A Specialized On-the-Fly Algorithm for Lexicon and Language Model Composition

Diamantino Caseiro, *Member, IEEE,* and Isabel Trancoso, *Senior Member, IEEE*

***Abstract*—This paper presents an algorithm for the composition of weighted finite-state transducers which is specially tailored to speech recognition applications: it composes the lexicon with the language model while simultaneously optimizing the resulting transducer. Furthermore, it performs these computations "on-the-fly" to allow easier management of the tradeoff between offline and online computation and memory. The algorithm is exact for local knowledge integration and optimization operations such as composition and determinization. Minimization and pushing operations are approximated. Our results have confirmed the efficiency of these approximations.**

***Index Terms*—Speech recognition, weighted finite-state transducers (WFSTs).**

I. Introduction

LARGE vocabulary continuous speech recognition is a difficult task, which is the reason why state-of-the-art systems use multiple sources of linguistic knowledge to better solve the problem. Those knowledge sources refer to various modeling levels, such as the acoustic, phonetic, lexical, and syntactic levels. Each of these sources adds to the complexity of the decoding algorithm, which is why the integration of new ones can be problematic. Frequently, new or uncommon knowledge sources are not tightly integrated in the main algorithm, but are rather used in later stages of processing to help choose the best sentence, from alternatives provided by the main algorithm.

At AT&T Laboratories, an approach was pioneered which greatly simplified the integration of knowledge sources in speech recognition [1]. The approach is based on the notion of weighted finite-state transducers (WFSTs). A WFST is a finite-state machine which allows the modeling of weighted relations. The approach consists of using WFSTs to model each knowledge source. The corresponding WFSTs are then combined using composition [2] and other general algorithms [3], [4] into a single WFST representing the search network, that is used by a Viterbi decoding algorithm [4]–[6]. This approach simplifies the integration of new knowledge sources, since the decoder is now independent of the particular knowledge sources. The problem thus shifted from the implementation of the recognition algorithm to the arguably simpler problem of finding suitable WFST representations. An additional advantage of the approach is that very effective optimizing algorithms, such as determinization [7], minimization [8], and pushing [9] can be applied to the combined WFST, allowing the flexible development of efficient systems, performing on par with or even surpassing traditional ones.

However, there are several problems with the use of WFST techniques in speech recognition. The first problem that we encountered is that a fundamental WFST optimization algorithm—determinization—requires a large quantity of memory relative to the size of the WFST being optimized. This problem imposes a limit on the size (and consequently, on the quality) of the systems which can be built. In the future, with the availability of more powerful hardware, this problem may lose relevance, since it only affects the development environment, nevertheless it was a major obstacle in our adoption of WFST techniques. The second problem, also memory related, is that the WFST resulting from the compilation of all knowledge sources, even after extensive optimization, can be very large, thus requiring large amounts of memory in runtime. This is a more serious problem, specially if our long-term goal is the ubiquity of speech interfaces in computers and electronic devices, where the speech recognition component should not be substantially more expensive than a keyboard. The third problem is related to the use of dynamic knowledge sources: usually, WFST techniques rely on generic algorithms for optimizing a single WFST network representing all knowledge sources in the system. This optimization is computationally expensive and is done offline. This means that the original knowledge sources are not available in runtime. Hence, it may be troublesome to preserve the optimality of the network, when dynamically adjusting the knowledge sources. Examples of adjustments include adaptation of the language model probabilities or addition of new words or pronunciations to the lexicon.

The solution for the second problem consists of extracting linear sequences of states from the optimized WFST, thus obtaining a much smaller network. The edges of this network are "on-the-fly" expanded into extracted sequences in runtime [6]. Other solutions, which also address the first problem, consist of factorizing the language model into a small model and a difference model, such that the original model is the composition of these two models. The small language model is integrated with the other knowledge sources in a network that is optimized offline. Information from the full language model is integrated in runtime by composing the network with the difference language model "on-the-fly." Variations of this approach were proposed by [10]–[12] the last two restricting the small language model to a unigram.

Manuscript received December 5, 2003; revised April 6, 2005. This work was supported in part by an FCT scholarship under PRAXIS XXI/BD/15836/98 and by FCT project POSI/PLP/47175/2002. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Ramesh A. Gopinath.

The authors are with the Spoken Language Systems Laboratory, INESC ID, Lisboa,Portugal, and also with the Instituto Superior Técnico, 1000-029 Lisboa, Portugal (e-mail: Diamantino.Caseiro@inesc-id.pt; Isabel.Trancoso@inesc-id.pt).

Digital Object Identifier 10.1109/TSA.2005.860838

This paper proposes an alternative approach to these problems. It is based on a specialized WFST composition algorithm which is especially tailored to integrate the lexicon with the language model; it composes both while simultaneously optimizing the resulting transducer. The algorithm can be used "on-the-fly" in runtime while preserving the original knowledge sources used. Compared to the referred "on-the-fly" algorithms, it does not require changes to the language model and uses its exact probabilities as soon as possible when performing language model lookahead.

In the rest of this paper, we introduce the basic idea behind our algorithm (Section II), present how the various optimization operations were developed (Sections III–V), and address how the algorithm can be applied to state-level optimization and context-dependency (Section VI). The two last sections present the results of our algorithm applied to large vocabulary recognition tasks for European Portuguese and our conclusions.

II. Composition of the Lexicon With the Language Model

A. Notation

For the sake of establishing our notation, let us define a *weighted finite-state transducer* over a semiring K as a tuple $T = (\Sigma, \Delta, Q, i, F, E, \lambda, \rho)$, where Σ is the finite input alphabet of the transducer, Δ is the finite output alphabet, Q is a finite set of states, $i \in Q$ is the initial state, $F \subseteq Q$ is the set of final states, $E \subseteq Q \times (\Sigma \cup \epsilon) \times (\Delta \cup \epsilon) \times Q \times K$ is a finite set of edges, and $\rho : F \rightarrow K$ is the final weight function mapping F to K. *Weights* are defined as members of a semiring. A *semiring* is a system $(K, \oplus, \otimes, \bar{0}, \bar{1})$ where K is a set of elements, and $\oplus$ and $\otimes$ are binary operations on K, with the following properties: $(K, \oplus, \bar{0})$ is a monoid with identity element $\bar{0}$, that is, it is closed under $\oplus$, $\oplus$ is associative, and $\bar{0}$ is an identity; $(K, \otimes, \bar{1})$ is a monoid with identity element $\bar{1}$; $\oplus$ is commutative; $\otimes$ distributes over $\oplus$; and $\bar{0}$ is its annihilator. In speech processing, two weight semirings are particularly useful: the *log probability* semiring $(\Re, \oplus_l, +, \infty, 0)$, where $\oplus_l$, defined as $a \oplus b = -\log(e^{-a} + e^{-b}))$, is the counterpart of the probability semiring $([0, 1], +, \times, 0, 1)$ in the logarithmic domain; and the *tropical semiring* $(\Re, \min, +, \infty, 0)$, often used as an efficient approximation to the log probability one.

B. Basic Idea

We base our algorithm for the lexicon and language model integration on the fact that the composition of sequential transducers is a sequential transducer [7]. This result is a straightforward consequence of the definitions of sequential transducer and transducer composition, and allows us to avoid the explicit determinization of the composition transducer, when the lexicon and the language model are represented as sequential transducers. A particularly useful aspect of this theorem is that composition can be efficiently implemented "on-the-fly."

