Speech Recognition
Lecture 11: Dynamic Recognition Transducer Construction

Cyril Allauzen
Google, NYU Courant Institute
allauzen@cims.nyu.edu
Recognition Transducer Construction

• The models $C$, $L$, $G$ can be combined and optimized with weighted finite-state composition and determinization as:

$$C \circ \text{det}(L \circ G)$$  \hspace{1cm} (1)

• An alternative construction, producing an equivalent transducer, is:

$$C \circ \text{det}(L) \circ G$$  \hspace{1cm} (2)

If $G$ is deterministic, Eq. 2 could be as efficient as Eq. 1 and avoids the determinization of $L \circ G$, greatly saving time and memory and allowing fast dynamic combination (useful in applications).

• However, standard composition presents three problems with Eq. 2:
  1. Determinization of $L$ moves back word labels creating delay in matching and creating (possibly very many) useless composition paths
  2. The delayed word labels in $L$ produce a much larger composed machine when $G$ is an n-gram LM.
  3. The delayed word labels push back the grammar weights along paths in the composed machine to the detriment of ASR pruning.
Composition Example
Dynamic Recognition Transducers

▷ Three algorithmic solutions

- **Composition with reachability filters**
  Allows efficient dynamic or static composition of $\text{det}(L)$ and $G$ [Allauzen et al., 2009].

- **Pre-initialized composition**
  Hybrid between *dynamic* and *static* constructions to provide better time-space tradeoff [Allauzen and Riley, 2013].

- **Integrated composition and replacement**
  Allows dynamically modifying the language model $G$ on a per-utterance basis [Aleski et al., 2015].
This Lecture

- Generalized Composition with Filters
- Pre-Initialization
- Composition with Replacement
Generalized Composition with Filters

- **Composition**: a crucial algorithm when dealing weighted finite-state transducers, used to:
  → apply finite-state models to inputs or combine cascaded models

- **Problem**: some transducers do not compose efficiently
  → classical algorithm creates many non-coaccessible states
  → waste both time and space

- **Previous work**: special-purpose algorithms designed for:
  speech recognition [Caseiro and Trancoso, 06; Cheng et al., 07; McDonough et al. 07; Oonishi et al., 08] and speech synthesis [Allauzen et al., 04]

- **This work**: generalized algorithm using a composition *filter*
  → filter is applied at each state and decides whether to continue
  → a fully general filter would have to do all composition work
  → specific filters tailored to particular but common cases
  → subsumes several specializations in an efficient way
Composition Reminder

• Definition: \((T_1 \circ T_2)(x, y) = \bigoplus_{z \in \Sigma^*} T_1(x, z) \otimes T_2(z, y)\)

• Example: [Tropical semiring \((\mathbb{R}, \min, +, \infty, 0)\)]

\[
\begin{array}{c}
\text{0} \quad \text{a:b/0.1} \quad \text{c:a/0.3} \\
\text{1} \quad \text{a:a/0.4} \\
\text{2} \quad \text{b:a/0.2} \quad \text{b:b/0.5} \\
\text{3/0.6} \\
\end{array}
\begin{array}{c}
\text{0} \quad \text{b:c/0.3} \\
\text{1} \quad \text{a:b/0.4} \\
\text{2/0.7} \\
\end{array}
\begin{array}{c}
\text{(0, 0)} \quad \text{a:c/0.4} \\
\text{(1, 1)} \quad \text{c:b/0.7} \\
\text{(1, 2)} \quad \text{a:b/1} \\
\text{(3, 2)/1.3} \\
\end{array}
\begin{array}{c}
\text{T_1} \\
\text{T_2} \\
\text{T_1 \circ T_2} \\
\end{array}
\]

• Algorithm:
  – States: \((q_1, q_2)\) with \(q_1\) in \(T_1\) and \(q_2\) in \(T_2\)
  – Transitions:
  \((q_1, a, b, w_1, q_1')\) and \((q_2, b, c, w_2, q_2') \sim ((q_1, q_2), a, c, w_1 \otimes w_2, (q_1', q_2'))\)
Handling $\epsilon$-transitions in Composition

- An $\epsilon$-transition in $T_1$ (resp. $T_2$) can be matched in $T_2$ (resp. $T_1$) by an $\epsilon$ transition or by staying in the same state

▷ As if there were an $\epsilon$ self-loop at each state in $T_1$ and $T_2$
  → labeled $\epsilon^L$

- Issue: Results in redundant $\epsilon$-paths in $T_1 \circ T_2$

→ [Mohri, Peirera and Riley, 96] used a filter transducer
Generalized Composition Algorithm

- Extend composition by generalizing the notion of composition filter

- **Composition Filter:**
  \[ \Phi = (T_1, T_2, Q_3, i_3, \bot, \varphi) \]
  - \( Q_3 \): set of filter states with \( i_3 \) initial and \( \bot \) blocking.
  - \( \varphi : (e_1, e_2, q_3) \mapsto (e'_1, e'_2, q'_3) \): transition filter

