An Algorithm for Fast Composition of Weighted Finite-State Transducers

John McDonough[1,3], *Emilian Stoimenov*[2] *and Dietrich Klakow*[1]

[1]Spoken Language Systems, Saarland University, Saarbrücken, Germany
[2]Institute for Theoretical Computer Science, University of Karlsruhe, Karlsruhe, Germany
[3]Insitute for Intelligent Sensor-Actuator Systems, University of Karlsruhe, Karlsruhe, Germany

Abstract

In automatic speech recognition based on weighted-finite transducers, a static decoding graph $HC \circ L \circ G$ is typically constructed. In this work, we first show how the size of the decoding graph can be reduced and the necessity of determinizing it can be eliminated by removing the ambiguity associated with transitions to the backoff state or states in G. We then show how the static construction can be avoided entirely by performing *fast on-the-fly composition* of HC and $L \circ G$. We demonstrate that speech recognition based on this on-the-fly composition approximately 80% more run-time than recognition based on the statically-expanded network R, which makes it competitive compared with other dynamic expansion algorithms that have appeared in the literature. Moreover, the dynamic algorithm requires a factor of approximately seven less main memory as the recognition based on the static decoding graph.

1. Introduction

State-of-the-art large vocabulary continuous speech recognition systems use subword units consisting of phones to model the words of a language. As coarticulation effects are prevalent in all speech, a phone must be modeled in its context to achieve optimal performance. The relevant contexts are most often chosen with a *decision tree* based on a measure of goodness such as the likelihood or entropy of a training set.

As originally proposed by Mohri *et al* [16, 14], a *weighted finite-state transducer* (WFST) that translates phone sequences into word sequences can be obtained by forming the *composition* $L \circ G$, where L is a *lexicon* which translates the phonetic transcription of a word to the word itself, and G is a *grammar* or *language model* (LM) which assigns to valid sequences of words a weight consisting of the negative log probability of this sequence. In earlier work [12], we proposed an algorithm for constructing a transducer HC that translates from sequences of Gaussian mixture models directly to phone sequences.

The HC transducer must normally be composed with $L \circ G$ to construct the complete recognition network. The final search graph, even after optimization, quite typically consists of millions of states and tens of millions of states. Indeed, the size of the final recognition network, along with the necessity of determinizing it, typically limit the size of the grammar G that can be used.

In the recent past, a popular approach for reducing the amount of random access memory (RAM) required to store $R = HC \circ L \circ G$ during decoding has been to factor R into the components A and B in one of several ways, then combine the factored pieces *on-the-fly* during recognition. Such on-the-fly composition fits readily into the usual Viterbi search implemented as a token passing algorithm; for the on-the-fly composition of $R = A \circ B$ it is simply necesseary to store a pointer to one edge of A and one edge of B in each token, then consider states of R with the form (n_A, n_B) where n_a is a node of A and n_b is a node of B.

There are three primary problems to be solved in such a on-the-fly composition: Firstly, production of *noncoaccessible states*—i.e., states for which there is no *successful path* to an end state—must be avoided, as their expansion during decoding represents wasted computation and hence decreases the efficiency of the search. During static composition, such nodes can simply be *purged* as a post-processing step. Such an operation is not feasible for dynamic composition, as the entire network must be constructed in order to determine which nodes are noncoaccessible. Caseiro and Trancoso [2] introduced a method for avoiding the creation of such states during dynamic composition of L and G. Their method was recently extended by Cheng *et al* [3] for a more general class of WFSTs.

Secondly, *weight pushing*, whereby the weights introduced by the grammar G and lexicon L are pushed as far towards the initial node as possible in order to maximize the efficiency of the search [15], must be approximated during dynamic composition. Caseiro and Trancoso [2] introduced a suitable approximation for weight pushing, which was also extended by Cheng *et al* [3]. We consider a further refinement of their technique in this work.

Thirdly, the factors A and B to be composed on-the-fly must be defined. Different approaches to this problem have appeared in the literature. In [2], $B = G$ was separated entirely from the other components. In [5] and [19], G was separated into an *incremental* LM $B = G_i$ as well as a *smearing* LM G_s which was statically composed with the other components to form $A = HC \circ L \circ G_s$. Finally, in [9], all components were dynamically composed during recognition.