Of course, real lexicon and language models are often not sequential. In the case of lexica, a source of ambiguity are homophone words or word sequences. In the case of language models, ambiguity may arise from finite-state approximations to n-gram language models or from class-based n-gram language model. When such ambiguous WFSTs are used, their composition might not be determinizable. In this case, the WFST determinization algorithm may not terminate [7], and disambiguating labels need to be inserted in the respective WFSTs using either heuristic [13] or principled [4] techniques so that a deterministic composition can be obtained. These labels are removed from the resulting transducer which is then no longer sequential, but nevertheless close enough for speech recognition purposes.

TABLE I
Toy Lexicon

Word	Phone Seq.		
AMO	a	m	o
ATO	a	t	o
OLA	o	l	a
OTO	o	t	o

In the following presentation, we will assume that both the lexicon loop and the language model are sequential. When they are not, the referred disambiguation techniques can be used to enforce sequentially. The proposed algorithm will still terminate and compute a correct composition transducer when the input WFSTs are not sequential, and generally, the result will be efficient for speech recognition, but not necessarily sequential.

To exemplify how the composition of sequential transducers can be used to obtain a deterministic composition transducer, we will use the toy lexicon listed in Table I, which can be represented by the sequential transducer L^* shown in Fig. 1, and the sequential language model G representing two single word sentences listed in Table II and also shown in Fig. 1. We assume that the lexicon WFST has a loop topology where the initial state is also the final state, here shown twice for clarity. In the figures, the symbol "_" represents an ϵ.

By definition, each state of the composed WFST corresponds to a pair of states from the argument transducers. For example, state (1,0) corresponds to state 1 of L^* and to state 0 of G.

As shown in Fig. 2, the composition $L^* \circ G$ is indeed sequential. However, besides generating useful states (0,0),(1,0),(3,0),(4,0),(0,1), and (0,2), the composition algorithm also generates nonuseful states (2,0), (5,0), and (6,0). It is normal for implementations of transducer composition to employ a post-processing step that removes such states. Nonuseful states are either nonaccessible or noncoaccessible. The generation of nonaccessible states can be avoided "on-the-fly" by generating the composition transducer state by state while performing a graph transversal starting at the initial state. The removal of noncoaccessible states is more problematic, because it requires either a post-processing step or the use of unbounded lookahead, neither of which is compatible with an efficient "on-the-fly" composition algorithm.

In order to avoid the generation of noncoaccessible states during composition, we propose a preprocessing step in which every ϵ output edge of the lexicon WFST is tagged with the set of non-ϵ output labels directly accessible from that edge [14]. The sets can be easily obtained by a post-order traversal of the lexicon WFST, in which the set assigned to an ϵ output edge is obtained as the union of the sets assigned to all edges leaving its destination state (non-ϵ output edges are assigned a singleton set). In our proposed composition algorithm, these sets are used

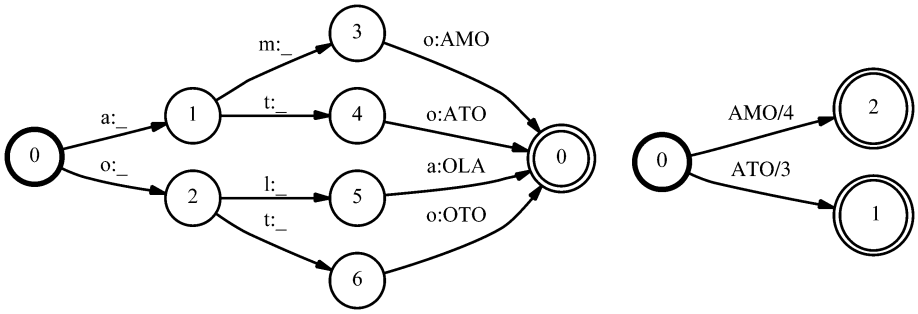


Fig. 1. Sample lexicon and language model WFSTs.

TABLE II
TOY LANGUAGE MODEL

Sentence	Cost $(-log)$
AMO	4
ATO	3

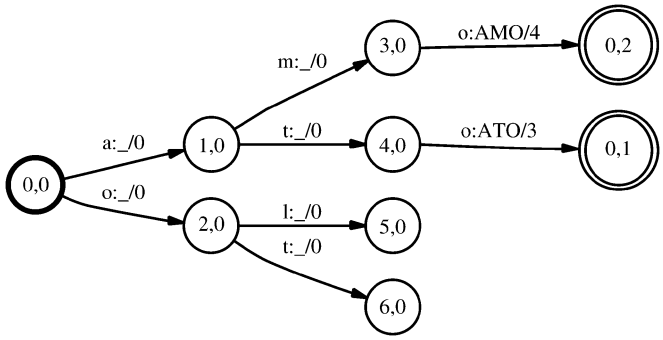


Fig. 2. Composition $L^* \circ G$.

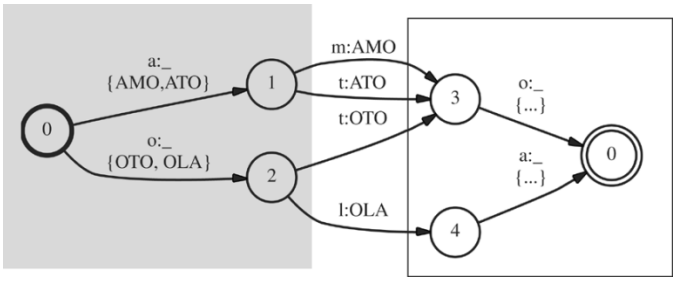


Fig. 3. Tagged lexicon loop $L^* = \min(L)^*$.

to avoid following ϵ output edges of the lexicon, when they do not lead to a useful path.

Fig. 3 shows a minimal deterministic tagged lexicon loop L^* representing our previous toy lexicon, after being tagged with the sets of non-ϵ output labels directly accessible from each edge. The set of all words is represented as $\{\ldots\}$. Fig. 4 shows its composition with the language model G.

C. Structure of the Search Space

In traditional large vocabulary continuous speech recognition systems, the search space is usually described in terms of lexicon copies, while in WFST-based approaches, an algebraic description is normally preferred. In this discussion of the specialized composition algorithm, we start by describing the search space graphically in terms of lexicon copies. To aid the discussion, we call the states in the form (i_l, q_g), where i_l is the initial state of

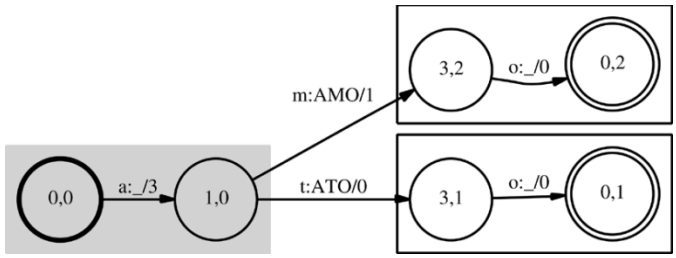


Fig. 4. Composition $L^* \circ G$.

INITIAL(q)
1 $(q_l, q_g) \leftarrow q$
2 return INITIAL(q_l) and INITIAL(q_g)

FINAL(q)
1 $(q_l, q_g) \leftarrow q$
2 return FINAL$(q_l) \otimes$ FINAL(q_g)

Fig. 5. Definition of INITIAL and FINAL functions for "on-the-fly" composition.

the lexicon and q_g is any language model state, as *anchor* states. We also call states in the lexicon that are between the initial state and a non-ϵ output edge, as *prefix* states (shown inside a gray rectangle in Fig. 3). Also, states in the lexicon, between a non-ϵ output edge and the final state, are called *suffix* states (shown inside a white rectangle). Assuming that the lexicon includes pronunciations for all words in the language model, anchor states (and indirectly, the language model) shape the global structure of the search space: every edge in the language model corresponds to paths between anchor states in the composition transducer.

