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

I. *Historical Remarks*

Weighted Transducers in Speech and Language Processing

Weighted finite-state transducers (WFSTs):

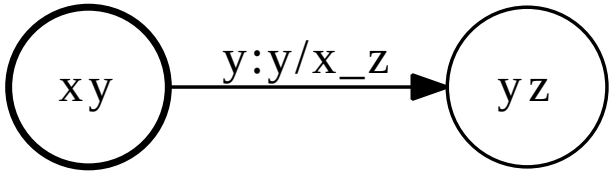
- Core component of many modern ASR systems: used by Google, Nuance, IBM, AT&T among others. Millions of daily users.
- Used as well in speech synthesis, optical character recognition, machine translation among others
- Over 2700 citations for top ten Google Scholar WFST papers
- Thousands of software downloads: (`www.openfst.org`)

Finite-State Automata/Transducers

- Theory:
  - Automata: Huffman, 1954; Moore, 1956; Kleene, 1956; Rabin and Scott, 1959
  - Transducers: Meely, 1955, Moore, 1956; Ginsburg, 1962; Eilenberg, 1967
  - Weighted Automata/Transducers: Schutenzenberger, 1961; Eilenberg, 1974; Salomaa and Soittola, 1978.
- Speech/NLP Applications:
  - Automata: Koskenniemi, 1992; Appelt, 1993; Roche and Schabes, 1995.
  - Transducers: Kaplan and Kay, 1981; Karttunen, Kaplan, and Zaenen; 1992; Kaplan and Kay, 1994; Mohri, 1994.
  - Weighted Transducers: Pereira, Riley, and Sproat, 1994; Mohri, Riley, and Sproat, 1996; Pereira and Riley, 1997; Mohri, Pereira and Riley 1998.

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:



- 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)

WeightedDeterminization(A)

1 $i' \leftarrow \{(i, \lambda(i)) : i \in I\}$
2 $\lambda'(i') \leftarrow \overline{1}$
3 $S \leftarrow \{i'\}$
4 **while** $S \neq \emptyset$ **do**
5 $\quad p' \leftarrow \text{Head}(S)$
6 $\quad \text{Dequeue}(S)$
7 $\quad$ **for** each $x \in i[E[Q[p']]]$ **do**
8 $\quad\quad w' \leftarrow \bigoplus \{v \otimes w : (p, v) \in p', (p, x, w, q) \in E\}$
9 $\quad\quad q' \leftarrow \{(q, \bigoplus \{w'^{-1} \otimes (v \otimes w) : (p, v) \in p', (p, x, w, q) \in E\}) :$
$\quad\quad\quad q = n[e], i[e] = x, e \in E[Q[p']]\}$
10 $\quad\quad E' \leftarrow E' \cup \{(p', x, w', q')\}$
11 $\quad\quad$ **if** $q' \notin Q'$ **then**
12 $\quad\quad\quad Q' \leftarrow Q' \cup \{q'\}$
13 $\quad\quad\quad$ **if** $Q[q'] \cap F \neq \emptyset$ **then**
14 $\quad\quad\quad\quad F' \leftarrow F' \cup \{q'\}$
15 $\quad\quad\quad\quad \rho'(q') \leftarrow \bigoplus \{v \otimes \rho(q) : (q, v) \in q', q \in F\}$
16 $\quad\quad\quad \text{Enqueue}(S, q')$
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.

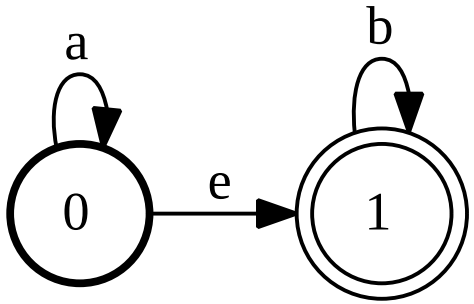
Weighted Transducers in Speech and Language Processing

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`.
- OpenGrm: open-source C++ grammar libraries: `www.opengrm.org`
  - NGram: n-gram language modeling
  - Thrax: finite-state (and beyond) rule compiler

Finite-State Automata

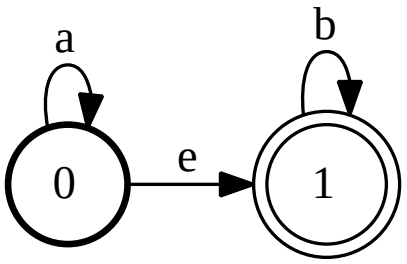


$$L(A) = \{a^n b^m \mid n, m \in \mathbb{N}\}$$

- A transition is labeled with a regular symbol or the empty string, ϵ.
- *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 (Σ, Q, E, I, F) with
  - Σ, 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 $[e = (p[e], i[e], n[e]) \in E]$
- $\pi = e_1 \dots e_k \in E^*$ 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 $i[\pi] = i[e_1]i[e_2] \dots i[e_n]]$
- A string $x \in \Sigma^*$ belongs to $L(A)$, the *language accepted by* A, if there exists a path π such that $p[\pi] \in I$, $n[\pi] \in F$, $i[\pi] = x$.