In this work, we first show how the size of the static recognition network $R = HC \circ L \circ G$ can be reduced and the necessity of determinizing it can be eliminated by removing the ambiguity associated with transitions to the backoff state or states in G. This is in fact similar to the approach suggested in [16]. We then show how the static construction of $HC \circ L \circ G$ can be avoided entirely by performing fast on-the-fly composition of $A = HC$ and $B = L \circ G$. Experiments with our initial implementation of the on-the-fly composition algorithm indicate it requires approximately 80% more run-time than recognition based on the statically-expanded network R, which makes it competitive compared with other dynamic expansion algorithms that have appeared in the literature [9, 2, 3]. As mentioned above, the key to achieving this efficiency is eliminating the expansion of dead end nodes during search; we also present

This work was sponsored by the German Ministry of Research and Technology (BMBF) under the *SmartWeb* project, grant number 01IMD01A.

978-1-4244-1746-9/07/$25.00 ©2007 IEEE

an algorithm for the latter that works for the most general class of WFSTs. Moreover, the loss of efficiency is offset by the fact that the dynamic algorithm requires a factor of approximately seven less main memory as the static algorithm.

The balance of this work is organized as follows. In Section 2, we briefly review the definition of a weighted finite-state transducer, then consider conventional static composition and along with the other similarity transformations typically used to build a static decoding graph. Section 3 shows how the size of the decoding graph can be reduced and the necessity of determinizing can be eliminated by explicitly modeling the transitions to the back-off node or nodes in G. The fast on-the-fly composition algorithm is discussed in Section 4, along with a technique for eliminating the expansion of dead-end paths during dynamic decoding. Section 5 then shows how weight pushing, a standard operation used in constructing a static decoding graph, can be simulated during dynamic decoding. The results of a set of timing studies with the static and dynamic decoding algorithms is presented in Section 6, along with statistics related to network size. In the final section, we present our conclusions and plans for future work.

2. Static Network Construction

Let us begin with formal definitions for the semiring and the weighted finite-state transducer (WFST).

Definition 2 *A* semiring $K = (\mathbb{K}, \oplus, \otimes, \bar{0}, \bar{1})$ *consists of a set* $\mathbb{K}$*, an associative and commutative operation* $\oplus$*, an associative operation* $\otimes$*, the identity* $\bar{0}$ *under* $\oplus$*, and the identity* $\bar{1}$ *under* $\otimes$*. By definition,* $\otimes$ *distributes over* $\oplus$ *and*

$$\bar{0} \otimes a = a \otimes \bar{0} = \bar{0}$$

Definition 3 *A* weighted finite-state transducer *(WFST)* $T = (\Sigma, \Omega, Q, E, i, F, \lambda, \rho)$ *on the semiring* $\mathbb{K}$ *consists of an* input alphabet Σ*, an* output alphabet Ω*, a set of states* Q*, a set of transitions*

$$E \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times (\Omega \cup \{\epsilon\}) \times \mathbb{K} \times Q,$$

a start state $i \in Q$*, a set of end states* $F \subseteq Q$*. A* transition $t = (p[t], l_i[t], l_o[t], w[t], n[t]) \in E$ *consists of a previous state* $p[t]$*, a next state* $n[t]$*, an input symbol* $l_i[t]$*, an output symbol* $l_o[t]$*, and a weight* $w[t]$*.*

When referring the sets of states or arcs of more than one transducer, we will distinguish among them with suitable subscripts; e.g., Q_S and E_S are respectively the sets of states and edges of transducer S.

Next we briefly describe conventional weighted composition.

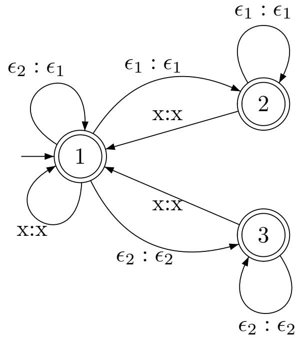
Definition 4 *Consider a transducer* S *which maps an input string* u *to an output string* v *with a weight of* w_1*. Consider also a transducer* T *which maps input string* v *to output string* y *with weight* w_2*. The* composition