Comparing the composition WFST with the lexicon, we see that two types of partial replication of the lexicon occur: one that starts in anchor states and replicates lexicon prefix states until non-ϵ output edges are found, and another which consists of suffix states and progresses until the end of the pronunciation. In the first type of replication, edges (and states) of the lexicon are filtered by the input labels (words) of edges leaving the language model state, so that only edges of the lexicon that match a language model label in its tagging set or in its output label are copied. In the second type, the filtering is done by the labels of incoming edges to the language model state which corresponds to the destination anchor state. Three replication regions are marked in Fig. 4, one prefix region, in gray, and two suffix ones in white rectangles. Due to this selective cloning of the lexicon WFST, much of its structure will be preserved in the composition WFST. Also, as long as pronunciation prefixes are shared in the lexicon WFST, the composition WFST is efficient for speech recognition, even when the lexicon loop WFST is not sequential.

```
ARCS(T, q = (q_l, q_g))
 1  A ← {}
 2  if q_l = INITIAL(L)
 3     then for each ε-input edge (q_g, ε, o_g, d_g, w_g) ∈ ARCS(G, q_g)
 4          do A ← A ⋃ (q, ε, o_g, (q_l, d_g), w_g)
 5  for each edge e_l = (q_l, l_l, o_l, d_l, w_l) ∈ ARCS(L, q_l)
 6  do if DESTSET(e_l) = Δ_l
 7        then A ← A ⋃ {(q, l_l, ε, (d_l, q_g), w_l)}
 8        else for each edge e_g = (q_g, l_g, o_g, d_g, w_g) ∈ ARCS(G, q_g)
 9             do
10                if l_g ∈ DESTSET(e_l)
11                  then if o_l = ε
12                          then A ← A ⋃ {(q, l_l, ε, (d_l, q_g), w_l)}
13                          else A ← A ⋃ {(q, l_l, o_g, (d_l, d_g), w_l ⊗ w_g)}
14  return A
```

Fig. 6. Noncoaccessible paths avoiding algorithm.

D. Algorithm

In this section, we present a first version of our specialized composition algorithm that uses the sets associated with the lexicon WFST to avoid the generation of noncoaccessible paths. In order to allow the incremental construction of the composition network, we followed the design of the AT&T FSM Library [3] for "on-the-fly" generation of WFSTs. In this design, the essential WFST interface consists of three functions:

- INITIAL (T) which returns the initial state of WFST T;
- FINAL (T, q) which returns the final cost $\rho(q)$ of state q if it is final or $\bar{0}$ otherwise;
- ARCS (T, q) which returns the set of edges leaving state q.

These functions are sufficient for a graph traversal algorithm to be able to explore the WFST without generating nonaccessible states. The INITIAL and FINAL functions (Fig. 5) are the same as used by the traditional composition algorithm. Our specialized algorithm differs only in the ARCS function (Fig. 6).

In the algorithm, lines 2 through 4 process ϵ-input edges in the language model. Two embedded loops, in lines 5 to 13, are used to match the set of edges leaving the lexicon state with the set of edges leaving the language model state. Edges in the suffix region of the lexicon (lines 6 and 7) are simply copied to the composition transducer. In line 10, the generation of noncoaccessible paths is avoided by comparing the input label of the language model edge with the set of labels reachable from the lexicon edge (accessed using the DESTSET function).

III. PUSHING

The first modification to the specialized composition algorithm previously described improves the distribution of language model weights through the paths. In fact, in the composition algorithm shown in Fig. 6, the language model weights are incorporated into the composed network only in non-ϵ output edges. This distribution of language model weights throughout paths is suboptimal for speech recognition, because the language model score is incorporated in a single step, and is delayed until the identity of a word is determined. It is usually much better to spread the language model score throughout the pronunciation path, even before that identity is completely known [15].

When the explicit weighted transducer determinization algorithm is used to optimize the composition of the lexicon with the language model as in [13], weights are naturally spread throughout pronunciation paths. Our composition algorithm can be improved to show a similar behavior. As a result, for each composition state, the improved algorithm keeps the accumulated language model weight which was spread from the previous anchor state. Then, in the ARCS function, the weight of the best matching language model edge is computed for each lexicon edge, so that when a composition edge is produced, the language model component of its weight is the difference between the accumulated weight at its origin state and the weight of the best matching language model edge.

On the other hand, when only one language model edge matches a lexicon edge, the improved algorithm can produce the output label immediately. This has the effect of pushing the output labels toward the initial state, enabling sharing of pronunciation suffixes as shown in Section IV. To avoid multiple output labels, a binary value is associated with each composition state indicating if it was already produced or not.

These two improvements of the algorithm do not change the topology of the network, only the distribution of weights and output labels throughout its paths.

Fig. 7 presents an improved version of the arcs function that provides pushing of weights and output labels.

The algorithm is similar to the previous one. For example, ϵ-input edges in the language model and edges in the suffix region of the lexicon are processed as before (in lines 2–4 and 6 and 7, respectively).

However, now, while the lexicon and language model edges are matched, the algorithm must also collect information necessary for the implementation of pushing. This information is kept in two global tables, $pushedCost$ and $pushed$, indexed with states d from the composition transducer. In the first array, $pushedCost[d]$ keeps the language model weight spread from the last anchor state, and in the second one, $pushed[d]$ keeps a binary flag indicating whether the output label was already produced or not. $pushed[d]$ is initialized as FALSE.

In addition, the algorithm collects in local variables, L and O, the sets of input and output labels of language model edges matching the current lexicon edge. The function PUSHPREFIXEDGES uses those auxiliary values to update newly created edges in the prefix region of the network, in order to implement pushing of weights and labels. When L is singleton, label pushing can be performed, and, accounting for a nondeterministic language model, an edge is produced for each output label found in matching language model edges (lines 7–11 in Fig. 8).

```
ARCS(T, q = (q_l, q_g))
 1  A ← {}
 2  if q_l = INITIAL(L)
 3    then for each ε-input edge (q_g, ε, o_g, d_g, w_g) ∈ ARCS(G, q_g)
 4           do A ← A ⋃ (q, ε, o_g, (q_l, d_g), w_g)
 5  for each edge e_l = (q_l, l_l, o_l, d_l, w_l) ∈ ARCS(L, q_l)
 6  do if DESTSET(e_l) = Δ_l
 7       then A ← A ⋃ {(q, l_l, ε, (d_l, q_g), w_l)}
 8       else if o_l = ε
 9              then A' ← O ← L ← {}
10                   p̂ ← 0̄
11                   for each edge (q_g, l_g, o_g, d_g, w_g) ∈ ARCS(G, q_g)
12                   do
13                     if l_g ∈ DESTSET(e_l)
14                       then d ← (d_l, q_g)
15                            A' ← A' ⋃ {(q, l_l, ε, d, w_l)}
16                            if pushed[q]
17                              then pushed[d] ← true
18                              else L ← L ⋃ {l_g}
19                                   O ← O ⋃ {o_g}
20                                   p̂ ← p̂ ⊕ w_g
21                   A = A ⋃ PUSHPREFIXEDGES(A', L, O, p̂)
22              else for each edge (q_g, l_g, o_g, d_g, w_g) ∈ ARCS(G, q_g)
23                   do
24                     if o_l = l_g
25                       then d ← (d_l, d_g)
26                            if pushed[q]
27                              then A ← A ⋃ {(q, l_l, ε, d, w_l)}
28                              else ŵ ← w_l ⊗ w_g ⊗ pushedCost[q]^{-1}
29                                   A ← A ⋃ {(q, l_l, o_g, d, ŵ)}
30                            if d_l ≠ INITIAL(L)
31                              then pushed[d] ← true
32  return A
```

Fig. 7. Algorithm for creating edges with weight and output label pushing.

```
PUSHPREFIXEDGES(A', L, O, p̂)
 1  A ← {}
 2  for each edge (q, l, o, d, w) ∈ A'
 3  do if p̂ ≠ 0̄
 4       then ŵ ← w ⊗ p̂ ⊗ pushedCost[q]^{-1}
 5            pushedCost[d] ← p̂
 6       else ŵ = w
 7     if L = {l̂}
 8       then pushed[d] ← true
 9            for each label ô ∈ O
10            do A ← A ⋃ {(q, l, ô, d, ŵ)}
11       else A ← A ⋃ {(q, l, o, d, ŵ)}
12  return A
```

Fig. 8. Function for fixing prefix edges.