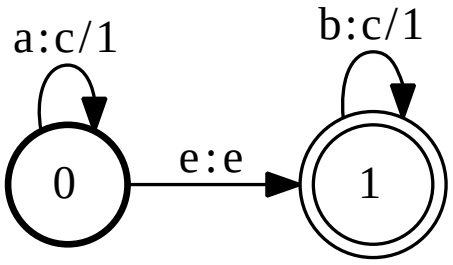
$$L(A) = \{a^n b^m \mid n, m \in \mathbb{N}\}$$

Weighted Finite-State Transducers

- Each transition e in a *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 *weight associated by* T to a pair of strings (x, y) is

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

$$\begin{aligned} P(x, y) &= \{\pi \mid p[\pi] \in I,\ n[\pi] \in F,\ i[\pi] = x,\ o[\pi] = y\}, \\ w[\pi] &= \lambda(\pi) + w[e_1] + w[e_2] \ldots + w[e_n] + \rho(\pi) \end{aligned}$$



$$T(a^n b^m, c^{m+n}) = n + m$$

Semirings

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

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

$$\begin{aligned} P(x,y) &= \{\pi \mid p[\pi] \in I,\ n[\pi] \in F,\ i[\pi] = x,\ o[\pi] = y\}, \\ w[\pi] &= \lambda(\pi) \otimes w[e_1] \otimes w[e_2] \ldots \otimes w[e_n] \otimes \rho(\pi) \end{aligned}$$

A *semiring* $(\mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1})$ = 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).

Semirings

Semiring	Set	$\oplus$	$\otimes$	$\bar{0}$	$\bar{1}$
Boolean	$\{0, 1\}$	$\vee$	$\wedge$	0	1
Probability	$\mathbb{R}_+$	$+$	$\times$	0	1
Log	$\mathbb{R} \cup \{-\infty, +\infty\}$	$\oplus_{\log}$	$+$	$+\infty$	0
Tropical	$\mathbb{R} \cup \{-\infty, +\infty\}$	min	$+$	$+\infty$	0

$\oplus_{\log}$ 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 ϵ-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

