Weighted Transducers in Speech and Language Processing

Michael Riley - riley@google.com
Google Research, NYC

September 12, 2014

Oriental COCOSDA 2014 - Phuket, Thailand

Thanks to Cyril Allauzen, Mehryar Mohri, Fernando Pereira, Johan Schalkwyk, Richard Sproat
Overview

- I: Historical Remarks
- II: Finite-State Transducers: Algorithms and Applications
- III: Pushdown Transducers: Algorithms and Applications
- IV: Current Research and Conclusion
Weighted Transducers in Speech and Language Processing

1. Historical Remarks
Weighted Transducers in Speech and Language Processing

Weighted finite-state transducers (WFSTs):

• Core component of many modern ASR systems: used by Google, Nuance, IBM, AT&T among others. Millions of daily users.

• Used as well in speech synthesis, optical character recognition, machine translation among others

• Over 2700 citations for top ten Google Scholar WFST papers

• Thousands of software downloads: (www.openfst.org)
Finite-State Automata/Transducers

• Theory:
  – Automata: Huffman, 1954; Moore, 1956; Kleene, 1956; Rabin and Scott, 1959

• Speech/NLP Applications:
WFSTs in Speech Recognition - I

• **Goals (1993):** common set of representations, algorithms and tools for weighted finite automata

• **Representation choice:**
  – Automaton/Acceptor: rational operations (union, concatenation, closure); determinization and minimization; cascades formed by recursive replacement (e.g., pronunciations into grammars)
  – Transducer: cascades formed by composition
  – Context-dependency: how to model phone $y/x_z$, phone $y$ in the context of $x$ and $z$: 

    ![Diagram](image)

• **Publication:** Pereira, Riley, and Sproat, 1994. “Weighted rational transductions and their application to human language processing”
WFSTs in Speech Recognition - II

- Non-determinism: How to deal with phonetic redundancy in LVCSR?
  - Tree-structured lexicon (Ney, 1992)
  - General determinization algorithm (Mohri, 1994; Mohri and Riley, 1997)

```plaintext
WeightedDeterminization(A)
1  i' ← {(i, λ(i)) : i ∈ I}
2  λ'(i') ← Φ
3  S ← {i'}
4  while S ≠ ∅ do
5    p' ← Head(S)
6    Dequeue(S)
7    for each x ∈ i[E[Q[p']]] do
8      w' ← ⊕{v ⊗ w : (p, v) ∈ p', (p, x, w, q) ∈ E}
9      q' ← {(q, ⊕{w'-1 ⊗ (v ⊗ w) : (p, v) ∈ p', (p, x, w, q) ∈ E}) : q = n[e], i[e] = x, e ∈ E[Q[p']]}
10     E' ← E' ∪ {(p', x, w', q')}
11    if q' ∉ Q' then
12      Q' ← Q' ∪ {q'}
13      if Q[q'] ∩ F ≠ ∅ then
14        F' ← F' ∪ {q'}
15        ρ'(q') ← ⊕{v ⊗ ρ(q) : (q, v) ∈ q', q ∈ F}
16     Enqueue(S, q')
17  return A'
```
WFSTs in Speech Recognition - III

- **Static compilation:** Can you combine a cross-word context-dependent lexicon and and n-gram language model into a single transducer without it blowing up?
  - Less than 2 times larger than the grammar with suitable determinization and minimization: Mohri and Riley, 1997

- **Dynamic compilation:** Can you combine the determinized lexicon and n-gram language model on-the-fly efficiently?
  - Caserio and Trancoso, 2001; Oonishi et al, 2009; Allauzen and Riley, 2010.
II. *Finite-State Transducers: Algorithms and Applications*
Motivation

- **Finite-State Automata/Acceptors:** Compact representations of regular (rational) languages that are efficient to search. Examples: pattern matching (grep, PCRE), tokenization, compression.

- **Finite-State Transducers:** Compact representations of rational binary relations that are efficient to search and combine/cascade. Examples: dictionaries, context-dependent rules

- **Weighted Automata:** Weights typically encode uncertainty as e.g. probabilities. Examples: n-gram language models, language translation models.