A. Discussion

Our algorithm imposes two restrictions on the structure of the lexicon WFST. The first is that the lexicon loop (L^*) must loop through the initial state, which is used by the algorithm as a marker for the boundary between the pronunciations of words. This marker is used to account for ϵ-input labels in the language model WFST (lines 2–4 in Figs. 6 and 7) and to ensure that the language model weight of any particular word is spread only through its pronunciation path and does not spill to adjacent words. The second restriction is that every path in the lexicon between the initial and final state must output one and only one word. This restriction ensures that the computation of the value $pushedCost[d]$, associated with a particular state d, is the same regardless of the path followed by the "on-the-fly" algorithm to reach the state. This is required by the "on-the-fly" nature of the algorithm. It is also the reason why the lexicon weight is not included in the pushed value, as the existence of lexicon edges with different weights but with the same destination state could result in different values for $pushedCost[d]$. The presented pushing algorithm resembles traditional language model lookahead in this aspect. Also, pushing is not performed across word boundaries.

Unlike other algorithms for computing language model lookahead, such as [16] or [15], we do not take advantage of the lexicon structure to reuse computed lookahead weights. These algorithms start the computation at the leaves of the lexicon tree and progress toward the root by computing the best language model score for each internal node as the best score of its child nodes.

Our algorithm instead takes the approach of computing the lookahead for each edge independently of others. Our approach is globally less efficient; however, it is better suited for an "on-the-fly" algorithm because it has the advantage of not requiring the computation of complete lexicon copies, allowing the creation of only the states required by the search algorithm.

IV. Minimization

Besides pushing, the other main global optimization operation is *minimization*; its main effect is to reduce the size of the network, an important concern when a static network is desired. Minimization is difficult to approximate in an "on-the-fly" algorithm because of its global nature. The difficulty comes from the fact that classic minimization algorithms build equivalence classes of states with the same right language, while typical

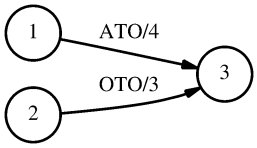


Fig. 9. Sample language model fragment G_2.

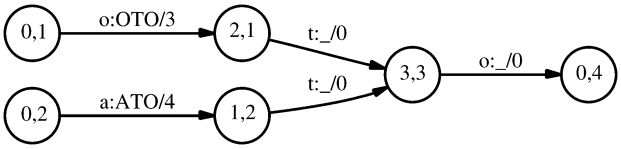


Fig. 10. Composition fragment $L^* \circ G_2$.

"on-the-fly" algorithms know nothing from a particular state onward. They are only required to know at least one path up to a particular state. Nevertheless, the specific structure of the composition transducer allows us to exploit and remove various sources of redundancy.

A. Minimization of the Language Model

In our efforts to approximate minimization, we begin by minimizing the language model in a preprocessing step. The rational is that if the language model is not a minimal deterministic transducer, then large portions of the composition transducer will be redundant. An additional motivation for minimizing the language model is that the composition's anchor states will tend to belong to different equivalence classes when classical minimization is applied to the composition transducer. However, this is not absolutely true: for example in the extreme cases where all words in the lexicon are homophone, or all edges of the language model WFST have the output label ϵ.

B. Suffix Sharing

One other source of redundancy in the composition network comes from the fact that pronunciation suffixes of words arriving at the anchor state are not completely shared. When the lexicon is minimal, the previous algorithm implicitly shares the suffixes present in the lexicon. However, after output-label pushing, word suffixes in the composition WFST are extended, allowing additional sharing.

Fig. 9 illustrates a sample language model fragment G_2, and when it is composed with the minimal lexicon L^*, the network shown in Fig. 10 is generated. As shown, the suffix sharing may not be complete. In order to closely approximate minimization, we would like to merge state (2,1) with state (1,2).

The redistribution of weights and output labels resulting from pushing enables further sharing of suffixes in the composition network. In order to achieve better suffix sharing than possible with a deterministic and minimal lexicon WFST (L_{det}), we will also employ an equivalent "suffix-sharing" right-sequential lexicon WFST (L_{suf}), which has the topology illustrated in Fig. 11. The topology is such that $reverse(L_{\text{suf}})$ is sequential, and all non-ϵ output labels leave the initial state. Our idea is to use L_{suf} as a guide to generate the suffixes, so that the lexicon component q_l of a composition state (q_l, q_g) can now refer

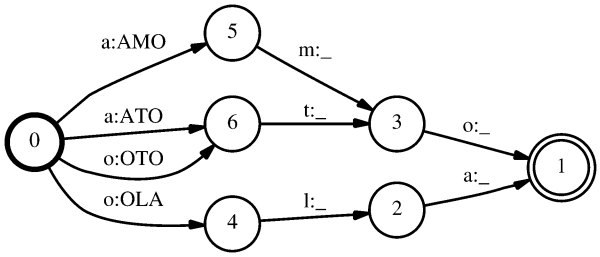


Fig. 11. Suffix sharing lexicon WFST L_{suf}.

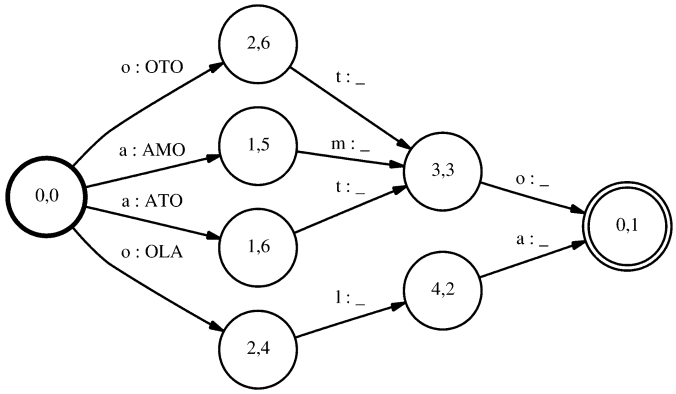


Fig. 12. $L_{\text{comp}} = \pi_1(L_{\text{det}}) \circ L_{\text{suf}}$.

either to a state from L_{det} or L_{suf}. Since no overlap is allowed between the set of states of the different lexica, the function of array $pushed[q]$ can be replaced by testing if $q_l \in Q_{\text{det}}$.

The main difficulty of using two lexica is establishing correspondences between their states. This situation occurs after the production of an edge with non-ϵ output label o. The destination state q_{det} of the edge must be replaced by states q_{suf} in L_{suf}. The first step to establish these correspondences is to compose the lexica, $L_{\text{comp}} = \pi_1(L_{\text{det}}) \circ L_{\text{suf}}$, illustrated in Fig. 12. And by construction it is equivalent to both L_{suf} and L_{det}. Using L_{comp}, the correspondence problem can be solved by selecting all suffix states q_{suf} such that the state $(q_{\text{det}}, q_{\text{suf}})$ exists in a path which outputs o in L_{comp}. The placement of the non-ϵ output labels in L_{suf} after the initial state simplifies the search for such paths.

