A Generalized Composition Algorithm for Weighted Finite-State Transducers

Cyril Allauzen, Michael Riley, Johan Schalkwyk

Google, Inc., 76 Ninth AV, New York, NY 10003

`{allauzen, riley, johans}@google.com`

Abstract

This paper describes a weighted finite-state transducer composition algorithm that generalizes the concept of the *composition filter* and presents filters that remove useless epsilon paths and push forward labels and weights along epsilon paths. This filtering permits the compostion of large speech recognition context-dependent lexicons and language models much more efficiently in time and space than previously possible. We present experiments on Broadcast News and a spoken query task that demonstrate an $\sim$5% to 10% overhead for dynamic, runtime composition compared to a static, offline composition of the recognition transducer. To our knowledge, this is the first such system with so little overhead.

Index Terms: WFST, LVCSR

1. Introduction

Weighted finite-state transducers (WFST)s have been shown to be a general and efficient representation in speech recognition [1]. They have been used to represent a language model G (an automaton over words), the phonetic lexicon L (a CI-phone-to-word transducer), and the context-dependency specification C (a CD-phone to CI-phone transducer). Further, these have been combined and optimized with finite-state operations of *composition* and *determinization* as:

$$C \circ \det(L \circ G) \qquad (1)$$

to produce a recognition transducer that is very efficient to search in ASR decoding. An attractive alternative construction that produces an equivalent transducer is:

$$(C \circ \det(L)) \circ G. \qquad (2)$$

If G is deterministic, Eq. 2 could result in a transducer as efficient as that in Eq. 1 for decoding and have the advantage that the determinization is restricted to the lexicon, greatly saving time and memory in the construction. When the recognition transducer is built offline, this would be a convenience. When (pre-built) $C \circ \det(L)$ is combined dynamically with G during recognition (useful in various applications), this would be a great benefit for efficient decoding.

Unfortunately, the standard WFST composition algorithm presents three significant problems with this alternate approach. L is typically constructed with the word output labels leaving its initial state; this makes for immediate matching with words in G during composition. However, the determinization of L moves back the word labels so that phonetic prefixes are shared, these prefixes now having ϵ output labels. Problem 1: This introduces delays in matching and will create useless paths, possibly very many, in standard composition with any G that is not complete (such as the compact WFST representation of a backoff n-gram model). Problem 2: Since the word labels will match later, the resulting transducer will be different even if the useless paths are trimmed. In fact, when G is an n-gram model, the result will typically be much larger. Problem 3: The weight distribution of the resulting transducer will also be different; grammar weights will appear on paths later, often to the detriment of ASR pruning.

This paper describes a generalization to the standard composition algorithm that solves each of these three problems. Section 2 presents the generalized composition algorithm, Section 3 describes large-vocabulary speech recognition experiments using various static and dynamic transducer constructions. Section 4 compares our approach to related work by others [2, 3, 4].

2. Composition Algorithm

A detailed description of weighted finite-state transducers - their theory, algorithms and applications to speech recognition - is given in [1]. The presentation here is limited those aspects needed for the generalized composition algorithm.

2.1. Preliminaries

A semiring $(\mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1})$ is ring that may lack negation. If $\otimes$ is commutative, we say that the semiring is *commutative*.

The *probability semiring* $(\mathbb{R}_+, +, \times, 0, 1)$ is used when the weights represent probabilities. The *log semiring* $(\mathbb{R} \cup \{-\infty, +\infty\}, \oplus_{\log}, +, \infty, 0)$, isomorphic to the probability semiring via the negative-log mapping, is often used in practice for numerical stability. The *tropical semiring* $(\mathbb{R} \cup, \{-\infty, +\infty\}, \min, +, \infty, 0)$, derived from the log semiring using the *Viterbi approximation*, is often used in shortest-path applications.