- **Algorithm:**
  - **States:** \( (q_1, q_2, q_3) \) with \( q_1 \) in \( T_1 \), \( q_2 \) in \( T_2 \) and \( q_3 \) a filter state.
  - **Transitions:** \( e_1 \) transition in \( q_1 \), \( e_2 \) in \( q_2 \) such that \( \varphi(e_1, e_2, q_3) = (e'_1, e'_2, q'_3) \) with \( q'_3 \neq \bot \)
    \[ \rightarrow ((q_1, q_2, q_3), i[e'_1], o[e'_2], w[e'_1] \otimes w[e'_2], (n[e'_1], n[e'_2], q'_3)) \]

- **2 types of filters:**
  - prevent redundant \( \epsilon \)-paths
  - look-ahead to prevent creating some non co-accessible states
Weighted-Composition\((T_1, T_2)\)

1. \(Q \leftarrow I \leftarrow S \leftarrow I_1 \times I_2 \times \{i_3\}\)
2. \textbf{while} \(S \neq \emptyset\) \textbf{do}
3. \((q_1, q_2, q_3) \leftarrow \text{HEAD}(S)\)
4. Dequeue\((S)\)
5. \textbf{if} \((q_1, q_2, q_3) \in F_1 \times F_2 \times Q_3\) \textbf{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. \(M \leftarrow \{(e_1, e_2) \in E^L[q_1] \times E^L[q_2] \text{ such that } \varphi(e_1, e_2, q_3) = (e'_1, e'_2, q'_3) \text{ with } q'_3 \neq \bot\}\)
9. \textbf{for each}(e_1, e_2) \in M \textbf{ do}
10. \((e'_1, e'_2, q'_3) \leftarrow \varphi(e_1, e_2, q_3)\)
11. \textbf{if} \((n[e'_1], n[e'_2], q'_3) \not\in Q\) \textbf{ 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. \textbf{return} \(T\)
Trivial Filter

- Allows all matching paths

- Filter $\Phi_{\text{trivial}}$:

  $Q_3 = \{0, \bot\}$, $i_3 = 0$ and $\phi(e_1, e_2, 0) = \begin{cases} 
  0 & \text{if } o[e_1] = i[e_2] \\
  \bot & \text{otherwise}
\end{cases}$

  $\rightarrow$ basic $\epsilon$-free composition algorithm

- For each state $(q_1, q_2)$ in $T = T_1 \circ T_2$, we need to compute:

  $$M = \{(e_1, e_2) \in E[q_1] \times E[q_2] : o[e_1] = i[e_2]\}$$

  $\rightarrow$ use a binary search over $E[q_2]$ for each transition in $E[q_1]$

  $\rightarrow$ complexity $O(|E[q_1]| \log |E[q_2]| + |M|)$

- Complexity: Time: $O(|T|_Q d(T_1) \log d(T_2) + |T|_E)$

  Space: $O(|T|)$

  $|\cdot|_Q$: number of states, $|\cdot|_E$: number of transitions, $|\cdot| = |\cdot|_Q + |\cdot|_E$

  $d(\cdot)$: maximal out-degree
Epsilon-Matching Filter

- Disallows redundant $\epsilon$-paths, favoring matching actual $\epsilon$-transitions

- **Filter $\Phi_{\epsilon\text{-match}}$:**
  
  \[ Q_3 = \{0, 1, 2, \perp\}, \ i_3 = 0 \] and $\varphi(e_1, e_2, q_3) = (e_1, e_2, q'_3)$ where:

  \[
  q'_3 = \begin{cases} 
  0 & \text{if } (o[e_1], i[e_2]) = (x, x) \text{ with } x \in B \text{ or if } (o[e_1], i[e_2]) = (\epsilon, \epsilon) \text{ and } q_3 = 0, \\
  1 & \text{if } (o[e_1], i[e_2]) = (\epsilon^L, \epsilon) \text{ and } q_3 \neq 2, \\
  2 & \text{if } (o[e_1], i[e_2]) = (\epsilon, \epsilon^L) \text{ and } q_3 \neq 1, \\
  \perp & \text{otherwise.}
  \end{cases}
  \]

→ composition algorithm of [Mohri, Peirera and Riley, 96]
Epsilon-Sequence Filter

- Read $\epsilon$’s on $T_1$ first, followed by $\epsilon$’s on $T_2$

- Filter $\Phi_{\text{seq}}$:
  \[ Q_3 = \{0, 1, \perp\}, \ i_3 = 0 \text{ and } \varphi(e_1, e_2, q_3) = (e_1, e_2, q'_3) \]

  \[ q'_3 = \begin{cases} 
  0 & \text{if } (o[e_1], i[e_2]) = (x, x) \text{ with } x \in B, \\
  0 & \text{if } (o[e_1], i[e_2]) = (\epsilon, \epsilon^L) \text{ and } q_3 = 0, \\
  1 & \text{if } (o[e_1], i[e_2]) = (\epsilon^L, \epsilon), \\
  \perp & \text{otherwise.} 
  \end{cases} \]

- **Complexity:**
  - **Time:** $O(|T|, Qd(T_1) \log d(T_2) + |T|_E)$
  - **Space:** $O(|T|)$
String-Potential Filter

- **String Potential:**
  \[ p_i(q) / p_o(q) \text{ longest common prefix of input/output of paths from } q \]