- **Algorithms:** Efficient methods for constructing, combining, optimizing and searching:
  - **OpenFst:** open-source C++ FST library: [www.openfst.org](http://www.openfst.org).
  - **OpenGrm:** open-source C++ grammar libraries: [www.opengrm.org](http://www.opengrm.org)
    - NGram: n-gram language modeling
    - Thrax: finite-state (and beyond) rule compiler
Finite-State Automata

\[ L(A) = \{ a^n b^m \mid n, m \in \mathbb{N} \} \]

- A transition is labeled with a regular symbol or the empty string, \( \epsilon \).
- **Acceptance condition:** There exists a path from the initial state (denoted by a bold circle) to a final state (denoted by a double circle).
Finite-State Automata

- A **finite-state automaton** $A$ is a 5-tuple $(\Sigma, Q, E, I, F)$ with
  - $\Sigma$, input alphabet
  - $Q, I \subseteq Q, F \subseteq Q$: states, initial states and final states
  - $E \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times Q$: transitions $[e = (p[e], i[e], n[e]) \in E]$

- $\pi = e_1 \ldots e_k \in E^*$ is a **path** in $A$ if $n[e_i] = p[e_i]$ for $1 \leq i < k$
  $[p[\pi] = p[e_1], n[\pi] = n[e_n] \text{ and } i[\pi] = i[e_1]i[e_2]\ldots i[e_n]]$

- A string $x \in \Sigma^*$ belongs to $L(A)$, the **language accepted by** $A$, if there exists a path $\pi$ such that $p[\pi] \in I, n[\pi] \in F, i[\pi] = x$.

$L(A) = \{a^n b^m \mid n, m \in \mathbb{N}\}$
Weighted Finite-State Transducers

- Each transition \( e \) in a \textit{weighted finite-state transducer} \( T \) has additionally:
  - an output label \( o[e] \in \Delta \cup \{\epsilon\} \) and
  - a weight \( w[e] \in \mathbb{R} \cup \{\infty\} \)

- The \textit{weight associated by} \( T \) to a pair of strings \( (x, y) \) is

\[
T(x, y) = \min_{\pi \in P(x, y)} w[\pi] \quad \text{with}
\]

\[
P(x, y) = \{ \pi \mid p[\pi] \in I, n[\pi] \in F, i[\pi] = x, o[\pi] = y \},
\]

\[
w[\pi] = \lambda(\pi) + w[e_1] + w[e_2] \ldots + w[e_n] + \rho(\pi)
\]

\[
T(a^n b^m, c^{m+n}) = n + m
\]
Semirings

More generally, the weight \( w[e] \in \mathbb{K} \), a semiring, and

\[
T(x, y) = \bigoplus_{\pi \in P(x, y)} w[\pi]
\]

with

\[
P(x, y) = \{ \pi \mid p[\pi] \in I, n[\pi] \in F, i[\pi] = x, o[\pi] = y \},
\]

\[
w[\pi] = \lambda(\pi) \otimes w[e_1] \otimes w[e_2] \ldots \otimes w[e_n] \otimes \rho(\pi)
\]

A semiring \((\mathbb{K}, \oplus, \otimes, 0, 1)\) = a ring that may lack negation.

- **Sum**: to compute the weight of a sequence (sum of the weights of the paths labeled with that sequence).

- **Product**: to compute the weight of a path (product of the weights of constituent transitions).

Weighted Transducers in Speech and Language Processing II: FST - Algorithms 13
### Semirings

<table>
<thead>
<tr>
<th><strong>Semiring</strong></th>
<th><strong>Set</strong></th>
<th><strong>⊕</strong></th>
<th><strong>⊗</strong></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>{0, 1}</td>
<td>\lor</td>
<td>\land</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Probability</td>
<td>\mathbb{R}_+</td>
<td>+</td>
<td>\times</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Log</td>
<td>\mathbb{R} \cup {-\infty, +\infty}</td>
<td>\oplus_{\log}</td>
<td>+</td>
<td>+\infty</td>
<td>0</td>
</tr>
<tr>
<td>Tropical</td>
<td>\mathbb{R} \cup {-\infty, +\infty}</td>
<td>min</td>
<td>+</td>
<td>+\infty</td>
<td>0</td>
</tr>
</tbody>
</table>

\(\oplus_{\log}\) is defined by: \(x \oplus_{\log} y = -\log(e^{-x} + e^{-y})\)
Properties

- **Recognition power**
  - A language is recognizable by an FSA iff it is regular

