pass search can use the trigram model from the beginning of the search, it is effective for both selecting correct paths and pruning hypotheses.

Next, we evaluated the latency of the one-pass and two-pass decoders. Low latency is desirable for online applications such as spoken dialogue systems and automatic closed captioning systems [24]. Our QA system is a kind of spoken dialogue

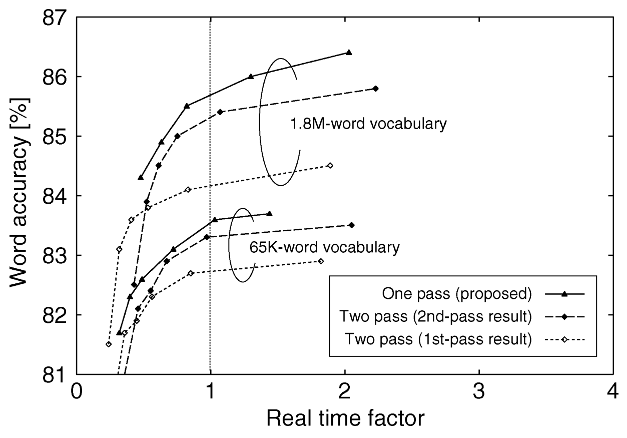

Fig. 13. Relationship between word accuracy and real time factor in ODQA task with one-pass and two-pass search strategies.

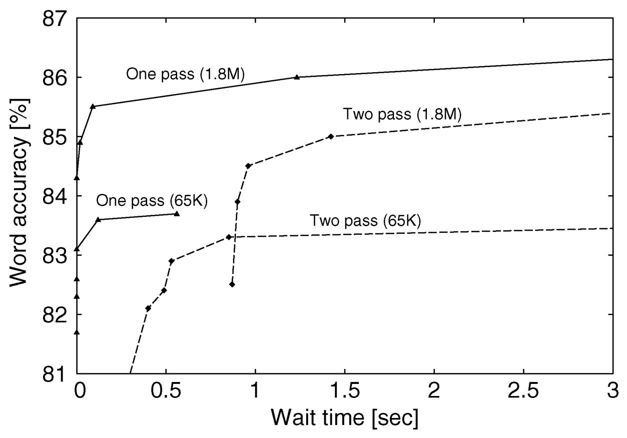

Fig. 14. Relationship between word accuracy and wait time in ODQA task with one-pass and two-pass search strategies. "Wait time" indicates the average latency from the end of each utterance to the output of the recognition result.

system. Low latency is also important because delayed responses usually make the system inconvenient and the user uneasy.

We measured the latency (wait time) from the end of each utterance to the output of the recognition result. In this experiment, we simulated an online condition under which frame-by-frame processing does not proceed beyond the utterance time. Thus, the wait time never becomes negative. Fig. 14 shows the average wait time for the one-pass and two-pass search methods. These results show that our one-pass decoding achieves a very low latency.

D. Discussion of Experimental Results

Here we report our experimental findings.

1) In histogram pruning, partial hypotheses are pruned at each frame except for the top M hypotheses based on their accumulated weights. With the proposed method, the parameter M was sufficient at about half of the value used with the standard on-the-fly method to achieve the same word accuracy. The reason is that the Viterbi search of our proposed method is performed based on the first WFST unlike the other methods, which are based on the composed WFST. Table V includes the histogram pruning parameter M, the real time factor, and the number of evaluated hypotheses

TABLE V
NUMBER OF HYPOTHESES GENERATED DURING DECODING

Task (WACC%)	CSJ (77.2)		ODQA-1.8M (85.5)	
On-the-fly method	standard	proposed	standard	proposed
Hist. pruning param. M	1500	600	2000	1200
Real time factor	1.24	0.88	1.23	0.82
#hyp/frame	2434	863	2607	1251

per frame when using each composition method for the same word accuracy. The number of evaluated hypotheses means the number of hypotheses generated and weighted at each time frame, i.e., h' generated by $NewHyp(\cdot)$ in Figs. 6, 7, or 8. In both tasks, the proposed method actually evaluated fewer hypotheses to achieve the same word accuracy. This fact results in a difference of speed with unchanged accuracy.