  Match two transitions iff potentials at destination are comparable

- **Filter \( \Phi_{sp} \):**
  \[ Q_3 = \{0, \perp\}, \ i_3 = 0, \text{ and } \varphi(e_1, e_2, q_3) = (e_1, e_2, q'_3) \text{ where:} \]
  \[ q'_3 = \begin{cases} 
  0 & \text{if } p_o(n[e_1]) \land p_i(n[e_2]) \in \{p_o(n[e_1]), p_i(n[e_2])\}, \\
  \perp & \text{otherwise.}
\end{cases} \]

  \[ \begin{array}{c}
  0 \quad a \quad 1 \\
  \quad a \quad 2 \\
  \quad 3 \quad b \quad 6 \\
  \quad 4 \quad b \quad 5 \\
  \quad d \quad c \quad 7 \\
  \end{array} \quad \begin{array}{c}
  0,0 \quad a \\
  \quad 1,1 \\
  \quad 2,2 \\
  \quad 3,2 \\
  \quad 4,3 \\
  \quad 5,3 \\
  \quad 6,3 \\
  \quad 7,4 \\
  \end{array} \]

- **Complexity:**
  \[ \begin{array}{c}
  \text{Time: } O(|T|Q d(T_1) \log d(T_2) + |T|E \min(\mu_1, \mu_2)) \\
  \text{Space: } O(|T| + |T_1|Q \mu_1 + |T_2|Q \mu_2)
\end{array} \]
Transition-Look-Ahead Filter

- Look ahead one step in the future

- $L_i(q)/L_o(q)$ set of input/output labels of transitions in $q$

- Filter $\Phi_{tr-la}$: \( Q_3 = \{0, \perp\} \), \( i_3 = 0 \), and \( \varphi(e_1, e_2, q_3) = (e_1, e_2, q'_3) \)

where:
\[
q'_3 = \begin{cases} 
0 & \text{if } L_o(n[e_1]) \cap L_i(n[e_2]) \neq \emptyset \text{ or } \epsilon \in L_o(n[e_1]) \cup L_i(n[e_2]), \\
\perp & \text{otherwise.}
\end{cases}
\]

- Complexity: Time: \( O(|T|Qd(T_1) \log d(T_2) + |T|E \log |B|) \)
  
  Space: \( O(|T| + (|T_1|_Q + |T_2|_Q) \log |B|) \)
Label-Reachability Filter

- Disallows following an $\epsilon$-path in $q_1$ that will fail to reach a non-$\epsilon$ label that match some transition in $q_2$

- **Label-Reachability** $r : Q_1 \times B \rightarrow \{0, 1\}$
  
  $$r(q, x) = \begin{cases} 
  1 & \text{if there exists a path from } q \text{ to some } q' \text{ with output label } x \\
  0 & \text{otherwise}
  \end{cases}$$

- **Filter $\Phi_{reach}$**: Same as $\Phi_{trivial}$ except when $o[e_1] = \epsilon$ and $i[e_2] = \epsilon^L$ then
  
  $$\varphi(e_1, e_2, 0) = (e_1, e_2, q_3) \text{ with } q_3 = \begin{cases} 
  0 & \text{if there exist } e'_2 \text{ in } q_2 \text{ such that } r(n[e_1], i[e'_2]) = 1 \\
  \bot & \text{otherwise}
  \end{cases}$$

---

![Diagram of Label-Reachability Filter](image)
Label-Reachability Filter

- **Complexity of composition**
  
  Time: $O(|T|Q(d(T_1) \log d(T_2) + d_\epsilon(T_1)c_r(T_1)) + |T|E)$
  
  Space: $O(|T| + S_r(T_1))$
  
  with $c_r(T_1)$ cost of one query of $r$ and $S_r(T_1)$ the total space for $r$

- **Representation of $r$**
  
  - Point representation: $R_q = \{x \in B : r(x, q) = 1\}$
    
    ▶ inefficient in time and space
  
  - Interval representation:
    
    $I_q = \{[x, y) : x, y \in \mathbb{N}, [x, y) \subseteq R_q, x - 1 \notin R_q, y \notin R_q\}$
    
    ▶ efficiency depends on the number of interval for each $R_q$
  
    * one interval per state trivial for a tree - found by DFS
  
    * one interval per state possible if $C1P$ holds
  
    * a modification of the Hsu’s (2002) $C1P$ Test gives a greedy algorithm for minimizing the number of intervals per state.
Label-Reachability Filter with Label Pushing

- When matching an $\epsilon$-transition $e_1$ with an $\epsilon^L$-loop $e_2$:
  - if there exists a unique $e'_2$ in $q_2$ such that $r(n[e_1], i[e'_2]) = 1$,
  - then allows matching $e_1$ with $e'_2$ instead

  → early output of $o[e'_2]$

- **Filter $\Phi_{\text{push-label}}$:** $Q_3 = \Sigma \cup \{\epsilon, \bot\}$ and $i_3 = \epsilon$
  - the filter state encodes the label that has been consumed early
Label-Reachability Filter with Weight Pushing

- When matching an $\epsilon$-transition $e_1$ with an $\epsilon^L$-loop $e_2$:  
  $\triangleright$ outputs early the $\oplus$-sum of the weight of the prospective matches