- **Closure properties**
  - Closed under sum (union), product (concatenation), kleene-closure and reversal, composition, intersection

- **Decidability results**
  - Membership \( x \in L(A) \) is decidable
  - Equivalence of two (deterministic) FSAs is decidable
  - Equivalence of two FSTs is undecidable
WFST Algorithms

- **Union**: Combines in alternation
- **Concatenation**: Combines in sequence
- **Closure**: Arbitrary repetition
- **Reversal**: Reverses paths
- **Inversion**: Inverts binary relation
- **Projection**: Projects relation to domain/range
- **Composition**: Relational composition of two transducers
- **Determinization**: Creates equivalent deterministic transducer
- **Epsilon removal**: Removes $\epsilon$-transitions
- **Shortest distance**: Finds single-source shortest-distances
- **Shortest path**: Finds single-source shortest path
- **Pruning**: Prunes states and transitions by path weight
- **Connection**: Removes non-accessible/non-coaccessible states

▷ Described in this talk
Composition – Illustration

- **Definition:**

\[(T_1 \circ T_2)(x, y) = \min_{z \in \Sigma^*} (T_1(x, z) + T_2(z, y))\]

- **Example:**
Composition Algorithm

- Assuming that $T_2$ has no input-$\epsilon$ transitions
  
  - **States:** $(q_1, q_2)$ with $q_1$ in $T_1$ and $q_2$ in $T_2$

- **Transitions:**
  
  - Regular symbol: $a \in \Sigma \cup \{\epsilon\}$, $b \in \Delta$ and $c \in \Gamma \cup \{\epsilon\}$
    
    \((q_1, a, b, w_1, q'_1) \text{ and } (q_2, b, c, w_2, q'_2) \leadsto ((q_1, q_2), a, c, w_1 + w_2, (q'_1, q'_2))\)

  - Epsilon: $a \in \Sigma \cup \{\epsilon\}$
    
    \((q_1, a, \epsilon, w_1, q'_1) \text{ and stay in } q_2 \leadsto ((q_1, q_2), a, \epsilon, w_1, (q'_1, q_2))\)

- When both sides have epsilons, an *epsilon filter* is required

- **Complexity:** $O(|T_1||T_2|)$ in the worst case
*Weighted Determinization:* Classical subset construction modified to ensure correct weights

*Transducer Determinization:* Application of weighted determinization: output strings in $\Delta^*$ treated as weights over the *string semiring* $S$.

*Weighted Transducer Determinization:* Application of weighted determinization: Weights are in $K \times S^*$

*Applies only to determinizable automata/transducers*

<table>
<thead>
<tr>
<th>Semiring</th>
<th>Set</th>
<th>$\oplus$</th>
<th>$\otimes$</th>
<th>$\overline{0}$</th>
<th>$\overline{1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>$\Delta^* \cup {\infty}$</td>
<td>$\wedge$</td>
<td>$\cdot$</td>
<td>$\infty$</td>
<td>$\epsilon$</td>
</tr>
<tr>
<td>Product</td>
<td>$K_1 \times K_2$</td>
<td>$\oplus_1 \times \oplus_2$</td>
<td>$\otimes_1 \times \otimes_2$</td>
<td>$(\overline{0}, \overline{0})$</td>
<td>$(\overline{1}, \overline{1})$</td>
</tr>
</tbody>
</table>

$\wedge$ is the longest common prefix. The string semiring is a *left semiring*. 

Weighted Transducers in Speech and Language Processing II: FST - Algorithms
Shortest Distance and Shortest Path

• Given an FST, computes
  – **Shortest Path**: the shortest accepting path in $T$
  – **Shortest Distance**: the weight of the shortest accepting path in $T$

• Shortest path algorithm is derived from the shortest-distance algorithm by keeping track of a parent pointer

• Complexity of the algorithm depends on the *queue discipline*; linear in $|T|$ if $T$ is acyclic.
Shortest Distance Algorithm