$$R = S \circ T$$

of S *and* T *maps string* u *directly to* y *with weight*

$$w = w_1 \otimes w_2.$$

In the absence of ϵ–transitions, the construction of such a composition is straightforward. It entails simply pairing the output symbols on the transitions of a node $n_S \in E_S$ with the input symbols on the transitions of a node $n_T \in E_T$, beginning with the initial nodes $i_S \in Q_S$ and $i_T \in Q_T$. Each $n_R \in Q_R$ is uniquely determined by the pair (n_S, n_T). The pairing of the transitions of n_S with those of n_T is *local*, inasmuch as it only entails the consideration of two nodes at a time. As R is constructed, it can so happen that nodes are created that do not lie on a successful path; i.e., from such a node, there is no path to an $n \in F_R$. As mentioned in the introduction, such nodes are typically purged from the graph after static composition. It is worth noting, however, that this purge step is *not* a local operation inasmuch as it is necessary to consider the entire transducer R in order to determine if any given node is on a successful path. We will return to this point in Section 4.



$\epsilon_1 : \epsilon_1$
$\epsilon_2 : \epsilon_1$
$\epsilon_1 : \epsilon_1$
x:x
2
1
x:x
3
x:x
$\epsilon_2 : \epsilon_2$
$\epsilon_2 : \epsilon_2$

Figure 1: Filter used during composition with ϵ–symbols, after Pereira and Riley [17].

When ϵ–symbols are introduced, composition becomes more complicated, as it is necessary to specify how and if an ϵ–symbol on the output of a transition in n_S can be combined with an ϵ–symbol on the input of n_T. In such a case, a node $n_R \in R$ is specified by a triple (n_S, n_T, f), where $f \in \{0, 1, 2\}$ is an index indicating the *state* of the composition filter. This filter is necessary to ensure that the resulting transducer $R = S \circ T$ satisfies the mathematical definition of composition [17]. Such a filter is shown in Figure 1. Essentially, the filter ensures that an ϵ–transition in A is only paired with an ϵ–transition in B when the filter is in State 1. If the filter is in State 2 or 3, then a non–ϵ symbol in A must be matched with a non–ϵ in B in order to return to State 1 before further ϵ–symbols from A and B can be paired together [17].

In prior work, we have constructed a complete recognition network R through the series of operations

$$R = \min \operatorname{push} \det(\min \det(HC) \circ \det(L \circ G)) \qquad (1)$$

This construction sequence is similar to the approach originally suggested in [16] as well as those used by others [2, 3]. The principal difference in our approach is that H, the hidden Markov model transducer (HMM), and C, the context-dependency transducer, are not constructed separately. Rather, as mentioned in the introduction, there is a single sequential transducer HC that maps from sequences of Gaussian mixture models (GMMs) directly to sequences of phones.

3. Reducing Static Network Size

Consider now the following definition and related theorem due to Mohri [13].

Definition 5 *A* sequential transducer *is deterministic and has no transitions with* ϵ *as input.*

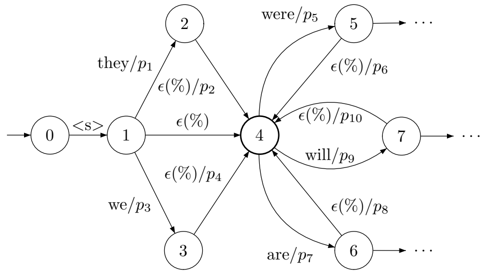


Figure 2: Simple grammar G.

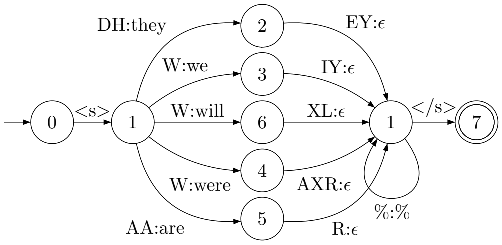


Figure 3: Lexicon L.