- Reachable weight $w_r : (q_1, q_2) \mapsto \bigoplus_{e \in E[q_2], r(q_1, i[e]) = 1} w[e]$

- Filter $\Phi_{\text{push-weight}}$: $Q_3 = \mathbb{K}$, $i_3 = \overline{1}$ and $\bot = 0$  
  $\triangleright$ the filter state encodes the weight that has been outputted early  
  if $o[e_1] = \epsilon$ and $i[e_2] = \epsilon^L$, $q_3' = q_3^{-1} \otimes w_r(n[e_1], q_2)$
Implementation

• Efficient computation of $w_r$
  - Requires fast computation of $s_q(x, y) = \bigoplus_{e \in E[q], i[e] \in [x, y]} w[e]$ for $q$ in $T_2$, $x$ and $y$ in $B = \mathbb{N}$
  - Achieved by precomputing $c_q(x) = \bigoplus_{e \in E[q], i[e] < x} w[e]$

\[ \triangleright s_q(x, y) = c_q(y) - c_q(x) \]

• Combining the different filters
  \[ \triangleright \text{by taking their cross-product } i.e. \Phi_{\text{e-seq}} \times \Phi_{\text{push-label}} \times \Phi_{\text{push-weight}} \]
Recognition Transducer Construction

- Lookahead much more efficient with unique pronunciations (C1P).
  - Create a lexicon where each pronunciation of a word is treated as a separate word: \(r\ eh\ d \rightarrow \text{read}_1,\ r\ iy\ d \rightarrow \text{read}_2\)
  - Create a unique pronunciation transducer \(U\) that maps each unique-pron word to original word.
  - Offline construct \(CL = det(C \circ L)\) and \(UG = U \circ G\).

- Build \(T = CL \circ UG\) statically or dynamically using the \(\epsilon\)-sequence filter combined with reachability filter with label and weight pushing.
Speech Recognition Example

- A large-vocabulary speech recognition task: DARPA Broadcast News
  - $\Omega$: set of BN English words, $|\Omega| = 70,897$
  - $\Pi$: set of English phonemes, $|\Pi| = 46$
  - $\Upsilon$: set of English tri-phonemic acoustic models, $|\Upsilon| = 20,910$

  - $G$: a 4-gram *language model*, a weighted automaton over $\Omega$
    - assigns a probability to a sentence, i.e. sequence of words
      $\rightarrow 2,213,539$ states and $10,225,015$ transitions
  - $L$: a minimal deterministic *lexicon transducer* over $\Pi \times \Omega$
    - maps phonemic pronunciations to word symbols
      $\rightarrow 63,283$ states and $145,710$ transitions
  - $C$: a min. det. *context-dependency transducer* over $\Upsilon \times \Pi$
    - maps tri-phonemic model sequences to phonemic sequences
      $\rightarrow 1454$ states and $88,840$ transitions
## Experimental Results

| T          | composition filter         | |T|_Q   | |T|_E | time (s) | mem. (MB) |
|------------|---------------------------|-----------------|---------|---------|----------|----------|
| C ◦ α      | trivial                   | 47,021,923      | 47,021,922 | 48.45   | 4704.0   |
|            | string-potential          | 1,043,734       | 1,043,733  | 8.97    | 351.0    |
| C ◦ L      | trivial                   | 1,952,555       | 3,527,612  | 2.77    | 225.0    |
|            | transition-look-ahead     | 120,489         | 149,972    | 0.84    | 33.4     |
| L ◦ G      | epsilon-sequencing        | ?               | ?         | >7200.0 | >32,768.0|
|            | label-reachability        | 30,884,222      | 39,965,633 | 177.93  | 3612.9   |
|            | lab.-reach. w/ lab.-push. | 13,377,323      | 22,151,870 | 113.72  | 1885.9   |

\[
\begin{align*}
\alpha & \text{ is a random string in } \Pi^{1000000} \\
C & \text{ highly non-deterministic on output: all transitions out of } xy \text{ have output } y \\
\text{input string potential is } \epsilon \text{ at most states in } L \\
\text{non-}\epsilon \text{ output labels are at the end of long output-}\epsilon \text{ paths in } L
\end{align*}
\]
# Recognition Experiments

<table>
<thead>
<tr>
<th>Broadcast News</th>
<th>Spoken Query Task</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Acoustic Model</strong></td>
<td><strong>Acoustic Model</strong></td>
</tr>
<tr>
<td>• Trained on 96 and 97 DARPA Hub4 AM training sets.</td>
<td>• Trained on &gt; 1000hrs of voice search queries</td>
</tr>
<tr>
<td>• PLP cepstra, LDA analysis, STC</td>
<td>• PLP cepstra, LDA analysis, STC</td>
</tr>
<tr>
<td>• Triphonic, 8k tied states, 16 components per state</td>
<td>• Triphonic, 4k tied states, 4 - 128 components per state</td>
</tr>
<tr>
<td>• Speaker adapted (both VTLN + CMLLR)</td>
<td>• Speaker independent</td>
</tr>
<tr>
<td><strong>Language Model</strong></td>
<td><strong>Language Model</strong></td>
</tr>
<tr>
<td>• 1996 Hub4 CSR LM training sets</td>
<td>• Trained on &gt; 1B words of google.com and voice search queries</td>
</tr>
<tr>
<td>• 4-gram language model pruned to 8M n-grams</td>
<td>• 1 million word vocabulary</td>
</tr>
<tr>
<td></td>
<td>• Katz back-off model, pruned to various sizes</td>
</tr>
</tbody>
</table>
### Recognition Experiments