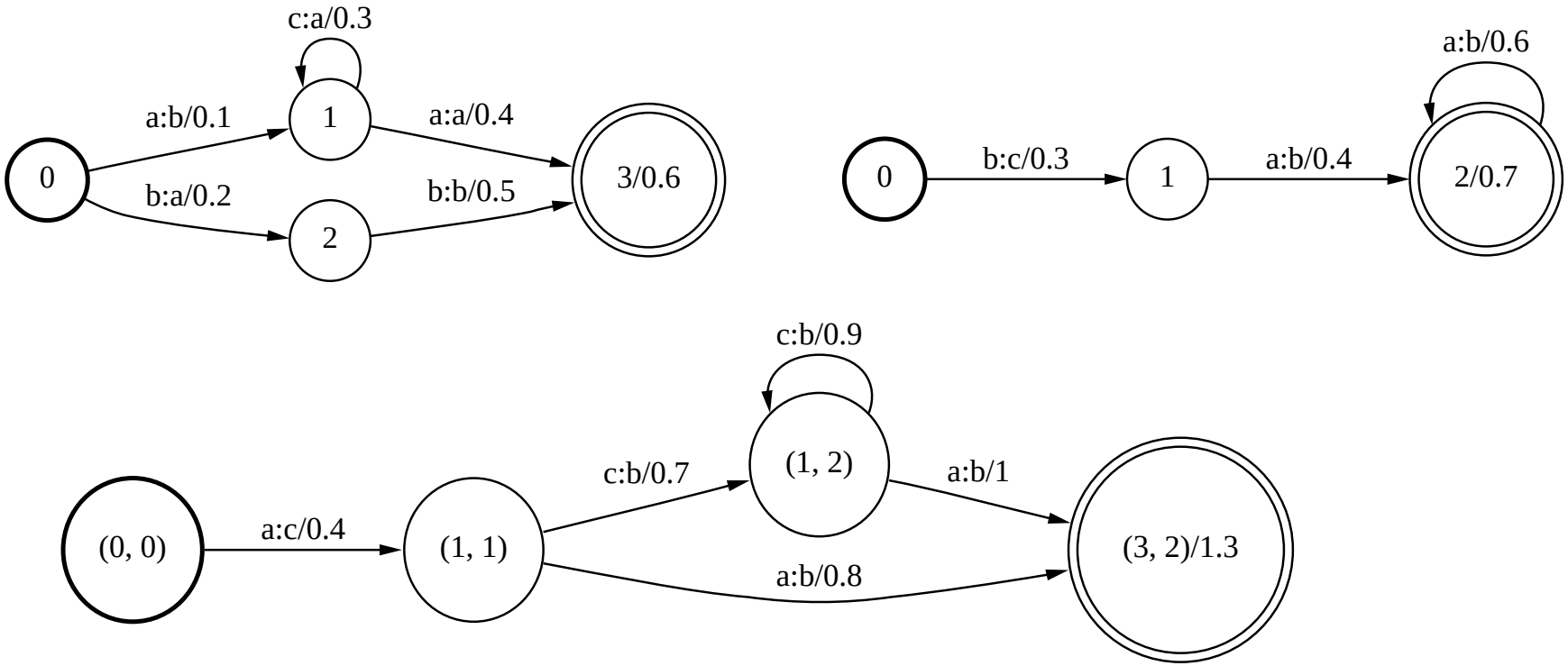
▷ Described in this talk

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

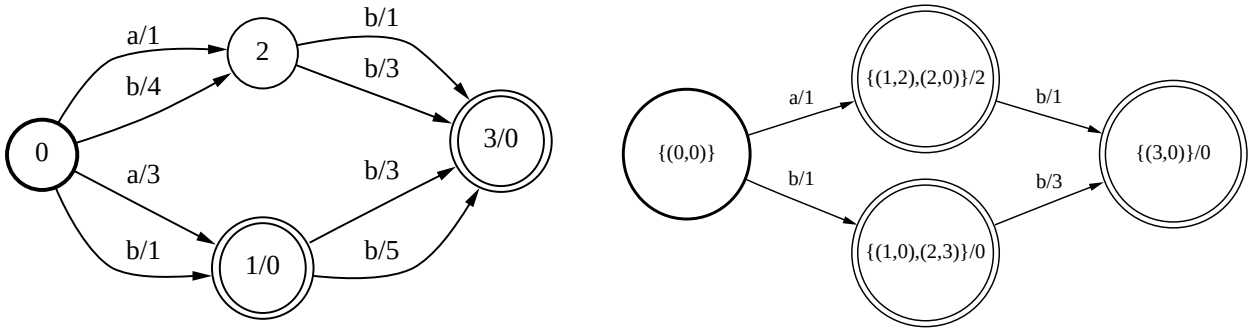
▷ Assuming that T_2 has no input-ϵ transitions

- 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 $(q_2, b, c, w_2, q_2') \rightsquigarrow ((q_1, q_2), a, c, w_1 + w_2, (q_1', q_2'))$
  - Epsilon: $a \in \Sigma \cup \{\epsilon\}$
    $(q_1, a, \epsilon, w_1, q_1')$ and stay in $q_2 \rightsquigarrow ((q_1, q_2), a, \epsilon, w_1, (q_1', q_2))$

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

- Complexity: $O(|T_1||T_2|)$ in the worst case

Determinization Algorithm



- Weighted Determinization: Classical subset construction modified to ensure correct weights
- Transducer Determinization: Application of weighted determinization: output strings in Δ^* treated as weights over the *string semiring* $\mathbb{S}$.
- Weighted Transducer Determinization: Application of weighted determinization: Weights are in $\mathbb{K} \times \mathbb{S}^*$
- Applies only to *determinizable* automata/transducers

Semiring	Set	$\oplus$	$\otimes$	$\overline{0}$	$\overline{1}$
String	$\Delta^* \cup \{\infty\}$	$\wedge$	$\cdot$	∞	ϵ
Product	$\mathbb{K}_1 \times \mathbb{K}_2$	$\oplus_1 \times \oplus_2$	$\otimes_1 \times \otimes_2$	$(\overline{0}, \overline{0})$	$(\overline{1}, \overline{1})$

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

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

ShortestDistance(T)

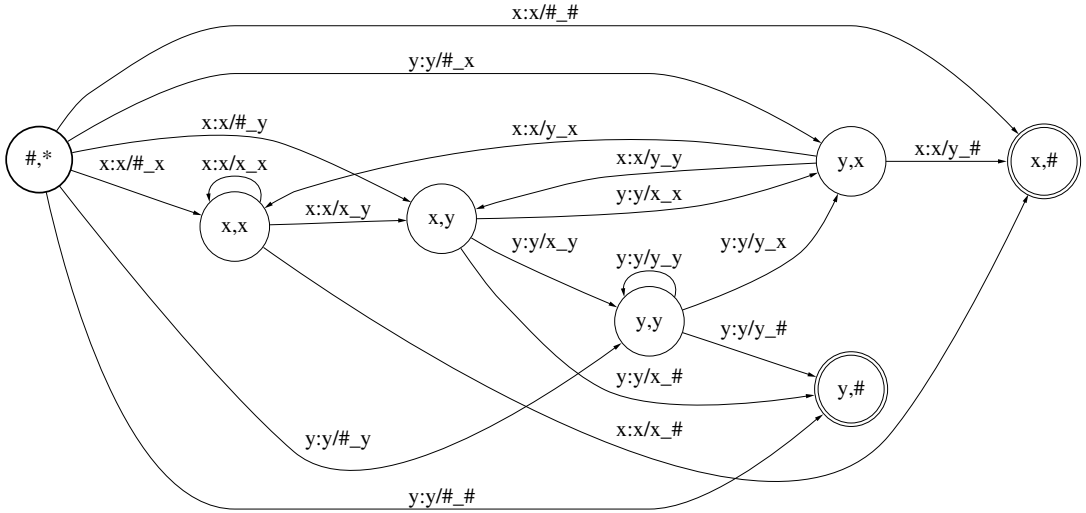
```
1   for each q ∈ Q do
2       d[q] ← ∞
3   d[i] ← 0
4   S ← i
5   while S ≠ ∅ do
6       q ← Head(S)
7       Dequeue(S)
8       for each e ∈ E[q] do
9           Relax(n[e], d[q] + w[e], S)
10  return d[f]                    ▷ f is the unique final state
```

Relax($q, w, \mathcal{S}$)