- \( d[q] \): minimum weight of a path from the unique initial state \( i \) to \( q \)

\[
\text{ShortestDistance}(T) \\
1 \quad \text{for each } q \in Q \text{ do} \\
2 \quad \quad d[q] \leftarrow \infty \\
3 \quad d[i] \leftarrow 0 \\
4 \quad S \leftarrow i \\
5 \quad \text{while } S \neq \emptyset \text{ do} \\
6 \quad \quad q \leftarrow \text{HEAD}(S) \\
7 \quad \quad \text{DEQUEUE}(S) \\
8 \quad \quad \text{for each } e \in E[q] \text{ do} \\
9 \quad \quad \quad \text{RELAX}(n[e], d[q] + w[e], S) \\
10 \quad \text{return } d[f] \\
\]

\( \triangleright f \text{ is the unique final state} \)

\[
\text{RELAX}(q, w, S) \\
1 \quad \text{if } d[q] > w \text{ then} \quad \triangleright \text{if } w \text{ is a better estimate of the distance from } q \text{ to } i \\
2 \quad \quad d[q] \leftarrow w \quad \triangleright \text{update } d[q] \\
3 \quad \quad \text{if } q \not\in S \text{ then} \quad \triangleright \text{enqueue } q \text{ in } S \text{ if needed} \\
4 \quad \quad \quad \text{ENQUEUE}(S, q) \\
\]
Application: First-pass ASR

- **Context-Dependent Triphone Transducer C:**

- **Pronunciation Lexicon Transducer L:**

- **N-Gram Word Grammar G:** *OpenGrm NGram FSAs* (www.opengrm.org)
Recognition Transducer Construction

• **Method 1:**

\[ C \circ Det(L \circ G) \]

– Built statically offline due to cost of determinization
– Highly efficient in time - all graph construction pre-compiled

• **Method 2:**

\[ Det(C \circ L) \circ G \]

– Outermost composition with algorithm as described behaves badly
– Solution: generalized composition with *lookahead filter*: precompute labels reachable from a (lexicon) state.
– Allows dynamic composition during recognition, trading space for (some) time
## Recognition Experiments

<table>
<thead>
<tr>
<th>Broadcast News</th>
<th>Voice Search</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 $&gt; 1000$hrs 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>Construction method</th>
<th>Broadcast News</th>
<th>Voice Search</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>RAM</td>
<td>Result</td>
</tr>
<tr>
<td>Static</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(1) $C \circ \text{Det}(L \circ G)$</td>
<td>7 min</td>
<td>5.3G</td>
</tr>
<tr>
<td>(2) $\text{Det}(C \circ L) \circ G$</td>
<td>2.5 min</td>
<td>2.9G</td>
</tr>
<tr>
<td>Dynamic</td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td>(2) $\text{det}(C \circ L) \circ G$</td>
<td>none</td>
<td>none</td>
</tr>
</tbody>
</table>

### Graphs

**Broadcast News**
- **Static** (red line)
- **Dynamic** (green dashed line)

**Voice Search**
- **Static** (red line)
- **Dynamic** (green dashed line)
Applications: Various

General form of pipeline:

\[
\begin{align*}
\text{Result} & = \text{ShortestPath}(\text{Input} \circ \text{Det}(\text{Model})) \\
\text{Input} & = \text{string} \lor \text{lattice} \\
\text{Model} & = \text{Model}_1 \circ \ldots \circ \text{Model}_n
\end{align*}
\]

<table>
<thead>
<tr>
<th>System</th>
<th>Result</th>
<th>Input</th>
<th>Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Second-pass ASR</td>
<td>one-best</td>
<td>speech lattice</td>
<td>n-gram</td>
</tr>
<tr>
<td>Let.-to-sound conversion</td>
<td>graphemes</td>
<td>phonemes</td>
<td>phonotactics \circ pair n-gram</td>
</tr>
<tr>
<td>Case restoration</td>
<td>lowercase text</td>
<td>mixed-case text</td>
<td>pair n-gram</td>
</tr>
<tr>
<td>Diacritic restoration</td>
<td>ascii text</td>
<td>latin-1 text</td>
<td>pair n-gram</td>
</tr>
<tr>
<td>Spelling correction</td>
<td>text</td>
<td>corrected text</td>
<td>confusion model \circ n-gram</td>
</tr>
</tbody>
</table>
Weighted Transducers in Speech and Language Processing

III. *Pushdown Transducers: Algorithms and Applications*
Motivation

• **Weighted Pushdown Automata/Acceptors:** Compact representations of *context-free* languages that are efficient to search. Examples: CF LMs, semantic grammars

• **Weighted Pushdown Transducers:** Compact representations of *simple synchronous context-free* binary relations that are efficient to search and combine/cascade. Examples: machine-translation lattices, parse forests

• **Algorithms:** Efficient methods for constructing, combining, optimizing and searching the above and in combination with finite automata.