The search for the set of suffix states equivalent to a state in the deterministic lexicon is performed in function FINDINSUFFIXLEX (Fig. 13). It is implemented as a simple graph traversal algorithm over the composition lexicon transducer. To accelerate the search, the results of this function can be memorized for reuse.

C. Tail Sharing

The algorithm presented in the previous section addresses the problem of minimizing the composition network. Yet, its ability to share suffixes common to different words is not used when the language model is an n-gram, because edges entering an n-gram language model are either backoff edges labeled with ϵ or all have the same label. When n-gram language models are used, sharing suffixes of the same word will achieve almost the same level of sharing. The main difference is that suffix sharing may perform better on multiple pronunciations of the same word.

```
FINDINSUFFIXLEX (q, l)
 1  S ← {}
 2  R ← {}
 3  for each edge (i_comp, l_l, o_l, d, w_l) ∈ ARCS(i_comp)
 4  do if o_l = l
 5        then ENQUEUE(S, d)
 6  while S ≠ {}
 7  do (q_det, q_suf) ← HEAD(S)
 8     DEQUEUE (S)
 9     if q_det = q
10        then if q_suf ∈ F_suf
11                then R ← R ⋃ {i_det}
12                else R ← R ⋃ {q_suf}
13        else for each edge ((q_det, q_suf), l_l, o_l, d, w_l) ∈ ARCS((q_det, q_suf))
14             do if d ∉ S
15                   then ENQUEUE(S, d)
16  return R
```

Fig. 13. Function FINDINSUFFIXLEX.

In this section, we present a technique for sharing suffixes which does not require a suffix lexicon and is particularly effective when n-gram language models are used.

The method proposed here resembles Brugnara and Cettolo's *tail sharing* technique [16] because it allows the sharing of pronunciation suffixes by instances of the same word. However, it is more general since it shares pronunciation suffixes regardless of their topology; in particular, it is not limited to sharing linear sequences of phones.

Instead of using a different lexicon WFST to guess the structure of suffixes, our tail sharing algorithm simply reuses paths taken by other instances of suffixes of the same word going into the same anchor state.

The modification of the pushing algorithm which implements tail-sharing consists of using a hash table H that contains states from the composition transducer. These states are indexed by a tuple $(q_l, o_l, (i_l, d_g))$, where q_l is a lexicon state, o_l is the word whose suffix is being shared, and (i_l, d_g) is the next anchor state. Whenever a state (q_l, q_g) is generated such that q_l is a prefix state but the non-ϵ output label o_l was already produced, a lookup is made in the hash table $H[q_l, o_l, (i_l, d_g)]$ to find a state along the same position in the lexicon (in other words, on a path of the same word and going to the same anchor state). If such a state is found, it is used instead of (q_l, q_g), otherwise (q_l, q_g) is entered in the hash table. These modifications are described in deeper detail in [17].

V. Optimizing the Algorithm

Our algorithms presented so far, are inefficient because the ARCS function running time is proportional to the product of the number of edges leaving the lexicon and the language model states. This quadratic complexity is the composition algorithm's worst case, when each lexicon edge matches every language model edge. Nevertheless, in practice, a lexicon edge matches with very few, if any, of the language model edges. In the previous sections, this sparseness was not explored.

Algorithms designed for use in large vocabulary speech recognition, such as the ones implemented in [3], explore this sparseness, due to the large size of the transducers used.

In this section, we will show how the presented algorithms can be improved to exhibit a running time proportional to the number of edges produced in the composition WFST.

The inefficiency of the previous algorithms lies in the embedded loops which match lexicon edges to language model edges. In an implementation of the classical composition algorithm, we can easily improve this matching if the edges leaving the language model and the lexicon state are presorted. Then it would be possible to use seeking functions (e.g., SEEKINPUT or SEEKOUPUT) to quickly skip over nonmatching edges, and the composition algorithm would resemble the MERGE algorithm for combining sorted arrays.

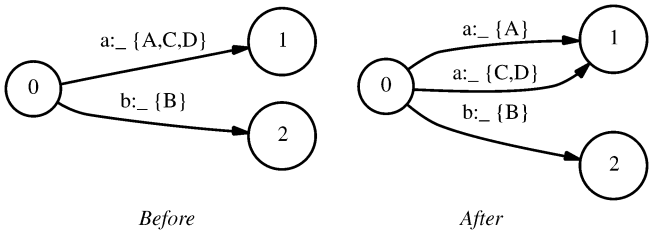


Fig. 14. Lexicon preprocessing: Split sets.

The problem we face when trying to port this idea to our algorithm is that, in our case, the lexicon edges should be sorted by the set returned by DESTSET. Yet, a complete order for those *tagging sets* cannot easily be found. Assuming a complete order over words $<_w$ (for example, the lexicographic order), two sets of words W_1 and W_2 can be ordered if either $\forall_{w_1 \in W_1, w_2 \in W_2}\ w_1 <_w w_2$ or $\forall_{w_1 \in W_1, w_2 \in W_2}\ w_2 <_w w_1$. If two sets cannot be thus ordered, we say that the range of words that they contain *overlaps*.

If the prefix region of the lexicon WFST is organized as a tree, a complete order can easily be found; however, we do not want to further restrict the structure of the lexicon. The solution we found consists of an automatic preprocessing step, which splits sets and replicates edges of the lexicon WFST, so that sets can be ordered. This process is exemplified in Fig. 14, where A, B, C, and D are arbitrary words, and the usual lexicographic order of words is assumed. In the example, we observe that the set $\{A, C, D\}$ is overlapping with set $\{B\}$. After preprocessing, the sets $\{A\}$, $\{C, D\}$, and $\{B\}$ do not overlap and can be sorted. This preprocessing step consists of splitting sets to avoid overlaps and replicating the edges whose sets were split. After this step, lexicon edges are sorted by their tagging set. In the worse case, each set will be a singleton, and the total number of edges in the preprocessed lexicon will be the same as the equivalent linear lexicon WFST, e.g., one with no shared pronunciation paths.

The pseudocode of the optimized algorithm and a more detailed explanation can be found in [17].

Due to the large number of sets tagging edges of the lexicon the use of a memory efficient representation for sets is important. One such representation consists of sorting words by the lexicographic order of their pronunciations and representing each set as a list of ranges of consecutive words.

VI. State-Level Optimization and Context-Dependency

The topic of this paper is lexicon and language model composition. However, our decoder requires a search space specified at the hidden Markov model (HMM) state level. Since our recognition system is based on context-independent units, we build this network as $H^* \circ (L^* \bullet G)$, where $\bullet$ is the specialized composition algorithm presented, and H^* is a transducer mapping from states (or distributions) to subword units and enforces the topology of HMMs.

Using this approach, the search network is optimized at the subword level. However, when tied-state context-dependent acoustic units are used, a significant boost in performance can be obtained by optimizing the network at the state level [6], [18]. The presented algorithm is able to optimize the network at this finer level. In particular, when word-internal context-dependent units are used, $\det(H^* \circ L^*)$ satisfies the restrictions imposed on the structure of the lexicon by the algorithm, and the search space can be built as $\det(H^* \circ L^*) \bullet G$. When cross-word context-dependent units are used, the network is built as $\det(H^*) \bullet (C' \circ (L^* \bullet G))$, where $\det(H^*)$ is a loop containing all physical (distinct) HMM models, and thus satisfies the specialized algorithm requirements. C' imposes the bigram constraints of triphone sequences, it is built as $C' = (\min(C \circ T))^{-1}$, where C maps context-independent units to logical (possible) context-dependent units, and T maps logical to physical units. We recall that, because of clustering, many logical units may be implemented by the same physical one. The inverse of C, C^{-1} is a deterministic transducer constructed as shown in [13]. We have not yet explored these ideas because our current recognition system is based on context-independent units.