2) In the decoder with the proposed algorithm, the amount of computation required for the acoustic likelihood still accounted for about 41% of the decoding in the CSJ task and 24% in the 1.8-million word ODQA task under a real time condition. Thus, the improvement in search efficiency provided by our algorithm should be greater than the improvement in the measured real time factor. If we exclude the computation of acoustic likelihoods in Table V, the real time factors of the standard and proposed algorithms in the CSJ task become 0.88 and 0.52, and those in the ODQA task become 1.03 and 0.62, respectively. Since no special mechanisms for the fast computation of acoustic likelihoods such as [25] are implemented in the decoder, larger differences in speed will be observed by incorporating such fast computation methods.
3) In both tasks, our method yielded almost the same accuracy as that of full composition and standard on-the-fly composition methods with wide beam settings. Thus, although our implementation includes an approximation, it seems to have no effect on word accuracy. However, the accumulated weight was sometimes slightly different from that of the other methods. This difference could arise from the declination of the time alignment. Thus, although our method yields satisfactory recognition results, it does not ensure the correct accumulated weight.
4) In some results in Fig. 11, the proposed method provided greater accuracy than the standard method. We analyzed the hypotheses and cohypotheses processed during decoding. In our implementation, no cohypothesis is pruned until the cohypothesis list to which it belongs is updated or merged, or the hypothesis delivering the list is pruned. This mechanism seems to be effective for avoiding search errors by beam pruning. When we changed our decoder so as to prune the cohypotheses at every frame, the word accuracy deteriorated. Accordingly, it is ineffective to prune the cohypotheses at every frame since it increases the amount of computation and the number of search errors.

VI. Conclusion

In this paper, we proposed a novel one-pass search algorithm with on-the-fly composition of WFSTs for LVCSR. In a spontaneous lecture speech transcription task with a 65k-word vocabulary, our proposed method outperformed not only standard on-the-fly composition, but also decoding based on a single WFST that was fully composed and optimized. In that task, our method needed only 38% of the memory required for decoding with a fully composed WFST. In addition, we achieved high-accuracy real-time speech recognition with an extremely large vocabulary of 1.8 million words. In decoding with this vocabulary size, the proposed algorithm was 1.5–3 times faster than the standard algorithm.

We performed a Viterbi search of our algorithm based on the first of the two WFSTs. The second WFST was only used to rescore the hypotheses generated during the search. Therefore, a narrower beam width can be applied in histogram pruning and this makes decoding faster. Although our algorithm includes an approximation in our practical implementation for speech recognition, we confirmed that recognition accuracy was not degraded by the approximation.

Since this new method is a general algorithm for the on-the-fly composition of WFSTs, it can be applied to both speech recognition and other applications. In the future, we would like to apply this technique to speech-input language processing including speech summarization [7], and speech understanding.

Acknowledgment

The authors would like to thank the Japanese Science and Technology Agency Priority Program, "Spontaneous Speech: Corpus and Processing Technology," for providing speech data and transcriptions. Furthermore, the authors would like to thank the MIT Spoken Language Systems Group for providing the tools for composing and optimizing the WFSTs. The authors used the SRI language modeling toolkit (SRILM) for shrinking the language models.