<table>
<thead>
<tr>
<th>Precomputation before recognition</th>
<th>Broadcast News</th>
<th>Spoken Query Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>Construction method</td>
<td>Time</td>
<td>RAM</td>
</tr>
<tr>
<td>Static</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(1) with standard composition</td>
<td>7 min</td>
<td>5.3G</td>
</tr>
<tr>
<td>(2) with generalized composition</td>
<td>2.5 min</td>
<td>2.9G</td>
</tr>
<tr>
<td>Dynamic</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(2) with generalized composition</td>
<td>none</td>
<td>none</td>
</tr>
</tbody>
</table>

#### Graphs

**Broadcast News**
- **X Real-Time**
- **Word Error Rate**
- Static (red) and Dynamic (green) curves

**Spoken Query Task**
- **X Real-Time**
- **Word Error Rate**
- Static (red) and Dynamic (green) curves
Recognition Experiments

- A small part of the recognition transducer is visited during recognition:

<table>
<thead>
<tr>
<th>Spoken Query Task</th>
<th>Static Number of states in recognition transducer</th>
<th>25.4M</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynamic</td>
<td>Number of states visited per second</td>
<td>8K</td>
</tr>
</tbody>
</table>

- Very large language models can be used in first-pass:

Word error rate as function of LM size
(with Ciprian Chelba and Boulos Harb)
Prior Work

- **Caseiro and Trancoso (IEEE Trans. on ASLP 2006):** they developed a specialized composition for a pronunciation lexicon $L$. If pronunciations are stored in a trie, then the words readable from a node form a lexicographic interval, which can be used to disallow non-coaccessible epsilon paths.

- **Cheng, et al. (ICASSP 2007); Oonishi, et al (Interspeech 2008):** they use methods apparently similar to ours, but many details are left unspecified, such as what are the representation of the reachable label sets. There are no published complexities, but the published results show a very significant overhead to the dynamic composition compared to a static recognition transducer.

- **This method:**
  - uses a very efficient representation of the label sets
  - uses a very efficient computation of the weight pushing
  - has a small overhead between static and dynamic composition
Implementation in OpenFst

OpenFst: an open-source weighted transducer library

- Default composition filter: epsilon-sequencing filter
- Filter can be easily and efficiently changed via templated options
  → to use the $\epsilon$-matching filter:
    ```
    ComposeFstOptions<StdArc, MatchComposeFilter> opts;
    ComposeFst<StdArc> result(t1, t2, opts);
    ```
- The filters\(^a\) described here are implemented in OpenFst:
  ```
  http://openfst.org/twiki/bin/view/FST/FstAdvancedUsage#Look_Ahead_Matchers
  ```
- Users can also define new filters by creating a class meeting the composition filter interface

\(^a\)String-potential filter not available yet
This Lecture

- Generalized Composition with Filters
- Pre-Initialization
- Composition with Replacement
Pre-Initialized Composition

- **Goal:** More efficient time-space tradeoff when combining the context-dependent lexicon and the language model that is searched during recognition.

- **Method:** Hybrid between the *dynamic* (or *on-the-fly* or *lazy*) composition and the *static* composition of the component models represented as weighted finite-state transducers (WFSTs) by pre-computing part of the recognition transducer and leaving the balance to be computed during recognition.

- **Results:** The time overhead of purely dynamic expansion can be reduced by over six-fold with only a 20% increase in memory in a collection of large-vocabulary recognition tasks available on the Google Android platform.
Static vs Dynamic Composition

- Recognition transducer construction:

\[ CL = C \circ Det(L) \]
\[ T = CL \circ G \]

- Static composition: fully combined ahead of time – very time-efficient in use, but requires the most space [8].

- Dynamic composition: only the small part of the composition visited during utterance recognition is created – saves considerable space (\(|CL| + |G| \ll |CL \circ G|\)) but uses more time [5, 9, 3]. Allows fast grammar modification prior to recognition.

- Hybrid composition: Finer-grain tradeoff between time and space than these two extremes [5].
Notations

- **Restriction of a transducer:**
  Let $T|_R = (\mathcal{A}, \mathcal{B}, Q_R, I_R, F_R, E_R, \lambda_R, \rho_R)$ denote a restriction of the transducer $T$ to $R \subseteq Q$ with
    - $Q_R = R \cup \{n[e] | e \in E \land p[e] \in R\}$,
    - $I_R = I \cap R$,
    - $F_R = F \cap R$, $E_R = \{e \in E | p[e] \in R\}$,
    - $\lambda_R = \lambda : I_R \rightarrow \mathbb{K}$ and
    - $\rho_R = \rho : F_R \rightarrow \mathbb{K}$.
  Thus $R$ restricts the states from which transitions may exit.
Pre-Initialized Composition