Theorem 6 (Mohri) *The composition of two sequential transducers is sequential.*

While simple, Mohri's theorem has profound theoretical and practical implications. First of all, consider the network construction sequence specified in (1). By far, the most resource intensive operation in terms of both computation and main memory is the determinization after the composition of $\min\det(HC)$ and $\det(L \circ G)$. According to Mohri's theorem, if both $\min\det(HC)$ and $\det(L \circ G)$ were sequential, this determinization could be eliminated entirely. This poses no problem for the context dependency transducer $\min\det(HC)$, as its construction ensures it is sequential. More problematic is $L \circ G$ because ϵ–transitions in G are typically used to enable transitions to the back-off node. While the ϵ–transitions can be removed with the ϵ–removal algorithm, this can lead to a massive increase in the number of transitions in the network.

Consider then the following modifications to the several transducers. Replace the ϵ-transitions in G with a *back-off symbol* %. Then, at the end of each word sequence in L, add a self-loop with % as input and output. Similarly, to the end of each three-state sequence in HC, add a self-loop with % as input and output. With these modifications, the $L \circ G$ component can be constructed according to

$$B = \det(\epsilon\text{–removal}(L \circ G)) \tag{2}$$

As the ϵ–transitions have now been removed from G, the ϵ–removal operation on $L \circ G$ does not cause a massive increase in the number of transitions in the model, as the only ϵ–transitions remaining in $L \circ G$ before removal stem from the ϵ–transition from last state of each word transcription in L to the "branch" state, from which all word transcriptions begin.

Consider the grammar G and lexicon L shown in Figures 2 and 3, respectively. In G, transitions to the back-off node, State 4, are labeled either ϵ or with %. Moreover, State 1 in L has a self-loop labeled with % on both input and output. Figures 4 and 5 display the resulting transducer when the %– and non–%–versions of G are composed with L and ϵ–removal is applied to the result. Clearly using the explicit back-off symbol % produces a composition $L \circ G$ with fewer transitions after ϵ–removal.

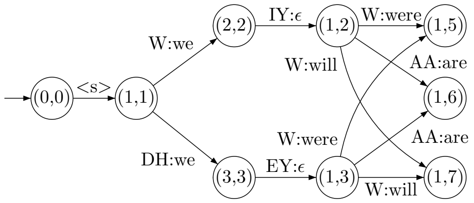


Figure 4: Result of composing L and G with no explict backoff symbol.

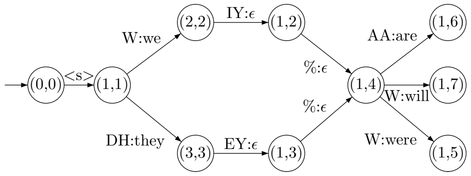


Figure 5: Result of composing L and G with an explict backoff symbol %.

As before, the HC component can be constructed according to

$$A = \min\det(HC) \tag{3}$$

Then the complete recognition network is obtained from

$$R = \min\text{push}(A \circ B) \tag{4}$$

As a final operation, the word boundary and back-off symbols are removed from R prior to its use in recognition. Dimensions for two decoding graphs based on shrunken and full trigram language models and constructed as described here are given in Table 1.

4. Fast On-the-Fly Composition

While the algorithm described in the last section is undoubtedly beneficial in terms of reducing the size of the final recognition network, the network obtained using the full WSJ language model still had nearly 50 million states and over 100 million edges. Decoding with this network could only be performed on a 64-bit workstation, and the size of the network in memory was nearly 7 Gb, which can be prohibitive even for research purposes, and impossible on platforms having only more modest memory. In this section, we consider a method whereby the such enormous memory requirements can be reduced by *eliminating* the static expansion of the final network.

Consider again the composition of A and B as defined in (4). As B has no ϵ–symbols on the input side, each node $n_R \in Q_{A \circ B}$ is uniquely defined by the pair (n_A, n_B), where $n_A \in Q_A$ and $n_B \in Q_B$. This implies that the complexity of the general composition algorithm introduced by the ϵ–symbols has been eliminated; B has no ϵ–symbols on the input, and an ϵ–symbol on the output side of A can always be taken. As R is typically many times larger than A and B prior to their composition, and remains so even after determinization, pushing, and minimization, we are led to consider the following strategy to reduce memory requirements. First of all, let us redefine B as

$$B = \min\text{push}\det(\epsilon\text{–removal}(L \circ G)) \tag{5}$$

Now, instead of *statically* composing A and B, we perform decoding with the *on-the-fly-composition* algorithm [9]. For the

Language Model	G		$HC \circ L \circ G$	
	Bigrams	Trigrams	Nodes	Arcs
Shrunken Trigram	431,131	435,420	14,187,005	32,533,593
Full Trigram	1,639,687	2,684,151	49,082,515	114,304,406

Table 1: Sizes of shrunken and full trigram language models and decoding graphs.

sequential A and B transducers considered here, the latter is a straightforward modification of the token passing algorithm whereby each token maintains a pointer to an edge in both A and B, and each active hypothesis is associated with a state $n_R = (n_A, n_B) \in Q_R$.