A *weighted finite-state transducer* $T = (\mathcal{A}, \mathcal{B}, Q, I, F, E, \lambda, \rho)$ over a semiring $\mathbb{K}$ is specified by a finite input alphabet $\mathcal{A}$, a finite output alphabet $\mathcal{B}$, a finite set of states Q, a set of initial states $I \subseteq Q$, a set of final states $F \subseteq Q$, a finite set of transitions $E \subseteq \overline{E} = Q \times (\mathcal{A} \cup \{\epsilon\}) \times (\mathcal{B} \cup \{\epsilon\}) \times \mathbb{K} \times Q$, an initial state weight assignment $\lambda : I \to \mathbb{K}$, and a final state weight assignment $\rho : F \to \mathbb{K}$. $E[q]$ denotes the set of transitions leaving state $q \in Q$.

Given a transition $e \in E$, $p[e]$ denotes its origin or previous state, $n[e]$ its destination or next state, $i[e]$ its input label, $o[e]$ its output label, and $w[e]$ its weight. A *path* $\pi = e_1 \cdots e_k$ is a sequence of consecutive transitions: $n[e_{i-1}] = p[e_i]$, $i = 2, \ldots, k$. The functions n, p, and w on transitions can be extended to paths by setting: $n[\pi] = n[e_k]$ and $p[\pi] = p[e_1]$ and by defining the weight of a path as the $\otimes$-product of the weights of its constituent transitions: $w[\pi] = w[e_1] \otimes \cdots \otimes w[e_k]$. A *string* is a sequence of labels; ϵ denotes the empty string.

The weight associated by T to any pair of input-output strings (x, y) is given by:

$$T(x,y) = \bigoplus_{\pi \in \cup_{q \in I,\, q' \in F} P(q,x,y,q')} \lambda[p[\pi]] \otimes w[\pi] \otimes \rho[n[\pi]], \qquad (3)$$

where $P(q, x, y, q')$ denotes the set of paths from q to q' with input label $x \in \mathcal{A}^*$ and output label $y \in \mathcal{B}^*$.

We denote by $|T|_Q$ denotes the number of states, $|T|_E$ the number of transitions, and $d(T)$ the maximum out-degree in T. The *size* of T is then $|T| = |T|_Q + |T_E|$.

2.2. Composition

Let $\mathbb{K}$ be a commutative semiring and let T_1 and T_2 be two weighted transducers defined over $\mathbb{K}$ such that the input alphabet $\mathcal{B}$ of T_2 coincides with the output alphabet of T_1. The result of the composition of T_1 and T_2 is a weighted transducer denoted by $T_1 \circ T_2$ and specified for all x, y by:

$$(T_1 \circ T_2)(x,y) = \bigoplus_{z \in \mathcal{B}^*} T_1(x,z) \otimes T_2(z,y). \tag{4}$$

Leaving aside transitions with ϵ inputs or outputs, the following rule specifies how to compute a transition of $T_1 \circ T_2$ from appropriate transitions of T_1 and T_2: (q_1, a, b, w_1, q_1') and (q_2, b, c, w_2, q_2') results in $((q_1, q_2), a, c, w_1 \otimes w_2, (q_1', q_2'))$. A simple algorithm to compute the composition of two ϵ-free transducers, following the above rule, is given in [1].

More care is needed when T_1 has output ϵ labels or T_2 input ϵ labels. An output ϵ label in T_1 may be matched with an input ϵ label in T_2, following the above rule with ϵ labels treated as regular symbols. However, an output ϵ label may also be read in T_1 without matching any actual transition in T_2. This case can be handled by the above rule after adding self-loops at every state of T_2 labeled on the inner tape by a new symbol ϵ^L and on the outer tape by ϵ and allowing transitions labeled by ϵ and ϵ^L to match. Similar self-loops are added to T_1 for matching input ϵ labels on T_2. However, this approach can result in redundant ϵ-paths since an epsilon label can match in the two above ways. The redundant paths must be *filtered* out because they will produce incorrect results in non-idempotent semirings (like the log semiring). We introduced the ϵ^L label to distinguish these two types of match in the filtering.