- **Algorithm:** Computes $T = T_1 \circ T_2$ given the initialization $T|_R$. Pseudo-code differs from standard composition by:
  - Initializing the set of states, the initial state, the final states and transitions from those in $T|_R$ at lines 1-4
  - Skipping computing the transitions leaving a state already in $T|_R$ at line 9; instead finding any successor states that need to be enqueued directly from $T|_R$ at lines 20-22.
Pre-Initialized Composition: Pseudo-code

\text{Weighted-Composition}(T_1, T_2, T_R) \\
1 \quad Q \leftarrow Q_R \\
2 \quad F \leftarrow F_R \\
3 \quad E \leftarrow E_R \\
4 \quad I \leftarrow I_R \\
5 \quad S \leftarrow I \\
6 \quad \textbf{while } S \neq \emptyset \textbf{ do} \\
7 \quad \quad (q_1, q_2) \leftarrow \text{Head}(S) \\
8 \quad \quad \text{Dequeue}(S) \\
9 \quad \quad \textbf{if } (q_1, q_2) \not\in R \textbf{ then} \\
10 \quad \quad \quad \textbf{if } (q_1, q_2) \in F_1 \times F_2 \textbf{ then} \\
11 \quad \quad \quad \quad \text{Insert}(F, (q_1, q_2)) \\
12 \quad \quad \quad \rho(q_1, q_2) \leftarrow \rho_1(q_1) \otimes \rho_2(q_2) \\
13 \quad \quad \quad \textbf{for each } (e_1, e_2) \in E[q_1] \times E[q_2] \textbf{ s. t. } o[e_1] = i[e_2] \textbf{ do} \\
14 \quad \quad \quad \quad \textbf{if } (n[e_1], n[e_2]) \not\in Q \textbf{ then} \\
15 \quad \quad \quad \quad \quad \text{Insert}(Q, (n[e_1], n[e_2])) \\
16 \quad \quad \quad \quad \quad \text{Enqueue}(S, (n[e_1], n[e_2])) \\
17 \quad \quad \quad \quad \quad w' \leftarrow w[e_1] \otimes w[e_2] \\
18 \quad \quad \quad \quad \quad \text{Insert}(E, ((q_1, q_2), i[e_1], o[e_2], w', (n[e_1], n[e_2]))) \\
19 \quad \quad \textbf{else} \\
20 \quad \quad \quad \textbf{for each } e \in E_R[(q_1, q_2)] \textbf{ do} \\
21 \quad \quad \quad \quad \textbf{if } n[e] \not\in R \textbf{ then} \\
22 \quad \quad \quad \quad \quad \text{Enqueue}(S, n[e]) \\
23 \quad \textbf{return } T
Pre-Initialized Composition

- **Implementation:**
  - **Naive:** $Q$, $F$, and $E$ represented as sets with $\text{INSERT}(Q, q)$ defined as $Q \leftarrow Q \cup \{q\}$, etc.
  
  - **Efficient/Shared:** Initialization step in lines 1-3 will be constant time and $T|R$ shared in parallel calls if we represent $Q = (Q_{\text{static}}, Q_{\text{dyn}})$ as a pair of sets denoting a *static* and *dynamic* part. Then the initialization on line 1, $Q \leftarrow Q_R$ becomes $Q \leftarrow (Q_R, \emptyset)$, $\text{INSERT}(Q, q)$ is defined as $Q \leftarrow (Q_{\text{static}}, Q_{\text{dyn}} \cup \{q\})$ and $q \in Q$ means $q \in Q_{\text{static}} \lor q \in Q_{\text{dyn}}$. Similar data structures for $F$ and $E$ can be defined.

- The algorithm can be generalized to allow for different filters that handle epsilon transitions and various forms of lookahead [9, 3].
Pre-Initialized Composition: Implementation

• Implemented in OpenFst [4], a C++ library for weighted finite-state transducers. Composition is templated on the data structures that represent $Q$, $F$, $E$ and the composition filter.

• In the standard case, simple set and filter representations are the default.

• In the speech recognition setting:
  – more complex set representations, with distinct static and dynamic components, and a lookahead filter ensuring efficient matching are used [3].
  – $T|_R$ represents the pre-built, static part of the recognition transducer shared among all utterances
  – The balance of the recognition transducer needed for each utterance is built dynamically and discarded at the end of each utterance.
Initialization Method: State Statistics

- Uses state statistics observed during decoding of representative utterances.

- Given an utterance $\mu_i$, let $O_i \subseteq Q$ denote states enqueued during recognition of $\mu_i$ in the fully-dynamic version ($R = I$):

- Given $M$ utterances and a count threshold $n$, choose the initialization set as:

  $$N(q) = \sum_{i=1}^{M} 1_{q \in O_i}$$

  $$R = \{ q \in Q \mid N(q) \geq n \}$$

  where $N(q)$ counts the number of utterances in which state $q$ was enqueued and $R$ contains all states whose count meets the threshold.

- Various values of $n$ can be used to trade-off time versus space.
Initialization Method: State Probabilities

- Uses the state probabilities intrinsic to the language and lexical models.

- If $T$ is stochastic: The shortest distance from $I$ to $q$:

$$
\delta(q) = \bigoplus_{\pi \in P(q)} w[\pi]
$$

is the probability (or negative log probability) of reaching state $q$ from $I$ where $P(q)$ is the set of paths from $I$ to $q$ [7].