References

[1] C. Hori, T. Hori, H. Isozaki, E. Maeda, S. Katagiri, and S. Furui, "Deriving disambiguous queries in a spoken interactive ODQA system," in *Proc. ICASSP*, 2003, vol. I, pp. 624–627.
[2] M. Mohri, F. Pereira, and M. Riley, "Weighted finite-state transducers in speech recognition," *Comput. Speech Lang.*, vol. 16, pp. 69–88, 2002.
[3] M. Riley, F. Pereira, and M. Mohri, "Transducer composition for contextdependent network expansion," in *Proc. Eurospeech*, 1997, vol. 3, pp. 1427–1430.
[4] M. Mohri and M. Riley, "Weighted determinization and minimization for large vocabulary speech recognition," in *Proc. Eurospeech*, 1997, vol. 1, pp. 131–134.
[5] S. Bangalore and G. Riccardi, "A finite-state approach to machine translation," in *Proc. ASRU*, 2001, pp. 381–388.
[6] F. Casacuberta, "Finite-state transducers for speech-input translation," in *Proc. ASRU*, 2001, pp. 375–380.
[7] T. Hori, C. Hori, and Y. Minami, "Speech summarization using weighted finite-state transducers," in *Proc. Eurospeech*, 2003, pp. 2817–2820.
[8] M. Mohri, F. Pereira, and M. Riley, "A rational design for a weighted finite-state transducer library," in *Proc. Int. Workshop Implementing Automata 1997*, 1997, vol. 1436, Lecture Notes in Computer Science, pp. 144–158.
[9] H. J. G. A. Dolfing and I. L. Hetherington, "Incremental language models for speech recognition using finite-state transducers," in *Proc. ASRU*, 2001, pp. 194–197.
[10] D. Willett and S. Katagiri, "Recent advances in efficient decoding combining on-line transducer composition and smoothed language model incorporation," in *Proc. ICASSP*, 2002, vol. I, pp. 713–716.
[11] D. Caseiro and I. Trancoso, "Transducer composition for on-the-fly lexicon and language model integration," in *Proc. ASRU*, 2001, pp. 393–396.

[12] ——, "A tail-sharing WFST composition for large vocabulary speech recognition," in *ICASSP*, 2003, vol. I, pp. 356–359.
[13] S. Ortmanns, H. Ney, and X. Aubert, "A word graph algorithm for large vocabulary continuous speech recognition," *Comput. Speech Lang.*, vol. 11, pp. 43–72, 1996.
[14] A. Ljolje, F. Pereira, and M. Riley, "Efficient general lattice generation and rescoring," in *Proc. Eurospeech*, 1999, pp. 1251–1254.
[15] T. Shimizu, H. Yamamoto, H. Masataki, S. Matsunaga, and Y. Sagisaka, "Spontaneous dialogue speech recognition using cross-word context constrained word graphs," in *Proc. ICASSP*, 1996, pp. 145–148.
[16] M. Woszczyna and M. Finke, "Minimizing search errors due to delayed bigrams in real-time speech recognition systems," in *Proc. ICASSP*, 1996, pp. 137–140.
[17] L. Nguyen and R. Schwartz, "The BBN single-phonetic-tree fast-match algorithm," in *Proc. ICSLP*, 1998, pp. 1827–1830.
[18] R. Schwartz and S. Austin, "A comparison of several approximate algorithms for finding multiple (N-BEST) sentence hypotheses," in *Proc. ICASSP*, 1991, pp. 701–704.
[19] T. Hori, Y. Noda, and S. Matsunaga, "Improved phoneme-historydependent search method for large-vocabulary continuous-speech recognition," *IEICE Trans. Info. Syst.*, vol. E86-D, no. 6, pp. 1059–1067, 2003.
[20] T. Kawahara, H. Nanjo, T. Shinozaki, and S. Furui, "Benchmark test for speech recognition using the corpus of spontaneous Japanese," in *Proc. SSPR*, 2003, pp. 135–138.
[21] T. Hori, "NTT speech recognizer with outLook on the next generation: SOLON," in *Proc. Commun. Scene Anal.*, 2004.
[22] H. Isozaki *et al.*, "Efficient support vector classifiers for named entity recognition," in *Proc. COLING*, 2002, pp. 390–396.
[23] A. Stolcke, "Entropy-based pruning of backoff language models," in *Proc. DARPA Broadcast News Transcription and Understanding Workshop*, 1998, pp. 270–274.
[24] M. Saraclar, M. Riley, E. Bocchieri, and V. Goffin, "Towards automatic closed captioning: Low latency real time broadcast news transcription," in *Proc. ICSLP*, 2002, pp. 1741–1744.
[25] E. Bocchieri, "Vector quantization for the efficient computation of continuous density likelihoods," in *Proc. ICASSP*, 1993, vol. II, pp. 692–695.