In [1], a *filter transducer* is introduced that is used with relabeling and the ϵ-free composition algorithm to correctly implement composition with ϵ labels. Our composition algorithm extends this by generalizing the *composition filter*.

Our algorithm takes as input two weighted transducers $T_1 = (\mathcal{A}, \mathcal{B}, Q_1, I_1, F_1, E_1, \lambda_1, \rho_1)$ and $T_2 = (\mathcal{B}, \mathcal{C}, Q_2, I_2, F_2, E_2, \lambda_2, \rho_2)$ over a semiring $\mathbb{K}$ and a composition filter $\Phi = (T_1, T_2, Q_3, i_3, \perp, \varphi, \rho_3)$, which has a set of filter states Q_3, a designated initial filter state i_3, a designated blocking filter state $\perp$, a transition filter $\varphi : E_1^L \times E_2^L \times Q_3 \rightarrow \overline{E}_1 \times \overline{E}_2 \times Q_3$ where $E_n^L = \bigcup_{q \in Q_n} E^L[q]$, $E^L[q_1] = E[q_1] \cup \{(q_1, \epsilon, \epsilon^L, \overline{1}, q_1)\}$ for each $q_1 \in Q_1$, $E^L[q_2] = E[q_2] \cup \{(q_2, \epsilon^L, \epsilon, \overline{1}, q_2)\}$ for each $q_2 \in Q_2$ and a final weight filter $\rho_3 : Q_3 \rightarrow \mathbb{K}$.

We shall see that the filter can be used in composition to block the expansion of some states (by entering the $\perp$ state) and modify the transitions and final weights (useful for optimizations).

The states in the output of composition are identified with triples of a state from each of the two input transducers and one from the filter. In particular, the algorithm outputs a weighted finite-state transducer $T = (\mathcal{A}, \mathcal{C}, Q, I, F, E, \lambda, \rho)$ implementing the composition of T_1 and T_2 where $Q \subseteq Q_1 \times Q_2 \times Q_3$ and $I \subseteq I_1 \times I_2 \times \{i_3\}$.

Figure 1 gives the pseudocode of this algorithm. E and F are all initialized to the empty set and grown as needed. The algorithm uses a queue S containing the set of state triples of states yet to be examined. The queue discipline of S is arbitrary and does not affect the termination of the algorithm. The state set Q is initially the set of triples of initial states of the original transducers and filter, as is I and S (line 1). Each time through the loop in lines 3-14, a new triple of states (q_1, q_2, q_3) is extracted from S (lines 3-4). The final weight of (q_1, q_2, q_3) is computed by $\otimes$-multiplying the final weights of q_1 and q_2 and the final filter weight when they are all final states (lines 5-7). Then, for each pair of transitions, the transition filter is first applied. If the new filter state is not the blocking state $\perp$ and a new transition is created from the filter-rewritten transitions (e_1', e_2') (line 14). If the destination state $(n[e_1'], n[e_2'], q_3')$ has not been found previously, it is added to Q and inserted in S (lines 11-13).

The composition algorithm presented here and several simple filters are available in the *OpenFst* library [10].

```
WEIGHTED-COMPOSITION($T_1, T_2$)
 1  $Q \leftarrow I \leftarrow S \leftarrow I_1 \times I_2 \times \{i_3\}$
 2  while $S \neq \emptyset$ do
 3      $(q_1, q_2, q_3) \leftarrow$ HEAD($S$)
 4      DEQUEUE($S$)
 5      if $(q_1, q_2, q_3) \in F_1 \times F_2 \times Q_3$ then
 6          $F \leftarrow F \cup \{(q_1, q_2, q_3)\}$
 7          $\rho(q_1, q_2, q_3) \leftarrow \rho_1(q_1) \otimes \rho_2(q_2) \otimes \rho_3(q_3)$
 8      $M \leftarrow$ $\{(e_1, e_2) \in E^L[q_1] \times E^L[q_2]$ such that
                $\varphi(e_1, e_2, q_3) = (e_1', e_2', q_3')$ with $q_3' \neq \perp\}$
 9      for each$(e_1, e_2) \in M$ do
10          $(e_1', e_2', q_3') \leftarrow \varphi(e_1, e_2, q_3)$
11          if $(n[e_1'], n[e_2'], q_3') \notin Q$ then
12              $Q \leftarrow Q \cup \{(n[e_1'], n[e_2'], q_3')\}$
13              ENQUEUE($S$, $(n[e_1'], n[e_2'], q_3')$)
14          $E \leftarrow E \cup$ $\{((q_1, q_2, q_3), i[e_1'], o[e_2'],$
                $w[e_1'] \otimes w[e_2'], (n[e_1'], n[e_2'], q_3'))\}$
15  return $T$
```