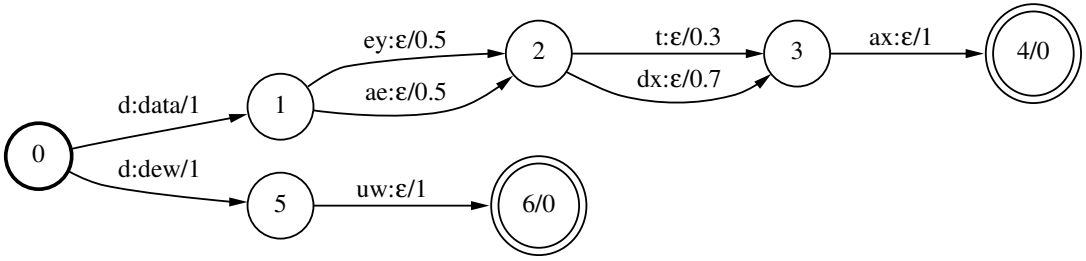
```
1   if d[q] > w then               ▷ if w is a better estimate of the distance from q to i
2       d[q] ← w                   ▷ update d[q]
3       if q ∉ S then              ▷ enqueue q in S if needed
4           Enqueue(S, q)
```

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:

$$Det(C \circ L) \circ G$$

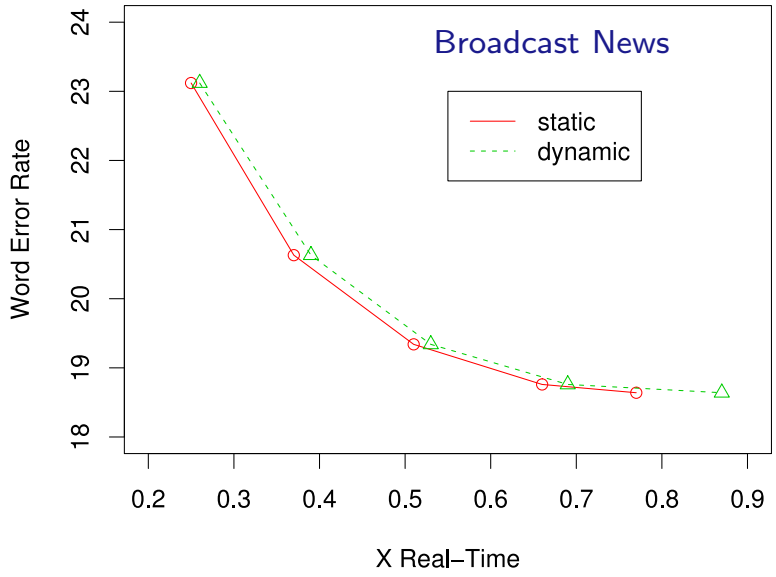
  - Outermost composition with algorithm as described behaves badly
  - Solution: generalized composition with *lookahead filter*: precompute labels reachable from a (lexicon) state.
  - Allows dynamic composition during recognition, trading space for (some) time

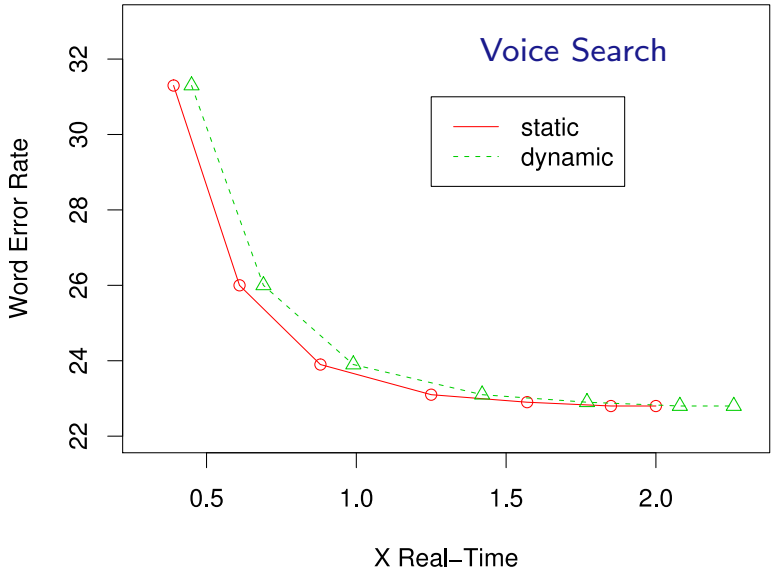
Recognition Experiments

Broadcast News	Voice Search
Acoustic Model	
• Trained on 96 and 97 DARPA Hub4 AM training sets. • PLP cepstra, LDA analysis, STC • Triphonic, 8k tied states, 16 components per state • Speaker adapted (both VTLN + CMLLR)	• Trained on $>$ 1000hrs of voice search queries • PLP cepstra, LDA analysis, STC • Triphonic, 4k tied states, 4 - 128 components pe state • Speaker independent
Language Model	
• 1996 Hub4 CSR LM training sets • 4-gram language model pruned to 8M n-grams	• Trained on $>$ 1B words of google.com and voic search queries • 1 million word vocabulary • Katz back-off model, pruned to various sizes