There remains one further problem to be solved. As mentioned in Section 2, nodes can be formed during the on-the-fly composition of A and B that *do not* lie on a successful path. During static composition, such nodes are purged. Expanding the set of active hypotheses across transitions from nodes that are not on successful paths during on-the-fly composition would clearly result in wasted computation. Here we propose a novel solution for eliminating the expansion of such non-coaccessible nodes.

Definition 7 *A node is* white *iff all of its outgoing transitions are on successful paths. A node is* black *iff none of its outgoing transitions are on successful paths. A node is* gray *iff it has at least one transition on a successful path and at least one transition that is not on a successful path.*

We now state a simple theorem with deep implications.

Theorem 8 *All paths from the initial node $i \in R$ to a black node must go through a gray node.*

Proof: Without loss of generality, the initial node i can be assumed to be white. Assuming that a path from the initial node to a black node would never cross a gray node leads immediately to a contradiction with the definition of a white node. □

Now we state a definition and another theorem.

Definition 9 *The* fence $\mathcal{F}$ *is that subset of black nodes that can be reached by a single transition from a gray node.*

Theorem 10 *All paths from i to a black node must cross a node $n_F \in \mathcal{F}$.*

Proof: The claim follows as a corollary of Theorem 8. □

Theorem 10 clearly implies that in order to *eliminate*, and not simply *avoid*, the expansion of any black node during on-the-fly composition, we need not store the indices of *all* black nodes, but rather only the indices of the fence nodes. Then, as soon as a fence node is encountered, the associated hypothesis is not expanded further.

The fence can be found as follows. Starting from the initial node $i_R = (i_A, i_B)$, perform a breadth first search (BFS) to discover all nodes in the set $\mathcal{A}$ that are *accessible* from i_R as well as the end nodes $F_R \subset \mathcal{A}$. Now reverse both A and B and searching backwards from each $n_F \in F_R$, discover all nodes in the set $\mathcal{C}$ that are both accessible from i_R and *coaccessible* from F_R. Clearly the set of black nodes is then $\mathcal{B} = \mathcal{A} - \mathcal{C}$. Now a third BFS can be conducted to discover the gray nodes, and therewith the fence $\mathcal{F}$. Although *all* accessible nodes in R must indeed be visited, it is only necessary to store the pair of indices $(n_A, n_B) \in R$ in order to identify it uniquely, given that, as previously mentioned, the composition filter is *not* required. Moreover, it is *unnecessary* to store the adjacency list of any node, given that a node will be visited *at most* three times during the fence construction procedure and the adjacency list can be efficiently regenerated with each visit. Hence, the fence construction procedure requires far less RAM than would be necessary to store the network R in memory were it fully expanded.

5. Dynamic Weight Pushing

Caseiro and Trancoso [2] proposed a technique for composing a dictionary transducer L with a grammar transducer G in which the arcs of L are annotated with supplemental information about the next non–ϵ–transitions that will be encountered during network transversal. This "lookahead" information was used both to avoid the creation of non-coaccessible states as well as to provide for dynamic weight pushing.

While we eliminate the creation of non-coaccessible states through use of the fence as described above, it is still useful to consider a scheme for dynamic weight pushing based on lookahead information. We used the following variation on the technique described by Caseiro and Trancoso [2]. First, we annotated each *node* n in HC with supplemental information about the next non–ϵ–transitions that would be encountered in a graph transversal beginning from n. This was achieved by initiating a depth first search from each node n in HC, which proceeded until a non-ϵ output symbol was encountered on a transition. Then during search, for a partial hypothesis terminating in state $(n_A, n_B) \in Q_R$, the next non–ϵ list in n_A was compared with the adjacency list of n_B to find the intersecting set of symbols. Of all transitions in the adjacency list of n_B with an input symbol in this intersection, the arc with the minimum weight was chosen and this minimum weight was added to the language model score of the partial hypothesis ending in state $(n_A, n_B) \in Q_R$. The token associated with each partial hypothesis also included the value of this pushed weight. When this value of pushed weight was greater than zero, the difference between the minimum weight in the intersecting set and the amount of weight already pushed was added to the LM score. In this manner, we ensured that the correct total LM weight would eventually be added to the partial hypothesis, at latest when the search advanced across the next arc in B containing the actual symbol. When this symbol was encountered, the remaining balance of the weight was added to the partial hypothesis and the value of the pushed weight was reset to zero.