Figure 1: Pseudocode of the composition algorithm.

2.3. Composition Filters

In this section, we consider particular composition filters.

2.3.1. Trivial Filter

Filter Φ_{trivial} blocks no paths and leaves transitions and final weights unmodified. For Φ_{trivial}, let $Q_3 = \{0, \perp\}$, $i_3 = 0$, $\varphi(e_1, e_2, q_3) = (e_1, e_2, q_3')$ with $q_3' = 0$ if $o[e_1] = i[e_2] \in \mathcal{B}$ and $\perp$ otherwise, and $\rho(q_3) = \overline{1}$ for all $q_3 \in Q_3$. With this filter, the pseudocode in Figure 1 matches the simple epsilon-free composition algorithm given in [1].

Let us assume that the transitions at each state in T_2 are sorted according to their input label. The set M of transitions to be computed line 8 is simply equal to $\{(e_1, e_2) \in E[q_1] \times E[q_2] : o[e_1] = i[e_2]\}$. It can be computed by performing a binary search over $E[q_2]$ for each transition in $E[q_1]$. The time complexity of computing M is then $O(|E[q_1]| \log |E[q_2]| + |M|)$. Since each element in M will result in a transition in T, the worst-case time complexity of the algorithm can be expressed as: $O(|T|_Q d(T_1) \log d(T_2) + |T|_E)$. The space complexity of the algorithm is $O(|T|)$.

2.3.2. Epsilon-Matching Filter

Filter $\Phi_{\epsilon\text{-match}}$ handles epsilon labels, but disallows redundant epsilon paths, preferring those that match actual ϵ labels. It leaves transitions and final weights unmodified.

For $\Phi_{\epsilon\text{-match}}$, let $Q_3 = \{0, 1, 2, \perp\}$, $i_3 = 0$, $\rho(q_3) = \overline{1}$ for all $q_3 \in Q_3$, and $\varphi(e_1, e_2, q_3) = (e_1, e_2, q_3')$ where:

$$q_3' = \begin{cases} 0 & \text{if } (o[e_1], i[e_2]) = (x, x) \text{ with } x \in \mathcal{B}, \\ 0 & \text{if } (o[e_1], i[e_2]) = (\epsilon, \epsilon) \text{ and } q_3 = 0, \\ 1 & \text{if } (o[e_1], i[e_2]) = (\epsilon^L, \epsilon) \text{ and } q_3 \neq 2, \\ 2 & \text{if } (o[e_1], i[e_2]) = (\epsilon, \epsilon^L) \text{ and } q_3 \neq 1, \\ \perp & \text{otherwise.} \end{cases} \tag{5}$$

With this filter, the pseudocode in Figure 1 matches the composition algorithm given in [1] with the specified composition filter transducer. The complexity of the algorithm is the same as when using the trivial filter.

2.3.3. Label-Reachability Filter

When composing states q_1 in T_1 and q_2 in T_2, filter Φ_{reach} disallows following an epsilon-labeled path from q_1 that will fail to reach a non-epsilon label that matches some transition leaving state q_2. It leaves transitions and final weights unmodified. This solves Problem 1 described in the introduction. For simplicity, we assume there are no input ϵ labels in T_1.