Recognition Experiments

Precomputation before recognition	Broadcast News			Voice Search		
Construction method	Time	RAM	Result	Time	RAM	Result
Static						
(1) $C \circ Det(L \circ G)$	7 min	5.3G	0.5G	10.5 min	11.2G	1.4G
(2) $Det(C \circ L) \circ G$	2.5 min	2.9G	0.5G	4 min	5.3G	1.4G
Dynamic						
(2) $det(C \circ L) \circ G$	none	none	0.2G	none	none	0.5G





Applications: Various

General form of pipeline:

$$\begin{aligned} Result &= ShortestPath(Input \circ Det(Model)) \\ Input &= string \vee lattice \\ Model &= Model_1 \circ \ldots \circ Model_n \end{aligned}$$

System	Result	Input	Model
Second-pass ASR	one-best	speech lattice	n-gram
Let.-to-sound conversion	graphemes	phonemes	phonotactics $\circ$ pair n-gram
Case restoration	lowercase text	mixed-case text	pair n-gram
Diacritic restoration	ascii text	latin-1 text	pair n-gram
Spelling correction	text	corrected text	confusion model $\circ$ n-gram

Weighted Transducers in Speech and Language Processing

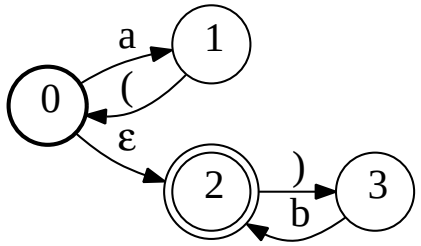
III. *Pushdown Transducers: Algorithms and Applications*

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`.

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:

  $$S \to \epsilon, \; S \to SS \text{ and } S \to aS\bar{a} \text{ for all } a \in A.$$

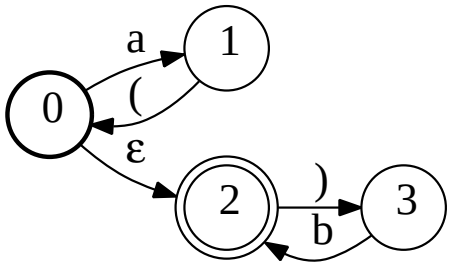
- For $b \in A \cup \overline{A}$, let

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

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

Pushdown Automata

- A *pushdown automaton* A is a 7-uple $(\Sigma, \Pi, \overline{\Pi}, Q, E, I, F)$ with
  - Σ, Π and $\overline{\Pi}$: input, open parenthesis and close parenthesis alphabets
  - Q, $I \subseteq Q$, $F \subseteq Q$: states, initial states and final states
  - $E \subseteq Q \times (\Sigma \cup \Pi \cup \overline{\Pi} \cup \{\epsilon\}) \times Q$: transitions $[e = (p[e], i[e], n[e]) \in E]$
- $\pi = e_1 \dots e_k \in E^*$ 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 $i[\pi] = i[e_1]i[e_2] \dots i[e_n]]$
- A string $x \in \Sigma^*$ belongs to $L(A)$, the *language accepted by* A, if there exists a path π such that $p[\pi] \in I$, $n[\pi] \in F$, $i[\pi]_{|\Sigma} = x$ and $i[\pi]_{|\Pi \cup \overline{\Pi}} \in D_\Pi$



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

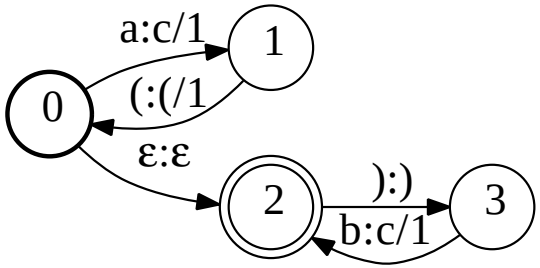
Weighted Pushdown Transducers

- Each transition e in a *weighted pushdown transducer* T has additionally:
  - an output label $o[e] \in \Delta \cup \Pi \cup \overline{\Pi} \cup \{\epsilon\}$ 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

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

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



$$T(a^n b^n, c^{2n}) = 3n$$

Properties

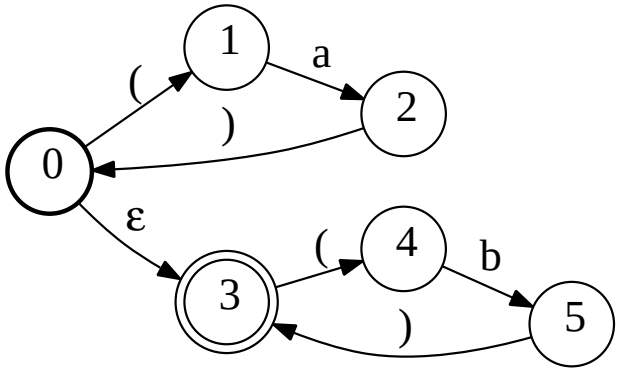
- Recognition power
  - A language is recognizable by a PDA iff it is context-free
  - A transduction is recognizable by a PDT iff it is a simple syntax-directed translation
- Closure properties
  - Closed under sum (union), product (concatenation), kleene-closure and reversal
  - Closed under composition/intersection 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