The dynamic weight pushing algorithm is illustrated in Figure 6. Initially a token is created with a pointer to the transition entering State 0 in HC, the upper graph, and the transition entering State 1 in $L \circ G$, the lower graph. For clarity, we do not show the self-loops that are normally present in HC. Beginning from State 0 in HC, the output symbols "M", "N", "P" and "T" can be reached. Of the transitions labeled with these symbols in $L \circ G$, the transition with input "M" has the minimum weight of 5. Hence, 5 is pushed onto the partial hypothesis in HC and we set $p = 5$. From State 1, only "N", "P" and "T" can be reached, and of the transitions in HC labeled with these sym-

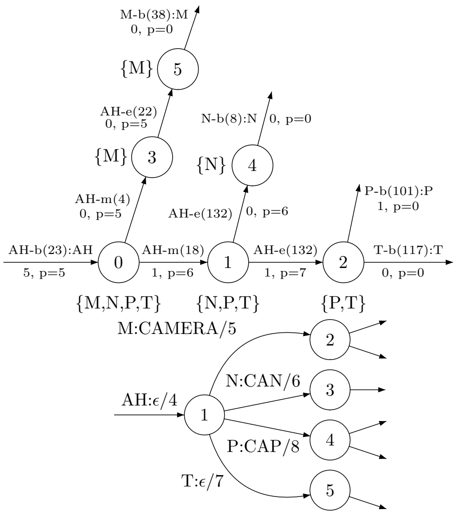


Figure 6: Illustration of dynamic weight pushing.

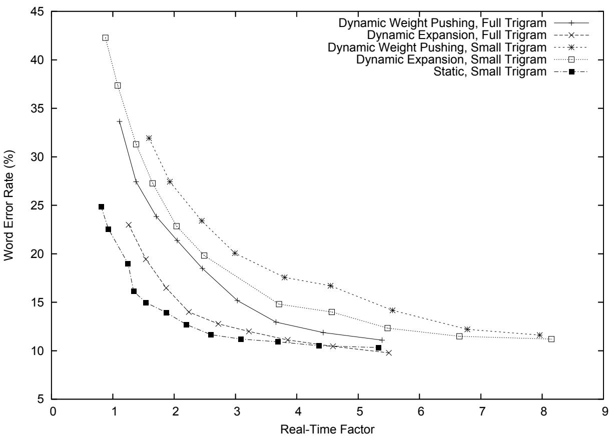


Figure 7: Word error rate vs. real-time factor for the static decoder as well as the dynamic expansion decoder both with and without dynamic weight pushing.

bols, that transition with input "N" has the minimal weight of 6. As $p = 5$ from the prior transition, we can push an additional weight of $6 - 5 = 1$ onto the partial hypothesis, set $p = 6$, and advance from State 0 to State 1. From State 2 in HC only "P" and "T" can be reached, which have weights 8 and 7 respectively in $L \circ G$. Thus in advancing from State 1 to State 2 an additional weight of $7 - 6 = 1$ is pushed onto the partial hypothesis and we set $p = 7$. In leaving State 2, no weight is pushed onto the hypothesis associated with "T" and a weight of $8 - 7 = 1$ is pushed onto the hypothesis associated with "P". In both cases, we set $p = 0$ as the actual phone symbols have now been matched and we are ready to begin the next round of lookahead with the transitions leaving States 4 and 5 in $L \circ G$.