For Φ_{reach}, let $Q_3 = \{0, \perp\}$, $i_3 = 0$, and $\rho(q_3) = \overline{1}$ for all $q_3 \in Q_3$. Define $r : Q_1 \times \mathcal{B} \rightarrow \{0, 1\}$ such that $r(x, q) = 1$

if there is a path π from q to some q' in T_1 with $o[\pi] = x$, otherwise let $r(x, q) = 0$. Then, let $\varphi(e_1, e_2, q_3) = (e_1, e_2, 0)$ if $o[e_1] = i[e_2]$ or $o[e_1] = \epsilon, i[e_2] = \epsilon^L$, and for some $e_2' \in E[p[e_2]], i[e_2'] \neq \epsilon$ and $r(i[e_2'], n[e_1]) = 1$. Otherwise let $\varphi(e_1, e_2, q_3) = (e_1, e_2, \perp)$.

Let us denote by $c_r(T_1)$ the cost of performing one reachability query in T_1 using r, by $S_r(T_1)$ the total space required for r, and by $d_\epsilon T_1$ the maximal number of output-ϵ transitions at a state in T_1. The worst-case time complexity of the algorithm is: $O(|T|_Q(d(T_1)\log d(T_2) + d_\epsilon(T_1)c_r(T_1)) + |T|_E)$, and the space complexity is $O(|T| + S_r(T_1))$.

There are different ways we can represent r and they will lead to different complexities for composition. We will assume for our analysis, whatever its representation, that r is precomputed and stored with T_1. In general, we exclude any T-specific precomputation from composition's time complexity.

Point Representation of r: Define $R_q = \{x \in \mathcal{B} : r(x, q) = 1\}$ for each state $q \in T_1$. If the labels in R_q are stored in a linked list, traversed linearly and each matched against sorted input labels in T_2 using binary search, then $c_r(T_1) = \max_q |R_q| \log d(T_2)$ and $S_r(T_1) = \sum_q |R_q|$.

Interval Representation of r: We can use intervals to represent R_q if $\mathcal{B} = [1, |\mathcal{B}|] \subset \mathbb{N}$ by defining $I_q = \{[x, y) : x, y \in \mathbb{N}, [x, y) \subseteq R_q, x - 1 \notin R_q, y \notin R_q\}$. If the intervals in I_q are stored in a linked list, traversed linearly and each matched against sorted input labels in T_2 using (lower-bound) binary search, then $c_r(T_1) = \max_q |I_q| \log d(T_2)$ and $S_r(T_1) = \sum_q |I_q|$.

Assuming the particular numbering of the labels is arbitrary, let permutation $\Pi : \mathcal{B} \to \mathcal{B}$ be a bijection that is used to relabel both T_1 and T_2 prior to composition. Among the $|\mathcal{B}|!$ different possible such permutations, some could result in far fewer intervals in I_q than others. In fact, there may exist a Π that results in one interval per I_q. Consider the $|\mathcal{B}| \times |Q_1|$ matrix $\mathbf{R}$ with $\mathbf{R}[i, j] = r(i, j)$. The condition that the I_q each contain a single interval is equivalent to the property that the ones in the columns of $\mathbf{R}$ are consecutive. A binary matrix $\mathbf{R}$ that has a permutation of rows that results in columns with consecutive ones is said to have the *Consecutive One's Property* (C1P). The problem has been extensively studied and has many applications [5, 6, 7, 8]. There are linear algorithms to find a permutation if it exists; the first, due to Booth and Lucker, was based on PQ-trees [5]. There are approximate algorithms when an exact solution does not exist [9]. Our speech applications to follow all admit C1P. As such, the interval representaion of r results in a significant complexity reduction over the point representation.

2.3.4. Label-Reachability Filter with Label Pushing

When matching an ϵ-transition e_1 in q_1 with an ϵ^L-loop in q_2, the Φ_{reach} filter allows this match if and only the set of transitions in q_2 that match the future in $n[e_1]$ is non-empty. In the special case where this set contains a unique transition e_2', the $\Phi_{\text{push-label}}$ filter allows e_1 to match e_2', resulting in the early output of $o[e_2']$. This solves Problem 2 described in the introduction.