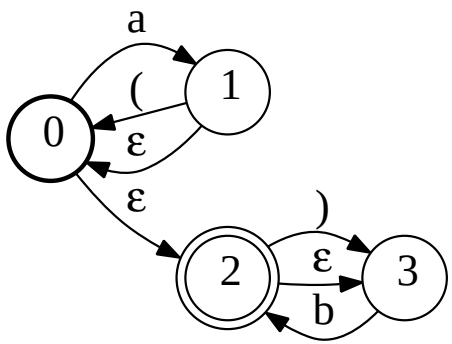
- A pushdown transducer T has *bounded stack* if there exists $K \in \mathbb{N}$ such that for any partially balanced path π from the initial state in T the number of not yet balanced parentheses is less than K:

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

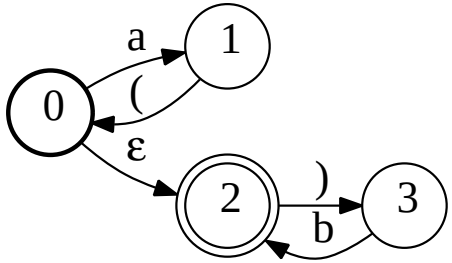
▷ If T has bounded stack, then it represents a rational transduction



bounded-stack
rational
a^*b^*



not bounded-stack
rational
a^*b^*



not bounded-stack
not rational
$\{a^n b^n | n \in \mathbb{N}\}$

WPDT Algorithms

Union	FST alg.
Concatenation/Closure	FST alg.
Reversal	trivial changes to FST alg.
Inversion/Projection	FST alg.
Expansion	PDT-specific alg.$^\diamond$
Replacement	PDT-specific alg.
▷ Composition	non-trivial changes to FST alg.
▷ Determinization	PDT-specifc alg$^\diamond$
Epsilon removal	FST alg.
▷ Shortest distance	PDT-specific alg.$^\diamond$
▷ Shortest path	PDT-specific alg.$^\diamond$
Pruning	PDT-specific alg. required
Connection	PDT-specific alg. required

▷ described in this talk

$^\diamond$Requires bounded-stack input.

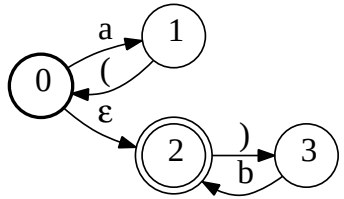
Composition

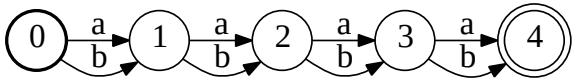
- Definition:

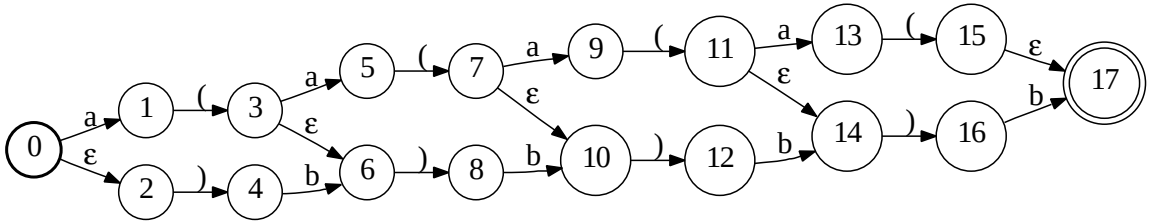
$$(T_1 \circ T_2)(x, y) = \min_{z \in \Sigma^*} (T_1(x, z) + T_2(z, y))$$

▷ 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 ϵ-transitions
    $\rightarrow$ parentheses are treated as different kind of epsilons







Composition Algorithm

▷ Assuming that T_2 has no input-ϵ 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 $(q_2, b, c, w_2, q_2') \rightsquigarrow ((q_1, q_2), a, c, w_1 + w_2, (q_1', q_2'))$
  - Epsilon: $a \in \Sigma \cup \{\epsilon\}$
    $(q_1, a, \epsilon, w_1, q_1')$ and stay in $q_2 \rightsquigarrow ((q_1, q_2), a, \epsilon, w_1, (q_1', q_2))$
  - Parenthesis: $a \in \Pi \cup \overline{\Pi}$
    (q_1, a, a, w_1, q_1') and stay in $q_2 \rightsquigarrow ((q_1, q_2), a, a, w_1, (q_1', q_2))$

▷ 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)