VII. Experimental Results

The recognition experiments described in this section were based on the European Portuguese broadcast news corpus collected in the scope of the ALERT European project [19]. A 57 k lexicon was used in all experiments. The lexicon was disambiguated by concatenating special disambiguation symbols to pronunciations as described in [13]. Several language models were used, varying in size from 3-gram models to large 4-gram language models. Efficiency was measured as runtime ratios (xRT) in a 1-GB RAM Pentium III computer running at 1 GHz.

The acoustic models used by our hybrid recognition system are based on three different feature extraction methods and on three multilayer perceptrons with the same structure, an input layer with nine context frames, a nonlinear hidden layer with over 1000 sigmoidal units, and 40 softmax outputs. The feature extraction methods are perceptual linear prediction (PLP), relative spectral transform (Log-RASTA), and modulation spectrogram (MSG). The output of the three multilayer perceptrons is averaged in the log probability domain [20].

TABLE III
Dimension of Search Networks

Network	States	Edges
min push det$(L \circ G)$	4,253,768	8,273,644
share suffixes log prob	4,346,025	8,413,535

A. Comparison With the Explicit Determinization Approach

In order to evaluate the effectiveness of the approximations introduced, we decided to compare them with the networks generated using explicit weighted determinization, minimization, and pushing algorithms. A 3-gram language model with three million n-grams was used.

In Table III, we show the size of the composition network obtained with weighted determinization, minimization and pushing, using the *log probability* semiring. The table also shows the size of the network obtained using our suffix-sharing weight-pushing algorithm. The deterministic lexicon WFST and the language model are minimized. The composition algorithm uses the *log probability* semiring.

We can see that our approximation has 2% more states or edges than the explicit determinization approach.

Fig. 15 shows that the performance of both algorithms is comparable; however, our approach has a better behavior at lower beams at the expense of higher ones. Overall, it is a good approximation to explicit determinization and offline optimization of the search network.

B. Evaluating the Relative Effect of the Semiring Choice

According to [9], the choice of the semiring used for pushing has a strong impact on the performance of the explicit determinization approach. To ascertain this impact in our algorithm, experiments were performed. Because we can independently control the semiring choice at the local level when spreading the weights through pronunciation paths and at the language model level by prepushing the language model, six different recognition experiments were conducted using two scenarios. 1) The language model is not pushed, pushed in the tropical semiring, or pushed in the log probability semiring. 2) The composition algorithm uses either log probability or tropical pushing. Suffix-sharing is used when building the static composition networks used in these experiments.

Fig. 16 shows experiments performed with the 3-gram language model used in the previous section. The behavior of the algorithms is affected by language model pushing, especially at higher beams, but without a clear pattern. At higher beams, the small difference between the methods is of little practical use. The most marked result is that composing with the log probability semiring provides the best behavior at low beams. This result is consistent with the ones reported in [9]. In the following experiments, composition was done using the log probability semiring while the language model was not pushed.

C. Overhead of "On-the-Fly" Generation of the Search Space

Experiments were done to assess the impact of generating the search space "on-the-fly" versus generating it using a precompiled static search network with the same 3-gram language model. Fig. 17 shows the performance of the pushing algorithm

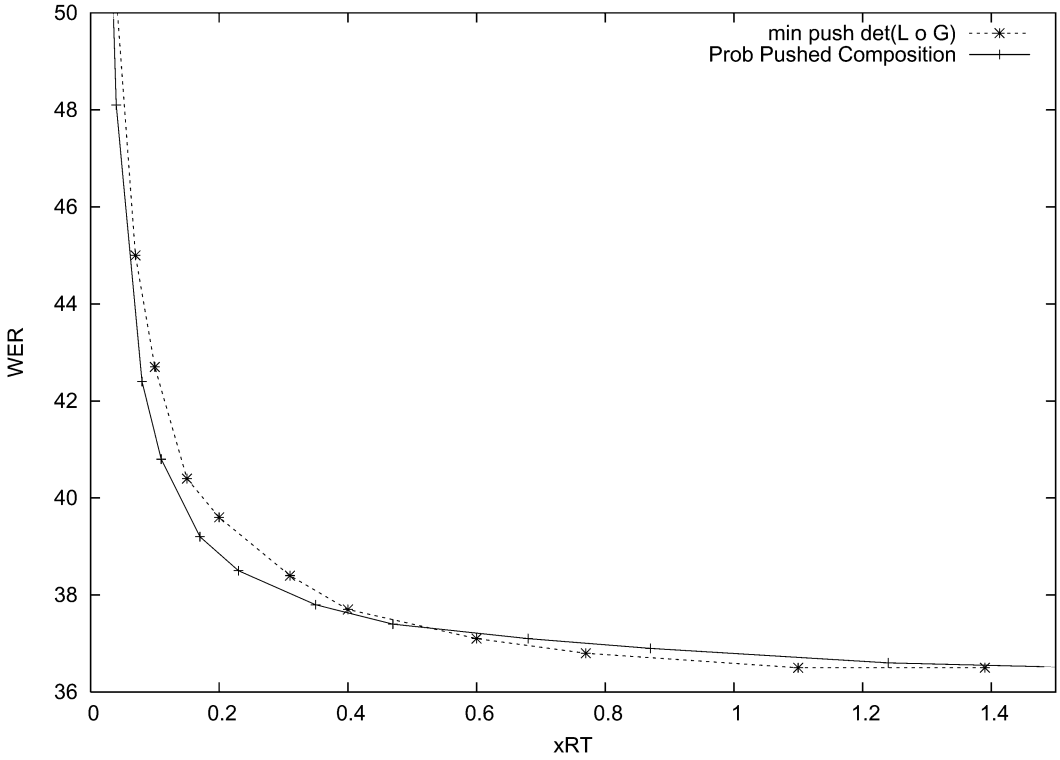


Fig. 15. Explicit determinization versus suffix sharing algorithm.

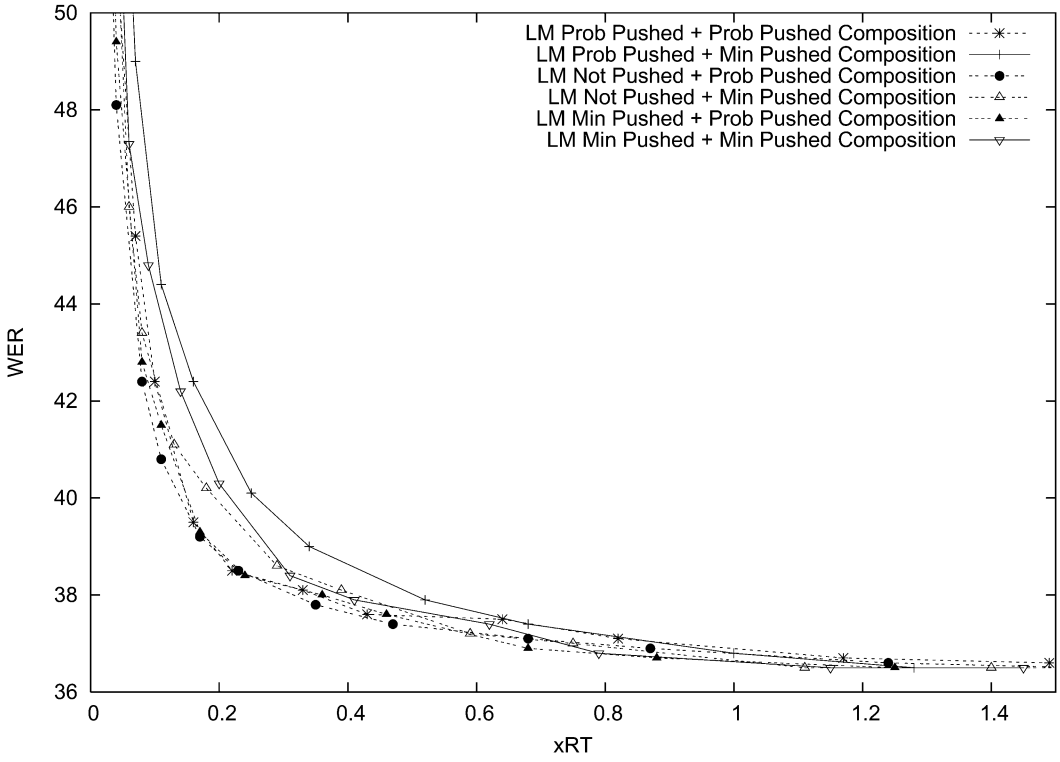


Fig. 16. Impact of pushing approximations in a medium size language model.