For $\Phi_{\text{push-label}}$, let $Q_3 = \{\epsilon, \perp\} \cup \mathcal{B}$, $i_3 = \epsilon$ and $\rho(q_3) = \overline{1}$ if $q_3 = \epsilon$ and $\rho(q_3) = \overline{0}$ otherwise. Let $\varphi(e_1, e_2, q_3) = (e_1, e_2, \epsilon)$ if $q_3 = \epsilon$ and $o[e_1] = i[e_2]$, or if $q_3 = o[e_1] = \epsilon$, $i[e_2] = \epsilon^L$ and $|\{e \in E[q_2] : r(n[e_1], i[e]) = 1\}| \geq 2$, or if $q_3 = o[e_1] \neq \epsilon$ and $i[e_2] = \epsilon^L$. Let $\varphi(e_1, e_2, q_3) = (e_1, e_2, q_3)$ if $q_3 \neq \epsilon$, $o[e_1] = \epsilon$, $i[e_2] = \epsilon^L$ and $r(n[e_1], q_3) = 1$. Let $\varphi(e_1, e_2, \epsilon) = (e_1, e_2', i[e_2'])$ if $o[e_1] = \epsilon$, $i[e_2] = \epsilon^L$ and $\{e \in E[q_2] : r(n[e_1], i[e]) = 1\} = \{e_2'\}$. Otherwise, let $\varphi(e_1, e_2, q_3) = (e_1, e_2, \perp)$.

The complexity of the algorithm is the same as when using the label-reachability filter.

2.3.5. Label-Reachability Filter with Weight Pushing

Similarily, when matching an ϵ-transition e_1 in q_1 with an ϵ^L-loop in q_2 we can use r to compute the set of transitions in q_2 that match the furure in $n[e_1]$. The $\Phi_{\text{push-weight}}$ filter allows the early output of the sum of weights of these prospective matches. This solves Problem 3 described in the introduction. We assume that any element x in $\mathbb{K}$ admits a $\otimes$-inverse denoted by x^{-1}.

For $\Phi_{\text{push-weight}}$, let $Q_3 = \mathbb{K}$, $i_3 = \overline{1}$, $\perp = \overline{0}$ and $\rho(q_3) = q_3^{-1}$ for all q_3 in Q_3. Define $w_r : Q_1 \times Q_2 \to \mathbb{K}$ such that $w_r(q_1, q_2) = \bigoplus_{e \in E[q_2], r(q_1, i[e]) = 1} w[e]$. Then, let $\varphi(e_1, e_2, q_3) = (e_1, (p[e_2], i[e_2], o[e_2], w', n[e_2]), q_3')$ where: $q_3' = \overline{1}$ and $w' = q_3^{-1} \otimes w[e_2]$ if $o[e_1] = i[e_2]$, $q_3' = w_r(n[e_1], q_2)$ and $w' = q_3^{-1} \otimes q_3'$ if $o[e_1] = \epsilon$ and $i[e_2] = \epsilon^L$ and $q_3' = \overline{0}$ and $w' = w[e_2]$ otherwise.

The use of this filter can results in a significant increase in complexity over the label-reachability filter due to the cost of computing w_r for each potential ϵ-match. However, when $\mathbb{K}$ is also a ring (like the log semiring for instance) and when using the interval representation, the computational cost increase can be avoided by precomputing, for each transition e in T_2, the sum of the weight of all the transitions in $p[e]$ with input label strictly less than $i[e]$. The contribution of each interval in $I_{n[e_1]}$ to $w_r(n[e_1], q_2)$ can then be computed by finding the transitions in q_2 corresponding to the lower and upper-bound of the match with that interval and taking the $\oplus$-difference of the corresponding precomputed cumulative weights.