• **Of Special Interest:** PDAs that represent regular languages - retain compact, recursive representation but can admit better algorithms

**OpenFst PDT Extension:** Open-source C++ library: [pdt.openfst.org](http://pdt.openfst.org).
Pushdown Automata

\[ L(A) = \{ a^n b^n \mid n \in \mathbb{N} \} \]

- A transition can be labeled by regular symbol or by a stack operation
- Stack operations are represented by pairs of open and close parentheses
  - *open parenthesis*: pushing a symbol on the stack
  - *close parenthesis*: popping from the stack the matching open paren.
- **Acceptance condition**: parentheses must balance along an accepting path
  > equivalent to accepting on empty stack at final states
Dyck Languages

• A Dyck language consists of “well-formed” or “balanced” strings over a finite number of pairs of parentheses. Thus

\[
( [ ( ) ( ) ] \{ \} [ ] ) ( )
\]

is in the Dyck language over 3 pairs of parentheses.

• **Dyck language** $D_A$: Let $n \in \mathbb{N}$, $A = \{a_1, \ldots, a_n\}$ and $\overline{A} = \{\overline{a}_1, \ldots, \overline{a}_n\}$. A string $x \in (A \cup \overline{A})^*$ belongs to $D_A$ iff it is recognized by the CFG:

\[
S \rightarrow \epsilon, \quad S \rightarrow SS \quad \text{and} \quad S \rightarrow aS\overline{a} \quad \text{for all} \quad a \in A.
\]

• For $b \in A \cup \overline{A}$, let

\[
\overline{b} = \begin{cases} 
\overline{a}_i & \text{if } b = a_i \\
 a_i & \text{if } b = \overline{a}_i 
\end{cases} \quad \left[\text{i.e. } \overline{\overline{a}_i} = a_i\right]
\]

and let $c_A(x)$ be the string obtained by iteratively deleting from $x$ all factors of the form $a\overline{a}$ with $a \in A$. 

Pushdown Automata

- A *pushdown automaton* \( A \) is a 7-uple \((\Sigma, \Pi, \Pi, Q, E, I, F)\) with
  - \( \Sigma, \Pi \) and \( \Pi \): input, open parenthesis and close parenthesis alphabets
  - \( Q, I \subseteq Q, F \subseteq Q \): states, initial states and final states
  - \( E \subseteq Q \times (\Sigma \cup \Pi \cup \Pi \cup \{\epsilon\}) \times Q \): transitions \([e = (p[e], i[e], n[e]) \in E]\)

- \( \pi = e_1 \ldots e_k \in E^* \) is a *path* in \( A \) if \( n[e_i] = p[e_i] \) for \( 1 \leq i < k \)
  \([p[\pi] = p[e_1], n[\pi] = n[e_n] \) and \( i[\pi] = i[e_1]i[e_2] \ldots i[e_n]\)

- A string \( x \in \Sigma^* \) belongs to \( L(A) \), the *language accepted by \( A \)*, if there exists a path \( \pi \) such that \( p[\pi] \in I, n[\pi] \in F \), \( i[\pi]|_\Sigma = x \) and \( i[\pi]|_{\Pi \cup \Pi} \in D_\Pi \)

\[
L(A) = \{a^n b^n \mid n \in \mathbb{N}\}
\]
Weighted Pushdown Transducers

- Each transition $e$ in a weighted pushdown transducer $T$ has additionally:
  - an output label $o[e] \in \Delta \cup \Pi \cup \bar{\Pi} \cup \{\epsilon\}$ and
  - a weight $w[e] \in \mathbb{R} \cup \{\infty\}$

If $i[e]$ or $o[e]$ is a parenthesis then $i[e] = o[e]$

- The weight associated by $T$ to a pair of strings $(x, y)$ is
  \[
  T(x, y) = \min_{\pi \in P(x, y)} w[\pi]
  \] with

\[
P(x, y) = \left\{ \pi \mid p[\pi] \in I, n[\pi] \in F, i[\pi]|_\Sigma = x, o[\pi]|_\Delta = y \text{ and } i[\pi]|_{\Pi \cup \bar{\Pi}} \in D_\Pi \right\}
\]

\[
T(a^n b^n, c^{2n}) = 3n
\]
Properties

- **Recognition power**
  - A language is recognizable by a PDA iff it is context-free
  - A transduction is recognizable by a PDT iff it is a simple syntax-directed translation