- Given a threshold $p$, we can choose the initialization set as:

$$
R = \{ q \in Q \mid \delta(q) \geq p \}.
$$

Various values of $p$ can be used to trade-off time versus space.

- If $T$ is not stochastic, it can be first *pushed* to make it so [8].
Experiments: Data and Models

• Test sets:
  – Word error rate: 27,327 transcribed utterances (153,796 words) sampled randomly and anonymized from Voice tasks available on the Google Android including SMS, Voice Search, Google Maps, and Facebook.
  – Speed: 1K utterances
  – State frequency statistics 10K separate utterances

• Models:
  – Acoustic model: DNN/HMM hybrid model whose input features are the concatenation of 26 consecutive frames of 40-dimensional log filterbank energies
  – Language model: Interpolated 5-gram model with 22.5 million n-grams trained from sources including typed data sources (e.g. web search queries or SMS) and spoken data sources consisting of ASR results from anonymized utterances filtered by recognition confidence.
  – Context-dependent lexicon: 4 million states and 3.4 million transitions
  – Static recognition transducer: 45 million states and 98 million transitions.
Experiments: Methods

- Decoding performed using:
  - multi-threaded server allowing processing of multiple utterances simultaneously
  - multi-threaded load tester that supplied the test utterances at a specified parallelism.

- Dynamic expansion involves work that cannot be shared, impacting both time and space with increasing parallelism. Results reported with a range of parallelism since it is an important factor to evaluate in practical systems that serve users at scale.

- Pruning beam set so that there are few search errors, comparable to our production settings.

- All timings performed on an HP Z600 workstation with 24 cores and 50 gbytes of RAM.
Experiments: Varied Initialization Results

parallelism = 1

parallelism = 2

parallelism = 4

parallelism = 8
Experiments: Varied Initialization Results

- Fully-dynamic approach uses about 1/3 the memory of the fully-static approach but has about 10% overhead relative to no parallelism and about 27% overhead with a parallelism of four.

- State statistics-based pre-initialization gives a range of time-memory operating points depending on the utterance count cutoff $n$. With $n = 20$ and a parallelism of 4, the excess recognition time is cut by over a factor of 6 compared to fully dynamic expansion with only about a 20% increase in RAM usage.

- With $n = 20$, about 1 million states are present in the initialization set $R$ or about 2% of the fully-expanded $T$.

- The state probabilities pre-initialization also gives a range of time-memory operating points but performs worse than the state statistics approach.
### Experiments: Varied Language Model Size Results

| n-grams     | WER  | % time overhead | \( |R| \) |
|--------------|------|-----------------|-------|
|              |      | fully-dynamic   | \( n = 20 \) | \( n = 20 \) |
| 12,324,719   | 11.0%| 26.4% 6.7%      | 1,017,531 |
| 22,329,335   | 10.7%| 28.5% 4.3%      | 1,034,218 |
| 45,280,869   | 10.4%| 31.8% 7.3%      | 1,027,841 |
| 70,594,261   | 10.4%| 36.3% 6.5%      | 1,009,975 |
Experiments: Varied Language Model Size Results

- The affect of the language model size is shown both when the pre-initialization set is $I$ and when it is chosen using the frequently seen states method with a utterance count cutoff of $n = 20$ using a parallelism of four in testing.

- The smaller LMs are derived from the largest through relative entropy pruning.

- The excess recognition time compared to the fully-static construction grows with increasing language model size for the fully-dynamic case.

- Using the state statistics method with $n = 20$, the number of states in $R$ is roughly constant and the time overhead varies between about 4-7% with no clear pattern with respect to LM size.
Discussion

- The dynamic approaches show increasing overhead with increasing parallelism up to 4 due to thread contention in the expansion. Expansion overhead decreases some with a parallelism of 8 probably due to other contentions dominating.

- The state frequency initialization method is better than the state probability method probably because:
  - the development set for collecting statistics well-matches the speed test set in the former whereas the LM is drawn from a wider range of sources for the latter.
  - the statistical method can take into account acoustic confusions unlike the probabilistic method.

- The time overhead is a weak function of the language model size for the state frequency initialization method. This suggests the overhead for a given $|R|$ is primarily determined by the underlying corpus on which the LM was trained.

- The fully-dynamic method can be used when the grammar is modified pre-recognition. The hybrid composition expansion methods can also be used so long as the pre-initialization is restricted to states in the original grammar.
This Lecture

- Generalized Composition with Filters
- Pre-Initialization
- Composition with Replacement
Dynamic Modification of the Recognition Transducer

- There is a need for a dynamic LM that changes on a per-user/per-utterance basis to take into account some contextual information, e.g. the user contacts.

- **Approach:**
  - A class-based LM where some of the class models are utterance-specific.
  - Allows us to insert the contextual information in the right context, e.g. contacts name after “call” or “send message to”.
  - Use the replacement operation to dynamically update the top-level LM with the relevant class models.
Class-based Language Model and Replacement

• A top-level (root) language model $G_r$: an automaton over words and class labels.

• For each class $c \in C$, a class model $G_c$: an automaton over words.