Takaaki Hori (M'04) received the B.E. and M.E. degrees in electrical and information engineering from Yamagata University, Yonezawa, Japan, in 1994 and 1996, respectively, and the Ph.D. degree in system and information engineering from Yamagata University in 1999.

Since 1999, he has been engaged in research on spoken language processing at the Cyber Space Laboratories, Nippon Telegraph and Telephone (NTT) Corporation, Kyoto, Japan. He was a Visiting Scientist at the Massachusetts Institute of Technology, Cambridge, from 2006 to 2007. He is currently a Research Scientist in the NTT Communication Science Laboratories, NTT Corporation.

Dr. Hori is a member of the Institute of Electronics, Information, and Communication Engineers (IEICE) and the Acoustical Society of Japan (ASJ).

Chiori Hori (M'02) received the B.E. and M.E. degrees in electrical and information engineering from Yamagata University, Yonezawa, Japan, in 1994 and 1997, respectively, and the Ph.D. degree from the Graduate School of Information Science and Engineering, Tokyo Institute of Technology (TITECH), Tokyo, Japan, in 2002.

From April 1997 to March 1999, she was a Research Associate in the Faculty of Literature and Social Sciences, Yamagata University. In 2002, she was engaged in spoken language processing research in the NTT Communication Science Laboratories, Nippon Telegraph and Telephone Corporation (NTT), Kyoto, Japan. She is currently a Researcher at InterACT, Language Technology Institute, Carnegie Mellon University, Pittsburgh, PA.

Dr. Hori is a member of the Institute of Electronics, Information, and Communication Engineers (IEICE) and the Acoustical Society of Japan (ASJ).

Yasuhiro Minami (M'02) received the M.Eng. and Ph.D. degrees in electrical engineering from Keio University, Yokohama, Japan, in 1988 and 1991, respectively.

He joined Nippon Telegraph and Telephone Corporation (NTT), Tokyo, Japan, in 1991, where he worked on robust speech recognition. He was a Visiting Researcher at the Massachusetts Institute of Technology, Cambridge, from 1999 to 2000. Since February 2000, he has been with NTT Communication Science Laboratories, Kyoto, Japan. His interests include modeling for speech recognition.

Dr. Minami is a member of the Institute of Electronics, Information, and Communication Engineering (IEICE) and the Acoustical Society of Japan (ASJ).

Atsushi Nakamura (M'03–SM'07) received the B.E., M.E., and Dr.Eng. degrees from Kyushu University, Fukuoka, Japan, in 1985, 1987, and 2001, respectively.

In 1987, he joined Nippon Telegraph and Telephone Corporation (NTT), Kyoto, Japan, where he engaged in the research and development of network service platforms, including studies on application of speech processing technologies into network services at Musashino Electrical Communication Laboratories, Tokyo, Japan. From 1994 to 2000, he was with the Advanced Telecommunications Research (ATR) Institute, Kyoto, as a Senior Researcher, working on the research of spontaneous speech recognition, construction of spoken language database and development of speech translation systems. Since April 2000, he has been with the Communication Science Laboratories, NTT, Kyoto. His research interests include acoustic modeling of speech, speech recognition and synthesis, spoken language processing systems, speech production and perception, computational phonetics and phonology, and application of learning theories to signal analysis and modeling.

Dr. Nakamura is a member of the Institute of Electronics, Information, and Communication Engineering (IEICE) and the Acoustical Society of Japan (ASJ). He received the Best Paper Award from the IEICE in 2004 and the Telecom Technology Award from the Technology Advancement Foundation in 2006.