3. Speech Recognition Applications

3.1. Methods

In the previous section, we presented composition filters that solve the three problems, described in the introduction, that are encountered when using standard composition for the speech transducer construction in Eq. 2, Of course, we need all these problems solved together. For this, we created a composition filter that combines features of the several filters presented above into a filter that used interval-based reachability testing with label and weight pushing and allowed epsilons on both transducers in composition.

We then used this for the construction in Eq. 2. L is constructed by first building a transducer that is a union of unique word pronunciations (subscripting words with multiple pronuncations as needed). This clearly satisfies C1P. Determinization (by forming a tree), minimization as an automaton (by merging equivalent futures), closure of this transducer (by leaving non-trivial futures unchanged) and composition with the context dependency transducer C (by splitting states without changing their futures) preserve C1P when applied to L.

3.2. Experiments

We evaluate these composition alternatives with recognition transducers compiled statically offline and using run-time expansion in two LVCSR tasks.

The baseline acoustic model used for these experiments is trained on Perceptual Linear Prediction (PLP) cepstra, uses a linear discriminative analysis transform to project from 9 consecutive 13-dimensional frames to a 39-dimensional feature space and uses semi-tied covarianecs. The acoustic model is triphonic, using about 8k tied states, modeling emission using 16-component Gaussian mixture densities. In addition to the baseline acoustic model, a feature space speaker adaptive model is used.

The first task evaluated is a Broadcast News (BN) system trained on the 96 and 97 DARPA Hub4 acoustic model training sets (about 150 hours of data) and the 1996 Hub4 CSR language model training set (128M words). This system uses a Good-Turing smoothed 4-gram language model, pruned using the Seymore-Rosenfeld algorithm to about 8M n-grams for a

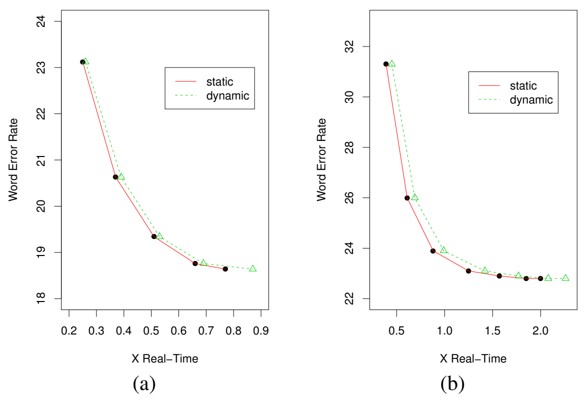

Figure 2: Real time factors for (a) Broadcast News and (b) spoken queries.

Language Model Size	WER
15M n-grams	22.8
23M n-grams	22.7
93M n-grams	22.6

Table 1: WER for different language model sizes.

vocabulary of about 71k words. The decoding strategy obtains a first transcript using the baseline model running with a narrow beam, then computes a feature space transform and maximum likelihood linear regression transform and then re-decodes the data with a large beam. This system obtains a 18.5% WER.

Using the 4-gram language model, offline compilation of the recognition transducer using Eq. 1 with standard composition took 7 minutes and 5.3G RAM, while using Eq. 2 with the generalized composition took 2.5 minutes and 2.9G RAM (using Eq. 2 with standard composition quickly exhausted all memory). Figure 2 compares using the statically-compiled recognition transducer versus the dynmaic expansion language model. It shows the WER as a function of the real time factor, after initial adaptation. The results on BN show on average, an $\sim$5% overhead for generating the search space dynamically. At wider beam this increases to $\sim$10%.

Next we explore the scalability of the algorithm using a spoken search query task. This uses a Katz smoothed 3-gram language model containing 2.4B n-grams for a vocabulary of 1M words. The increased vocabulary and language model size provide an ideal use case for studying the scalability of the proposed algorithm. The decoding strategy is a simple one-pass system.