▷ The class models $G_c$ can be utterance-dependent.

• Utterance-specific language model used for recognition:

$$G = \text{Replace}(G_r, (G_c)_{c \in c})$$

• Issue: Efficiently build a recognition transducer using this $G$. 
Replacement Example

$G_{\$ROOT}$

$G_{\$CONTACTS}$

Replace($G_{\$ROOT}, G_{\$CONTACTS}$)
**CLG-level Replacement**

- Given a class-based language model specification \((G_c)_{c \in N}\), we build an individual recognition transducer \(T_c\) for each class label/nonterminal \(c\):

\[
T_c = CL \circ G_c
\]  

and then define the recognition transducer:

\[
T = \text{Replace}((T_c)_{c \in N}).
\]

- **Major issue:**
  - The left and right phonetic contexts of a phrase in \(c\) are always the start and end phone (silence).
  - The right phonetic context of the word preceding that phrase as well as the left phonetic context of the word following that phrase will also be incorrect.
G-level Replacement

• Leverage the fact that the composition algorithm with reachability filters [3] can be applied on-demand to a dynamic language model:

\[
T = CL \circ \text{Replace}((G_c)_{c \in N}).
\]  

(5)

• **Issue:** Performing both composition and replacement on demand in this manner significantly increases the run-time overhead.

• **Solution:** Pre-Initialization.
  
  – States in \( T \) are of the form \((p, (q, s))\) where \( p \) is a state in \( CL \) and \((q, s)\) a state in Replace((\(G_c\))_{c \in N}).
  
  – A state of the form \((q, \epsilon)\) corresponds to a state in the main language model and hence does not change between utterances.
  
  – Restrict to only pre-initialize states of the form \((p, (q, \epsilon))\).

▷ Benefit fully from the pre-initialization while keeping the ability to update the class-specific models on a per-utterance basis.
Handling Out-of-Vocabulary Words

• Issue:
  – Class-specific model might contains OOVs.
  ▷ OOVs extremely likely for contact names.
  – $CL$ is built and optimized offline, difficult to add new names to it while keeping the result optimized and updating the reachability filter data accordingly.
Handling Out-of-Vocabulary Words

- **Solution**: a form of $LG$-level replacement.
  - Add to $L$ single-phone monophone word for every CI phone.
  - Create an FST $M$ mapping each monophone word to the corresponding CI phone.
  - Build a $c$-specific restricted lexicon $L_c$ containing pronunciations for the words in $G_c$ and we build a new replace component for $c$ as:
    \[
    G'_c = M \circ \text{Det}(L_c) \circ G_c. \tag{6}
    \]
  - Given $C' \subseteq C$, the set of classes for which we want to allow OOVs, we build the recognition transducer as:
    \[
    T = CL \circ \text{Replace}((G_r, G_c, G'_c)_{c \in C \setminus C', c' \in C'}) \tag{7}
    \]
    where $r$ denotes the root nonterminal.
Experimental Setup

• For each languages, two test sets were collected: [1]
  – **Contacts:** utterances including a contacts voice action, e.g. “call Michael”.
  – **Anti-contacts:** utterances confusable with a contacts voice action, e.g. “call MacDonald’s” or “call of duty”.

▷ **Goal:** Improve contacts voice action recognition while reducing over-triggering on non-contacts voice queries and commands.
## Experimental Results

<table>
<thead>
<tr>
<th>Language</th>
<th>Test Set</th>
<th>NUtts</th>
<th>CLG WER[%]</th>
<th>G WER[%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>German</td>
<td>CONTACTS</td>
<td>2,928</td>
<td>16.3</td>
<td>12.3</td>
</tr>
<tr>
<td></td>
<td>CONTACTS-W-SEL</td>
<td>2,928</td>
<td>10.2</td>
<td>9.6</td>
</tr>
<tr>
<td></td>
<td>ANTI-CONTACTS</td>
<td>2,937</td>
<td>33.3</td>
<td>26.0</td>
</tr>
<tr>
<td>British</td>
<td>CONTACTS</td>
<td>2,588</td>
<td>22.4</td>
<td>21.2</td>
</tr>
<tr>
<td>English</td>
<td>CONTACTS-W-SEL</td>
<td>2,588</td>
<td>20.5</td>
<td>16.8</td>
</tr>
<tr>
<td></td>
<td>ANTI-CONTACTS</td>
<td>2,740</td>
<td>47.0</td>
<td>31.7</td>
</tr>
<tr>
<td>American</td>
<td>CONTACTS</td>
<td>10,670</td>
<td>10.7</td>
<td>7.0</td>
</tr>
<tr>
<td>English</td>
<td>CONTACTS-W-SEL</td>
<td>10,670</td>
<td>7.9</td>
<td>6.1</td>
</tr>
<tr>
<td></td>
<td>ANTI-CONTACTS</td>
<td>4,650</td>
<td>15.7</td>
<td>10.7</td>
</tr>
</tbody>
</table>
Conclusion

▷ A general approach for building recognition transducer for class-based models.

- Correctly models cross-word dependencies.
- Allows for dynamically modifying class grammars on a per-utterance basis.
- Handles out-of-vocabulary in class models.
- Compatible with pre-initialization: good time/memory tradeoff.
References