- **Closure properties**
  - Closed under sum (union), product (concatenation), kleene-closure and reversal
  - Closed under composition/intersection with finite-state transducers/automata

- **Decidability results**
  - Membership \((x \in L(A))\) is decidable
  - Equivalence of two PDAs or PDTs is undecidable
  - Rationality of \(L(A)\) is undecidable
  - Existence of equivalent deterministic PDA is undecidable
Bounded-Stack Pushdown Transducers

- A pushdown transducer $T$ has **bounded stack** if there exists $K \in \mathbb{N}$ such that for any partially balanced path $\pi$ from the initial state in $T$ the number of not yet balanced parentheses is less than $K$:

$$|c_\Pi(i[\pi]|_{\Pi \cup \overline{\Pi}})| < K$$

▷ If $T$ has bounded stack, then it represents a rational transduction

<table>
<thead>
<tr>
<th>bounded-stack</th>
<th>rational</th>
<th>$a^<em>b^</em>$</th>
</tr>
</thead>
<tbody>
<tr>
<td>not bounded-stack</td>
<td>rational</td>
<td>$a^<em>b^</em>$</td>
</tr>
<tr>
<td>not bounded-stack</td>
<td>not rational</td>
<td>${a^n b^n \mid n \in \mathbb{N}}$</td>
</tr>
</tbody>
</table>
WPDT Algorithms

Union: FST alg.
Concatenation/Closure: FST alg.
Reversal: trivial changes to FST alg.
Inversion/Projection: FST alg.
Expansion: PDT-specific alg.
Replacement: PDT-specific alg.
▷ Composition: non-trivial changes to FST alg.
▷ Determinization: PDT-specific alg.
Epsilon removal: FST alg.
▷ Shortest distance: PDT-specific alg.
▷ Shortest path: PDT-specific alg.
Pruning: PDT-specific alg. required
Connection: PDT-specific alg. required
▷ described in this talk

◊ Requires bounded-stack input.
Composition

- Definition:

\[(T_1 \circ T_2)(x, y) = \min_{z \in \Sigma^*} (T_1(x, z) + T_2(z, y))\]

▷ If \(T_1\) is a PDT and \(T_2\) is an FST,
then \(T_1 \circ T_2\) can be represented by a PDT

- Algorithm:
  - Bar-Hillel construction
  - Same algorithm as composition of finite-state transducers with \(\epsilon\)-transitions
    → parentheses are treated as different kind of epsilons
Composition Algorithm

▶ Assuming that $T_2$ has no input-$\epsilon$ transitions

• States: $(q_1, q_2)$ with $q_1$ in $T_1$ and $q_2$ in $T_2$

• Transitions:
  
  – Regular symbol: $a \in \Sigma \cup \{\epsilon\}$, $b \in \Delta$ and $c \in \Gamma \cup \{\epsilon\}$
    
    $(q_1, a, b, w_1, q'_1)$ and $(q_2, b, c, w_2, q'_2) \leadsto ((q_1, q_2), a, c, w_1 + w_2, (q'_1, q'_2))$
  
  – Epsilon: $a \in \Sigma \cup \{\epsilon\}$
    
    $(q_1, a, \epsilon, w_1, q'_1)$ and stay in $q_2 \leadsto ((q_1, q_2), a, \epsilon, w_1, (q'_1, q_2))$
  
  – Parenthesis: $a \in \Pi \cup \overline{\Pi}$
    
    $(q_1, a, a, w_1, q'_1)$ and stay in $q_2 \leadsto ((q_1, q_2), a, a, w_1, (q'_1, q_2))$

▶ When both sides have epsilons, an epsilon filter generalized to handle parentheses can be used

• Complexity: $O(|T_1||T_2|)$ in the worst case
Determinization Algorithm

- Application of weighted determinization: parentheses in $(\Pi \cup \overline{\Pi})^*$ are treated as weights over the parenthesis semiring.
- Applies to bounded-stack PDAs (in general undecidable)