The first set of experiments on the spoken query task uses a pruned version of the language model containing approximately 14M n-grams. Offline compilation of the recognition transducer using Eq. 1 took 10.5 minutes and 11.2G RAM while using Eq. 2 took 4 minutes and 5.3G RAM. Figure 2(b) shows the WER as a function of the real time factor for both the static and dynamically-generated search space evaluated on a 14,000 query test set. On average the composition has an $\sim$10% CPU overhead. The recognition transducer has 25.4M states but during decoding only a tiny fraction, 8K states per second (median), is explored.

A distinct advantage of computing the search space dyncamically is a significant reduction in the overall memory footprint of the recognizer. For the spoken query task, the static network of 25.4M states requires approximately 1.4GB of memory. When applying the composition dynamically, we only have to load the individual components of the composition (i.e., $C \circ \det(L)$ and G). This allows us to significantly scale the size of the language model G. In Table 1 we measure the WER as a function of language model size.

The explored part of the search space remains relatively constant independent of the size of the language model. The net effect is that we can increase the size of the language model G substantially without further increase in CPU. This allows us to explore substantially bigger language models in the first pass of the recognizer.

4. Discussion

In related prior work, Caseiro and Trancoso [2] developed a specialized composition with the lexicon L. In particular, they observed that if the word pronunciations were stored in a trie, the words that can be read per node form a lexicographic interval. They used this to disallow following epsilon paths that don't match words in the grammar. Cheng [3] and Oonishi [4] generalized this approach to work with more general transducers. Their approaches have similarities to ours but many details were left unspecified including how they computed and represented the sets R_q and used them in composition. While complexities were not provided, their speech recognition experiments showed considerable overhead to their compositions. This could be due to a less efficient r representation or weight-pushing method. To our knowledge, ours is the first dynamically-expanded WFST-based recognizer that has a small overhead compared to using static transducers in a state-of-the-art LVCSR system.

5. Acknowledgements

We thank Mehryar Mohri for suggesting using a generalized composition filter for solving problems such as those addressed here.

6. References

[1] M. Mohri, F. Pereira, and M. Riley, "Speech recognition with weighted finite-state transducers," in *Handbook of Speech Processing*, Y. H. Jacob Benesty, Mohan Sondhi, Ed. Springer, 2008, pp. 559–582.

[2] D. Caseiro and I. Trancoso, "A specialized on-the-fly algorithm for lexicon and language model composition," in *IEEE Trans. on Audio, Speech and Lang. Proc.*, vol. 14, no. 4, 2006, pp. 1281–1291.

[3] O. Cheng, J. Dines, and M. Doss, "A generalized dynamic composition algorithm of weighted finite state transducers for large vocabulary speech recognition," in *Proc. ICASSP*, vol. 4, 2007, pp. 345–348.

[4] T. Oonishi, P. Dixon, K. Iwano, and S. Furui, "Implementation and evaluation of fast on-the-fly wfst composition algorithms," in *Proc. Interspeech*, 2008, pp. 2110–2113.

[5] K. Booth and G. Lueker, "Testing for the consecutive ones property, interval graphs, and graph planarity using pq-tree algorithms," *J. of Computer and System Sci.*, vol. 13, pp. 335–379, 1976.

[6] M. Habib, R. McConnell, C. Paul, and L. Viennot, "Lex-BFS and partition refinement with appli- cations to transitive orientation, interval graph recognition and consecutive ones testing," *Theor. Comput. Sci.*, vol. 234, pp. 59–84, 2000.

[7] W.-L. Hsu and R. McConnell, "PC trees and circular-ones arrangements," *Theor. Comput. Sci.*, vol. 296, no. 1, pp. 99–116, 2003.

[8] J. Meidanis, O. Porto, and G. Telles, "On the consecutive ones property," *Discrete Appl. Math.*, vol. 88, pp. 325–354, 1998.

[9] M. Dom and R. Niedermeier, "The search for consecutive ones submatrices: Faster and more general," in *ACID*, 2007, pp. 43–54.

[10] C. Allauzen, M. Riley, J. Schalkwyk, W. Skut, and M. Mohri, "OpenFst Library," *http://www.openfst.org*, 2007.