6. Experiments

After beamforming, the feature extraction of our ASR system was based on cepstral features estimated with a warped *minimum variance distortionless response* [20] (MVDR) spectral envelope of model order 30. Due to the properties of the warped MVDR, neither the Mel-filterbank nor any other filterbank was needed. The warped MVDR provides an increased resolution in low–frequency regions relative to the conventional Mel-filterbank. The MVDR also models spectral peaks more accurately than spectral valleys, which leads to improved robustness in the presence of noise. Front-end analysis involved extracting 20 cepstral coefficients per frame of speech and performing global cepstral mean subtraction (CMS) with variance normalization. The final features were obtained by concatenating 15 consecutive frames of cepstral features together, then performing a *linear discriminant analysis* (LDA) to obtain a feature of length 42. The LDA transformation was followed by a second global CMS, then a global STC transform [8].

The training data used for the experiments reported here was taken taken from the WSJ0 and WSJCAM0 [6] sets, for a total of 40 hours of training material. Acoustic models estimated with two different HMM training schemes were used for several decoding passes: conventional maximum likelihood (ML) HMM training [4, §12], and speaker-adapted training under a ML criterion (ML-SAT) [1]. Our baseline system was fully continuous with 3,500 codebooks and a total of 180,656 Gaussian components.

The experimental results described here were generated after four decoding passes on the close-talking microphone from the Speech Separation Challenge Part 2 development set. The individual decoding passes are also described in [10]. The fourth decoding pass was based on *vocal tract length normalization* [18] (VTLN), *maximum likelihood linear regression* [11] (MLLR), and *constrained maximum likelihood linear regression* [7] (CMLLR). These parameters were estimated based on the word lattices from the third pass.

Figure 7 shows the results of a set of timing studies based on the static decoder as well as the dynamic expansion decoding algorithm described in Section 4 both with and without the dynamic weight pushing algorithm presented in Section 5. We used two language models for these experiments: the shrunken trigram and the full trigram whose dimensions are given in Table 1. The static network was based on the small trigram.

The results in Figure 7 indicate that the dynamic expansion algorithm requires approximate 80% more execution time than the static algorithm when the same language model is used. They also show that, disappointingly, the dynamic weight pushing algorithm does not decrease the execution time, much the opposite in fact. We attribute this to the fact that our initial implementation of the on-the-fly composition algorithm was based on matching the output symbols of HC with the input symbols of $L \circ C$ through a linear search over the adjacency lists that had been appropriately sorted. Greater efficiency could undoubtedly be achieved if this linear search were replaced with a hash table access, as the output symbols appearing in the adjacency list of any given node in HC are typically very sparse due to the nature of the decision tree used in its construction. The relative efficiency of the dynamic search algorithm with respect to its static counterpart can be greatly improved by using the full trigram language model, as indicated by the fact that for relatively wide beams, the word error rate (WER) achieved by the dynamic decoder with the full trigram is actually lower than that achieved by the static decoder with the shrunken trigram.

Shown in Table 6 are the task image sizes for the static and dynamic decoders. As is clear from the tabulated sizes, when the same small trigram is used for both the static and dynamic decoders, the dynamic decoder requires a factor of approximately seven less RAM. This enables the full trigram to be applied during dynamic decoding, which requires less than 500 Mb. The full trigram could not be used with static decoder on the 32-bit machines used for these experiments re-

	Static	Dynamic	
Beam	Small Trigram	Small Trigram	Full Trigram
120.0	1380	179	470
130.0	1381	182	473
140.0	1384	187	476
155.0	1389	200	485

Table 2: Task image sizes in Mb for the static and dynamic expansion decoders at various beam settings.

ported here. On a work station with a 64-bit operating system, 7 Gb of RAM were required merely to load the decoding graph built from the full trigram, and the entire task image was approximately 8 Gb. It is interesting to note that the size of the task image for the dynamic decoder with the full trigram is nearly a factor of three smaller than that of the static decoder with the shrunken trigram.

7. Conclusions and Future Work

In this work, we have proposed an algorithm for dynamically composing HC, which maps from sequences of Gaussian mixtures to phones, and $L \circ G$, which maps from sequences of phones to sequences of words. We also showed how a structure dubbed the fence can be found that prevents non-coaccessible nodes from being generated during dynamic composition, and thereby reducing the efficiency of the search. With respect to decoding based on the static graph $HC \circ L \circ G$, the dynamic algorithm requires approximately 80% more execution time. We found that the dynamic composition algorithm actually became more accurate than the static decoding algorithm at a given real-time factor when a larger trigram language model was used with the former. The large trigram LM could not be used with the static decoder because the resulting decoding graph would not fit into the the main memory of our 32-bit work stations.