<table>
<thead>
<tr>
<th>SEMIRING</th>
<th>SET</th>
<th>$\oplus$</th>
<th>$\otimes$</th>
<th>$\overline{0}$</th>
<th>$\overline{1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parenthesis</td>
<td>$(\Pi \cup \overline{\Pi})^* \cup {\infty}$</td>
<td>$\cup$</td>
<td>$\cdot$</td>
<td>$\infty$</td>
<td>$\epsilon$</td>
</tr>
</tbody>
</table>

$a \cdot \overline{a} = \epsilon$ if $a \in \Pi$. 
Shortest Distance and Shortest Path

- Given a bounded-stack PDT, computes
  - **Shortest Path**: the shortest balanced accepting path in $T$
  - **Shortest Distance**: the weight of the shortest balanced accepting path in $T$

- Naive algorithm has exponential complexity

- **Idea**: Memoize
  - Similar idea used in shortest path over hypergraphs or RTNs

![Diagram of a weighted transducer with labeled arcs and nodes representing states and transitions.]
Shortest Distance Algorithm

- $d[q, s]$ : minimum weight of a balanced path from $s$ to $q$
- $B[q, a]$ : set of close parenthesis transitions that can balanced an incoming open parenthesis transition in $q$ labeled by $a \in \Pi$

\[
\text{ShortestDistance}(T) \\
\begin{align*}
1 & \text{ for each } q \in Q \text{ and } a \in \Pi \text{ do} \\
2 & \quad B[q, a] \leftarrow \emptyset \\
3 & \quad \text{GetDistance}(T, i) \quad \triangleright i \text{ is the unique initial state} \\
4 & \quad \text{return } d[f, i] \quad \triangleright f \text{ is the unique final state}
\end{align*}
\]

\[
\text{Relax}(q, s, w, S) \\
\begin{align*}
1 & \quad \text{if } d[q, s] > w \text{ then} \quad \triangleright \text{if } w \text{ is a better estimate of the distance from } q \text{ to } s \\
2 & \quad d[q, s] \leftarrow w \quad \triangleright \text{update } d[q, s] \\
3 & \quad \text{if } q \not\in S \text{ then} \quad \triangleright \text{enqueue } q \text{ in } S \text{ if needed} \\
4 & \quad \text{ENQUEUE}(S, q)
\end{align*}
\]
GetDistance($T, s$)

1. for each $q \in Q$ do
2. \[ d[q, s] \leftarrow \infty \]
3. \[ d[s, s] \leftarrow 0 \]
4. \[ S_s \leftarrow s \]
5. while $S_s \neq \emptyset$ do
6. \[ q \leftarrow \text{HEAD}(S_s) \]
7. \[ \text{DEQUEUE}(S_s) \]
8. for each $e \in E[q]$ do
9. \[ \text{if } i[e] \in \Sigma \cup \{\epsilon\} \text{ then} \quad \triangleright \ i[e] \text{ is a regular symbol} \]
10. \[ \text{RELAX}(n[e], s, d[q, s] + w[e], S_s) \]
11. \[ \text{elseif } i[e] \in \Pi \text{ then} \quad \triangleright \ i[e] \text{ is an close parenthesis} \]
12. \[ B[s, i[e]] \leftarrow B[s, i[e]] \cup \{e\} \]
13. \[ \text{elseif } i[e] \in \Pi \text{ then} \quad \triangleright \ i[e] \text{ is an open parenthesis} \]
14. \[ \text{if } d[n[e], n[e]] = \infty \text{ then} \]
15. \[ \text{GetDistance}(T, n[e]) \]
16. \[ \text{for each } e' \in B[n[e], i[e]] \text{ do} \]
17. \[ \text{RELAX}(n[e'], s, d[q, s] + w[e] + d[p[e'], n[e]] + w[e'], S_s) \]
Shortest Distance Algorithm

• Algorithm can be modified to compute the shortest path through $T$ by keeping track of a parent pointer

• Complexity of the algorithm depends on the *queue discipline*; cubic in $|T|$ if $T$ is acyclic (ignoring stack symbols).
  – the number of non-infinite $d[q, s]$ is $|Q|^2$ in the worst case
  – for each open parenthesis transition $e$, $|B[n[e], i[e]]|$ could be in $O(|E|)$ in the worst case