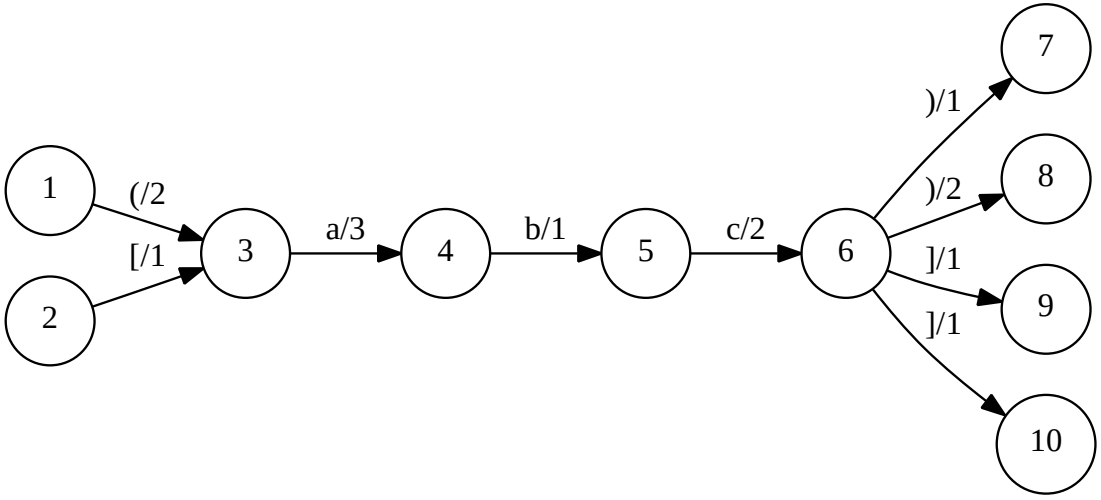
Semiring	Set	$\oplus$	$\otimes$	$\overline{0}$	$\overline{1}$
Parenthesis	$(\Pi \cup \overline{\Pi})^* \cup \{\infty\}$	$\cup$	$\cdot$	∞	ϵ

$a \cdot \overline{a} = \epsilon$ 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



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 $a \in \Pi$

ShortestDistance(T)

```
1  for each q ∈ Q and a ∈ Π do
2        B[q, a] ← ∅
3  GetDistance(T, i)          ▷ i is the unique initial state
4  return d[f, i]             ▷ f is the unique final state
```

Relax($q, s, w, \mathcal{S}$)

```
1  if d[q, s] > w then        ▷ if w is a better estimate of the distance from q to s
2        d[q, s] ← w          ▷ update d[q, s]
3        if q ∉ S then        ▷ enqueue q in S if needed
4              Enqueue(S, q)
```

```
GetDistance(T, s)
 1  for each q ∈ Q do
 2        d[q, s] ← ∞
 3  d[s, s] ← 0
 4  S_s ← s
 5  while S_s ≠ ∅ do
 6        q ← Head(S_s)
 7        Dequeue(S_s)
 8        for each e ∈ E[q] do
 9              if i[e] ∈ Σ ∪ {ε} then                    ▷ i[e] is a regular symbol
10                    Relax(n[e], s, d[q, s] + w[e], S_s)
11              elseif i[e] ∈ Π̄ then                      ▷ i[e] is an close parenthesis
12                    B[s, ī[e]] ← B[s, ī[e]] ∪ {e}
13              elseif i[e] ∈ Π then                      ▷ i[e] is an open parenthesis
14                    if d[n[e], n[e]] = ∞ then
15                          GetDistance(T, n[e])
16                    for each e' ∈ B[n[e], i[e]] do
17                          Relax(n[e'], s, d[q, s] + w[e] + d[p[e'], n[e]] + w[e'], S_s)
```

Shortest Distance Algorithm

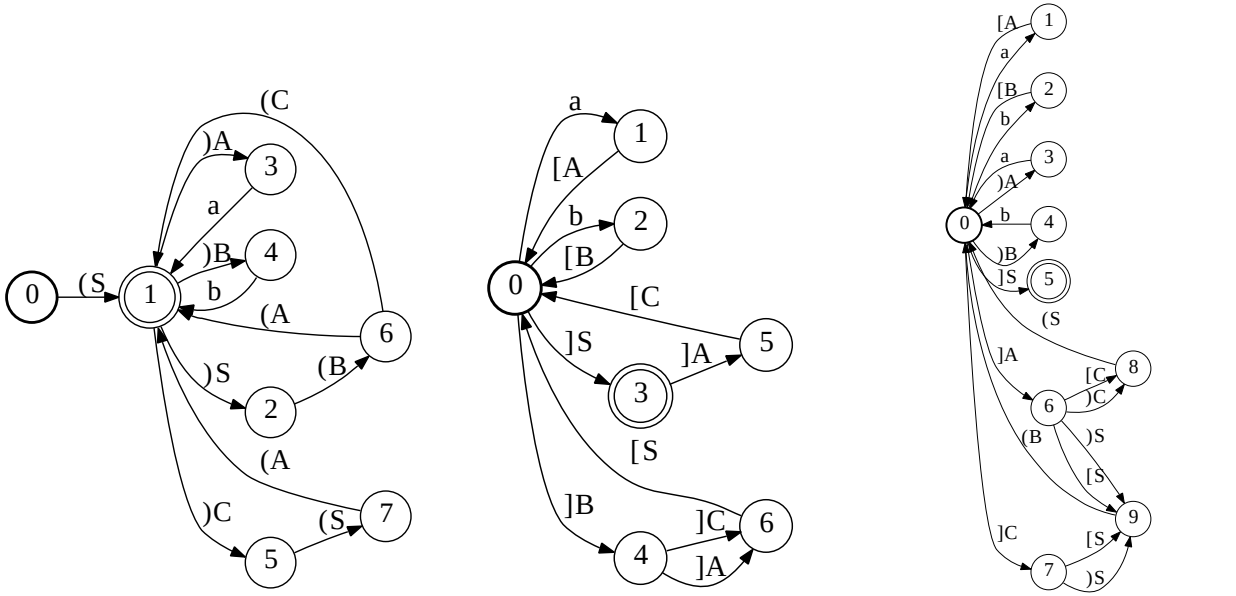
- Algorithm can be modified to compute the shortest path through T by keeping track of a parent pointer
- Complexity of the algorithm depends on the *queue discipline*; cubic in $|T|$ if T is acyclic (ignoring stack symbols).
  - the number of non-infinite $d[q, s]$ is $|Q|^2$ in the worst case
  - for 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:

$$S \to AB,\ S \to CB,\ C \to AS,\ A \to a \text{ and } B \to b$$

can be parsed with:



(a) left parser (b) right parser (c) left corner parser

$ShortestPath(s \circ T)$ parses input s (e.g. a string or a lattice) with transducer T.