We proposed here a refinement of the dynamic weight pushing algorithms that have appeared previously in the literature. We were, however, unable to achieve any reduction in execution time by using such a dynamic weight pushing algorithm. We attribute this to the fact that the static weight pushing over $L \circ G$ achieves a nearly optimal distribution of weights, so that further dynamic pushing over HC yields no significant advantage. It is also likely that replacing the linear search used to match the output symbols of HC with the input symbols of $L \circ G$ with a hash access would greatly improve the efficiency of both the dynamic weight pushing as well as the on-the-fly composition itself. Modifying our implementation to include such a dynamic hash access will be a primary objective in the future.

8. References

[1] T. Anastasakos, J. McDonough, R. Schwarz, and J. Makhoul. A compact model for speaker-adaptive training. In *Proc. ICSLP*, pages 1137–1140, 1996.

[2] D. Caseiro and I. Trancoso. A specialized on-the-fly algorithm for lexicon and language model composition. *IEEE Trans. Audio, Speech and Language Processing*, 14(4), 2006.

[3] Octavian Cheng, John Dines, and Mathew Magimai Doss. A generalized dynamic composition algorithm of weighted finite state transducers for large vocabulary speech recognition. In *Proc. ICASSP*, 2007.

[4] J. Deller, J. Hansen, and J. Proakis. *Discrete-Time Processing of Speech Signals*. Macmillan Publishing, New York, 1993.

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

[6] Jeroen Fransen, Dave Pye, Tony Robinson, Phil Woodland, and Steve Young. WSJCAM0 corpus and recording description. Technical Report CUED/F-INFENG/TR.192, Cambridge University Engineering Department (CUED), Speech Group, Trumpington Street, Cambridge CB2 1PZ, UK, Sept. 1994.

[7] M. J. F. Gales. Maximum likelihood linear transformations for HMM-based speech recognition. *Computer Speech and Language*, 12, 1998.

[8] M. J. F. Gales. Semi-tied covariance matrices for hidden Markov models. *IEEE Transactions Speech and Audio Processing*, 7:272–281, 1999.

[9] T. Hori and A. Nakamura. Generalized fast on-the-fly composition algorithm for WFST-based speech recognition. In *Proc. Interspeech*, 2005.

[10] Kenichi Kumatani, Uwe Mayer, Tobias Gehrig, Emilian Stoimenov, John McDonough, and Matthias Wölfel. Minimum mutual information beamforming for simultaneous active speakers. In *Proc. ASRU*, submitted, 2007.

[11] C. J. Leggetter and P. C. Woodland. Maximum likelihood linear regression for speaker adaptation of continuous density hidden markov models. *Computer Speech and Language*, 9:171–185, Apr. 1995.

[12] J. McDonough, Matthias Wölfel, and Emilian Stoimenov. On maximum mutual information speaker-adapted training. *Computer Speech and Language*, to appear.

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

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

[15] M. Mohri and M. Riley. A weight pushing algorithm for large vocabulary speech recognition. In *Proc. ASRU*, pages 1603–1606, Aarlborg, Denmark, Sep. 2001.

[16] M. Mohri, M. Riley, D. Hindle, A. Ljolje, and F. Periera. Full expansion of context-dependent networks in large vocabulary speech recognition. In *Proc. ICASSP*, volume II, pages 665–668, Seattle, 1998.

[17] F. Pereira and M. Riley. Speech recognition by composition of weighted finite automata. In E. Roche and Y. Schabes, editors, *Finite-State Language Processing*, pages 431–453. MIT Press, Cambridge, MA, 1997.

[18] L. Welling, H. Ney, and S. Kanthak. Speaker adaptive modeling by vocal tract normalization. *IEEE Trans. Speech Audio Proc.*, 10(6):415–426, 2002.

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

[20] M.C. Wölfel and J.W. McDonough. Minimum variance distortionless response spectral estimation: Review and refinements. *IEEE Signal Process. Mag.*, 22(5):117–126, Sept. 2005.