• When $T$ has been obtained by converting an RTN or an hypergraph into a PDT, the complexity is *linear* ($O(|T|)$)
  – for each $q$, there exists a unique $s$ such that $d[q, s]$ is non-infinity
  – for each close parenthesis transition $e$, there exists a unique open parenthesis transition $e'$ s.t. $e \in B[n[e'], i[e']]$
Application: Parsing

PDTs can be used to parse and different parsing strategies can be obtained from different PDT compilations of a CFG. E.g., the grammar:

\[ S \rightarrow AB, \; S \rightarrow CB, \; C \rightarrow AS, \; A \rightarrow a \; \text{and} \; B \rightarrow b \]

can be parsed with:

\[ \text{ShortestPath}(s \circ T) \] parses input \( s \) (e.g. a string or a lattice) with transducer \( T \).
Application: Machine Translation

Hierarchical machine translation can be expressed as:

\[ \text{ShortestPath}(T \circ M) \]

where \( T = \pi_2(s \circ G) \), \( s \) is the source sentence, \( G \) is a synchronous context-free grammar, and \( M \) is a n-gram language model. The representation of \( T \) determines the translation strategy:

- **T as a CFG/hypergraph**: Chiang [2007]
- **T as finite-state acceptor**: Iglasias, et al.[2009]
- **T as a pushdown acceptor**: Iglasias, et al.[2011]

<table>
<thead>
<tr>
<th>Representation</th>
<th>Time Complexity</th>
<th>Space Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>CFG/hypergraph</td>
<td>( O(</td>
<td>s</td>
</tr>
<tr>
<td>PDA</td>
<td>( O(</td>
<td>s</td>
</tr>
<tr>
<td>FSA</td>
<td>( O(e</td>
<td>s</td>
</tr>
</tbody>
</table>
IV. Current Research and Conclusion
Current Research Topics - Core Algorithms I

- **Disambiguation**
  - Removes redundant successful paths
  - More efficient than determinization for disambiguation
  - Applications: shortest path, $n$-best, compact lattices

![Diagram of automata](image)

**det(T)**

**disamb(T)**
Current Research Topics - Core Algorithms II

- ’Min’ Determinization
  - Determinizes non-functional transducers by retaining only the ’best’ of the ambiguous outputs (over the tropical semiring)
  - Applications: converting search into lookups

![Diagram of a weighted grapheme-to-phoneme transducer]

Stop-Vowel-Stop

Weighted grapheme-to-phoneme transducer has 3.6K transitions.

![Diagram of an unweighted grapheme-to-phoneme transducer]

Unweighted Grapheme-to-Phoneme Transducer
Current Research Topics - Efficiency

- **Dynamic Mutation**
  - Efficiently adding new words and n-grams to optimized, context-dependent recognition transducers
  - Related to *deterministic union* but more complex

- **Linear Automata**
  - Efficient representation of linear models such as CRFs as finite automata
  - Goal: on-the-fly representation using a minimal state space
Current Research Topics - Scalability

- **Large scale - Distributed Algorithms**
  - Represent/combine/optimize automata across thousands of machines
  - Some algorithms admit a map-reduce solution
  - Some algorithms require a more graph-based solution

- **Small Scale - More Compact Representations**
  - Goal: develop a theory of automata entropy
  - Find compression algorithms that meet the entropy bounds
  - Early promising results with a generalization of Lempel-Ziv
Current Research Topics - Generality

• Multi-stack Automata
  – Extend PDTs to allow limited copying
  – Two unrestricted stacks equivalent to a Turing machine
  – Stack restrictions - allow pushing onto (popping from) the second stack only if the first is empty
  – Leads to mildly context-sensitive languages

• Syntax-directed Transductions
  – Pushdown automata can only represent simple syntax-directed trans-lations (no re-ordering of non-terminals)
  – Pushdown processors have proposed to handle the more general case but are complex
  – Applications: hierarchical machine translation
Conclusion

- **Weighted Finite-State Transducers**: Provides a compact representation of rational relations and admits efficient algorithms for their construction, combination, optimization and search.

- **Weighted Pushdown Transducers**: Provides a compact representation of (certain) context-free relations and admits efficient algorithms for their construction, combination, optimization and search, especially when they underlyingly represent rational relations.