with suffix or tail-sharing, and using a static network or creating the network "on-the-fly." As expected, we observe an overhead when the network is created "on-the-fly." The overhead relative to the static versions is high at low beams, but drops below 15% at higher beams. We also observe that in this task, the tail-sharing algorithm is a good approximation to suffix sharing.

D. Comparison With Other "On-the-Fly" Approaches

A large 4-gram language model with 27 million n-grams was used to compare our algorithm with the incremental language model "on-the-fly" approach [10]. In this approach, a small language model G_s is used to build a static optimized network such

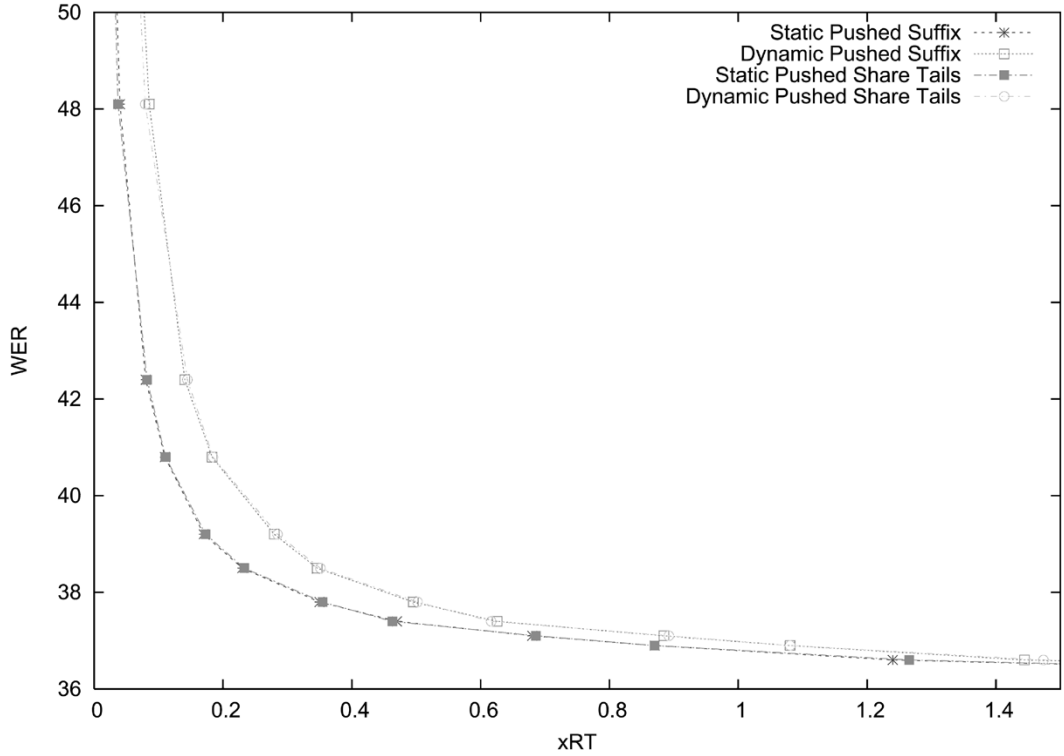


Fig. 17. "On-the-fly" versus static network.

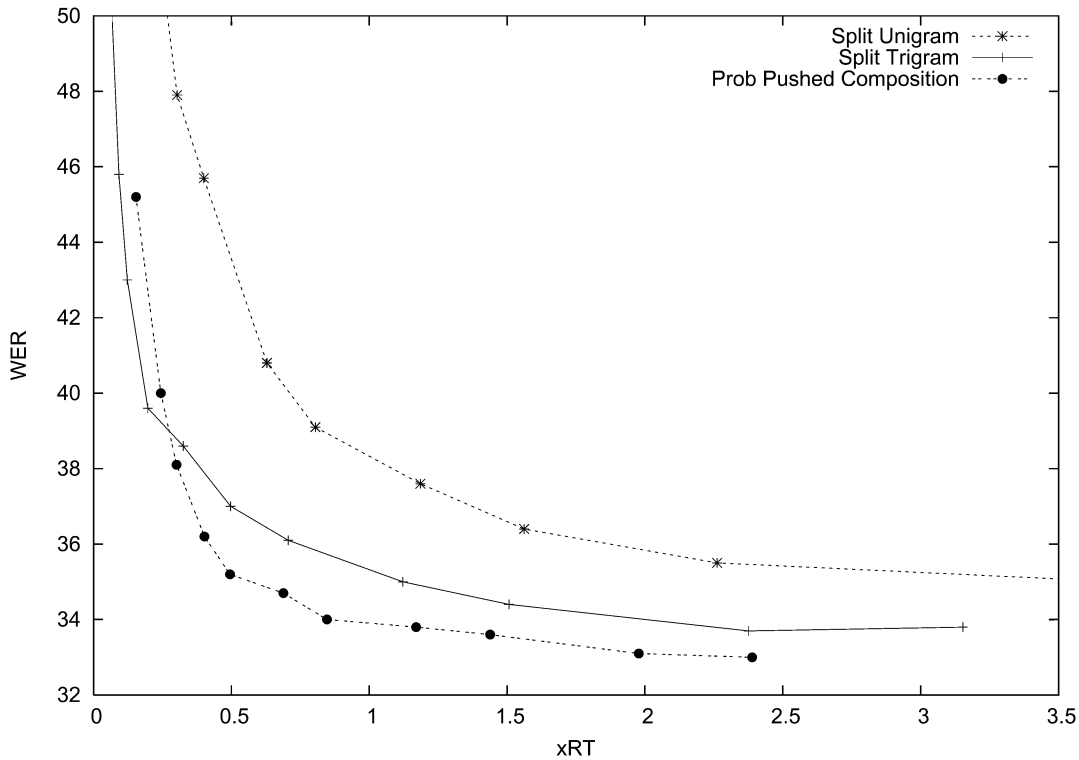


Fig. 18. Comparison with other "on-the-fly" approaches.

as $N = \min \text{ push } \det(H \circ L \circ G_s)$. The information from the full language model G_f is included by using a specialized composition operation $\bullet$ to compose $N \bullet G_d$ "on-the-fly" in runtime. The transducer G_d is such that $G_f = G_u \circ G_d$; see [10] for details. We compared our algorithm with two instances of this approach, using either a unigram with 57 000 n-grams, or a trigram language model with 1.8 million n-grams, as small language models. The results are presented in Fig. 18. The better behavior of the presented algorithm is due to better modeling of language model lookahead. Its memory requirements are similar to the requirements of the incremental approach using a unigram small language model.