Application: Machine Translation

Hierarchical machine translation can be expressed as:

$$ShortestPath(T \circ M)$$

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]

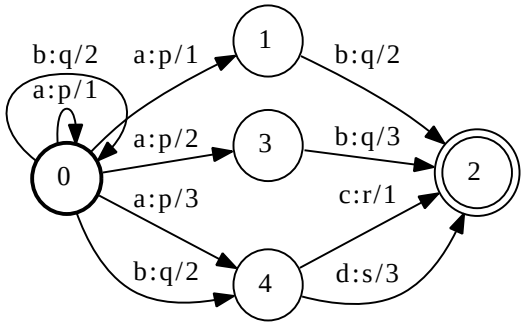
Representation	**Time Complexity**	**Space Complexity**
CFG/hypergraph	$O(\|s\|^3 \, \|G\| \, \|M\|^3)$	$O(\|s\|^3 \, \|G\| \, \|M\|^3)$
PDA	$O(\|s\|^3 \, \|G\| \, \|M\|^3)$	$O(\|s\|^3 \, \|G\| \, \|M\|^2)$
FSA	$O(e^{\|s\|^3 \|G\|} \, \|M\|)$	$O(e^{\|s\|^3 \|G\|} \, \|M\|)$

Weighted Transducers in Speech and Language Processing

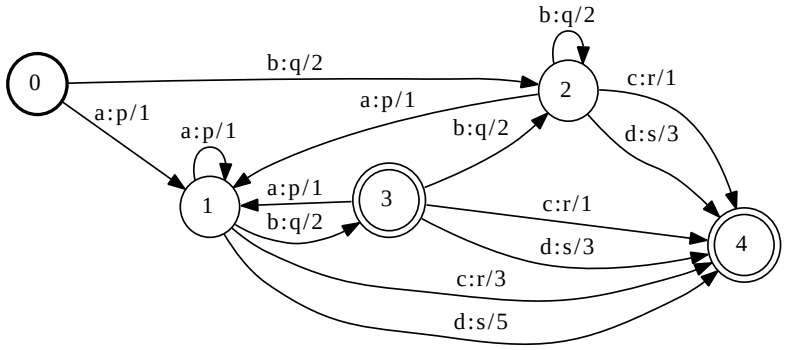
IV. *Current Research and Conclusion*

Current Research Topics - Core Algorithms I

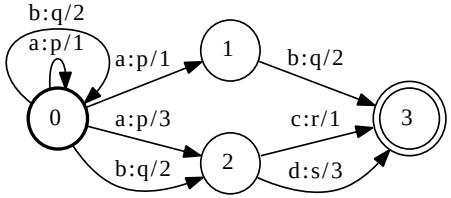
- Disambiguation
  - Removes redundant successful paths
  - More efficient than determinization for disambiguation
  - Applications: shortest path, n-best, compact lattices



T



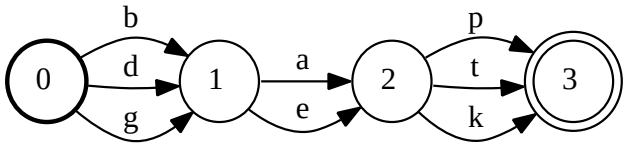
det(T)



disamb(T)

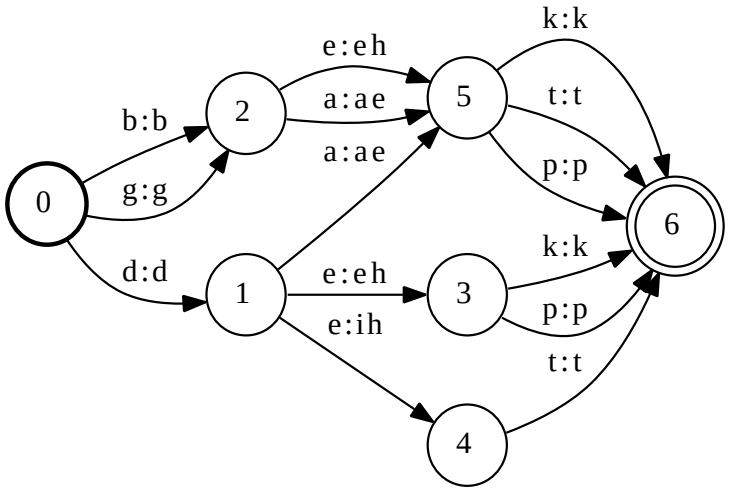
Current Research Topics - Core Algorithms II

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



Stop-Vowel-Stop

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



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 translations (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.