VIII. Conclusion

This paper presented various specialized optimization algorithms for composing the lexicon with the language model. The algorithms provide exact simultaneous composition and determinization of the lexicon and language model WFSTs. Furthermore, versions which approximate minimization and pushing of weights and output labels were also presented. The algorithms were designed to permit an efficient "on-the-fly" generation of the composition network for use in a dynamic speech recognizer.

The specialized algorithms impose some limitations on the structure of the lexicon. The most restrictive one is forces the lexicon to loop through the initial state. This restriction can be easily overcome if all pronunciatons terminate with a special "end-of-word" symbol to be identified by the composition algorithm.

Although not described in this paper, it is interesting to compare the performance of the WFST approach with our previous non-WFST-based decoder [21]. The specialized composition algorithm using weight pushing achieves a 6× reduction of the real-time factor.

References

[1] F. Pereira and M. Riley, "Speech recognition by composition of weighted finite automata," in *Finite-State Language Processing*, E. Roche and Y. Schabes, Eds. Cambridge, MA: MIT Press, 1997, pp. 431–453.

[2] M. Mohri, F. Pereira, and M. Riley, "Weighted automata in text and speech processing," in *Proc. ECAI Workshop*, Aug. 1996, pp. 46–50.

[3] ——, "A rational design for a weighted finite-state transducer library," in *Proc. Automata Implementation. 2nd Int. Workshop on Implementing Automata, WIA '97.*, vol. 1436, Lecture Notes in Computer Science, 1998, pp. 144–158.

[4] C. Allauzen and M. Mohri, "Generalized optimization algorithm for speech recognition transducers," in *Proc. ICASSP*, vol. 1, Hong Kong, China, Apr. 2003, pp. 352–355.

[5] M. Mohri, F. Pereira, and M. Riley, "Transducer composition for context-dependent netword expansion," in *Proc. Eurospeech*, Rhodes, Greece, Sep. 1997, pp. 1427–1430.

[6] M. Mohri and M. Riley, "Integrated context-dependent networks in very large vocabulary speech recognition," in *Proc. Eurospeech*, vol. 2, Budapest, Hungary, Sep. 1999, pp. 811–814.

[7] M. Mohri, "Finite-state transducers in language and speech processing," *Comput. Linguistics*, vol. 23, no. 2, pp. 269–311, Jun. 1997.

[8] ——, "Minimization algorithms for sequential transducers," *Theor. Comput. Sci.*, vol. 234, pp. 177–201, 2000.

[9] M. Mohri and M. Riley, "A weight pushing algorithm for large vocabulary speech recognition," in *Proc. Eurospeech*, Aalborg, Denmark, Sep. 2001, pp. 1603–1606.

[10] H. Dolfing and I. Hetherington, "Incremental language models for speech recognition using finite-state transducers," in *Proc. ASRU Workshop*, Trento, Italy, Dec. 2001, pp. 194–197.

[11] D. Caseiro and I. Trancoso, "A decoder for finite-state structured search spaces," in *Proc. ASR Workshop*, Paris, France, Sep. 2000, pp. 35–39.

[12] D. Willett and S. Katagiri, "Recent advances in efficient decoding combining on-line transducer composition and smoothed language model combination," in *Proc. ICASSP*, vol. 1, Orlando, FL, May 2002, pp. 713–716.

[13] M. Mohri, M. Riley, D. Hindle, A. Ljolje, and F. Pereira, "Full expansion of context-dependent networks in large vocabulary speech recognition," in *Proc. ICASSP*, vol. 2, Seattle, WA, May 1998, pp. 665–668.

[14] D. Caseiro and I. Trancoso, "On integrating the lexicon with the language model," in *Proc. Eurospeech*, Aalborg, Denmark, Sep. 2001, pp. 2131–2134.

[15] S. Ortmanns, H. Ney, and A. Eiden, "Language-model look-ahead for large vocabulary speech recognition," in *Proc. ICSLP*, vol. 4, Philadelphia, PA, Oct. 1996, pp. 2095–2098.

[16] F. Brugnara and M. Cettolo, "Improvements in tree-based language model representation," in *Proc. Eurospeech*, Madrid, Spain, Sep. 1995, pp. 2133–2136.

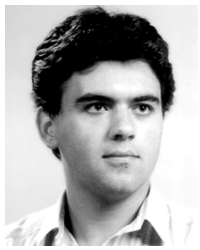
[17] D. Caseiro, "Finite-State Methods in Automatic Speech Recognition," Ph.D. dissertation, Instituto Superior Técnico, Lisbon, Portugal, 2003.

[18] H. Dolfing, "A comparison of prefix tree and finite-state transducer search space modelings for large-vocabulary speech recognition," in *Proc. ICSLP*, Denver, CO, Sep. 2002, pp. 1305–1308.

[19] H. Meinedo, N. Souto, and J. Neto, "Speech recognition of broadcast news for the european portuguese language," in *Proc. ASRU Workshop*, Trento, Italy, Dec. 2001, pp. 319–322.

[20] H. Meinedo and J. Neto, "Combination of acoustic models in continuous speech recognition hybrid systems," in *Proc. ICSLP*, vol. 2, Beijing, China, Oct. 2000, pp. 931–934.

[21] ——, "Automatic speech annotation and transcription in a broadcast news task," in *Proc. MSDR*, Hong Kong, Apr. 2003, pp. 95–100.



Diamantino Caseiro (M'00) graduated in informatics and computer engineering in 1994 from Instituto Superior Técnico (IST), Lisbon, Portugal, and received the M.Sc. degree in electrical and computer engineering and the Ph.D. degree in computer science, also from IST, in 1998 and 2003, respectively.

He has been a Lecturer at IST since 2000, first as a Teaching Assistant, then as an Assistant Professor since 2004 (on compilers, and analysis and synthesis of algorithms). He has been a Researcher at INESC, Lisbon, since 1996 with the Speech Processing Group, which became the Spoken Language Systems Laboratory (L2F) in 2001. His first research topic was automatic language identification. His Ph.D. topic was finite-state methods in automatic speech recognition. He has participated in several European and national projects, and currently leads one national project on weighted finite-state transducers applied to spoken-language processing.

Dr. Caseiro is a member of the International Speech Communication Association (ISCA), the ACM, and the IEEE Computer Society.

Isabel Trancoso (SM'03) received the Licenciado, Mestre, Doutor, and Agregado degrees in electrical and computer engineering from Instituto Superior Ténico (IST), Lisbon, Portugal, in 1979, 1984, 1987, and 2002, respectively.

She has been a Lecturer at IST since 1979, having coordinated the EEC course for six years. She is currently a Full Professor, teaching speech processing courses. She is also a Senior Researcher at INESC ID Lisbon, having launched the Speech Processing Group, now restructured as Spoken Language Systems Laboratory, in 1990. Her first research topic was medium-to-low bit-rate speech coding. From October 1984 through June 1985, she worked on this topic at AT&T Bell Laboratories, Murray Hill, NJ. Her current scope is much broader, encompassing many areas in speech recognition and synthesis, with a special emphasis on tools and resources for the Portuguese language.

Dr. Trancoso was a member of the International Speech Communication Association (ISCA) Board (1993–1998), the IEEE Speech Technical Committee (since 1999), and the Permanent Council for the Organization of the International Conferences on Spoken Language Processing (since 1998). She was Editor-in-Chief of the IEEE Transactions on Speech and Audio Processing (2003–2005), Member-at-Large of the IEEE Signal Processing Society Board of Governors (2006–2008), and Vice-President of ISCA (2005–2009). She chaired the Organizing Committee of the Eurospeech'2005 Conference (INTERSPEECH'2005) that took place in September 2005 in Lisbon.