Part I. Mathematical Foundation and Algorithms

Weighted Finite-State Transducers in Speech Recognition

(*) with contributions from Fernando Pereira and Mark-Jan Nederhof.

mohri,m{research.att.com

AT&T Labs - Research

Mehryar Mohri and Michael Riley (*)
3. Applications

- Increased generality: finite-state transducers, multi-phones,
- Handwriting uncertainty: text, handwritten text, speech, image, multimedia.

2. Weights

- General algorithms: rational operations, intersection.
- Efficient algorithms for a variety of problems (e.g., string-matching).

1. Efficiency and Generality of Classical Automata Algorithms

Why Weight Finite-State Transducers?
Software Libraries

Part I. Algorithms

Mohri & Riley

http://research.att.com/sw/tools/gsm/

Grammar and statistical language models.

STN Library: Grammar Library – General software collection for con-

http://research.att.com/sw/tools/fsm/

stucture and modulating weighted automata and transducers representing

transducers.

TRN Library: Finite-State Machine Library – General software utili-

Software Libraries
Repsentations.

**Definition Level**: Specificiation of labels, of costs, and of kinds of FSM

FSMdump("out.fsm", out);

out = FSMinterface("fsm1", "fsm2");

fsm1 = FSMload("fsm1.int", "fsm2");

fsm2 = FSMload("fsm2.int", "fsm3");

fsm3 = FSMload("fsm3.int", "fsm4");

User Program Level: Programs that read from and write to files of

The FSM libraries construct, combine, minimize, and search weighted finite

FSM Library
Binary Format: Compiled representation used by all FSM utilities.

- Symbols Files
- Transducer Files
- Acceptor Files

Textual Format: Used for manual inputting and viewing of FSMs.

FSM File Types
\[
\begin{array}{|c|c|c|c|c|}
\hline
0 & \infty & + & \text{min} & \{\infty, +\} \cap \mathbb{R} \\
\hline
0 & \infty & + \oplus \log & \{\infty, +\} \cap \mathbb{R} & \log \\
\hline
1 & 0 & \times & + & \mathbb{R}^+ \\
\hline
1 & 0 & \vee & \wedge & \{0, 1\} \\
\hline
1 & 0 & \otimes & \oplus & \text{SET} \\
\hline
\end{array}
\]

**Semiring**

**Constituent transitions:**

**Product** \( \cdot \) to compute the weight of a path (product of the weights of paths labeled with that sequence).

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

**Weight Sets:** Semirings

A semiring \( (\mathbb{K}, \otimes, \oplus, 0, 1) \) is a ring that may lack negation.
Symbols File (A.sym):

Acceptor File (A.txt):

Graphical Representation (A.ps):

Automata/Acceptors
Symbols File ($\mathcal{L}$ symbols):

```
\begin{align*}
\text{red} & : \text{yellow}/0.5 \\
\text{green} & : \text{blue}/0.3 \\
\text{blue} & : \text{green}/0 \quad \text{blue} : \text{green}/0.3 \\
\text{yellow} & : \text{red}/0.6 \\
\text{green} & : \text{blue}/0.3 \\
\text{blue} & : \text{yellow}/0.5 \\
\end{align*}
```

Transducer File ($\mathcal{L}$.text):

```
\begin{align*}
2 & : \text{yellow} \quad \text{red}/0.6 \\
1 & : \text{green} \quad \text{blue}/0.3 \\
0 & : \text{red} \quad \text{yellow}/0.5 \\
\end{align*}
```

Graphical Representation ($\mathcal{L}$.ps):

```
\begin{align*}
\text{Graph} & \quad \text{Transducers}
\end{align*}
```
- \( p(R_1, x, y_2) \): paths in \( p(R_1, x, y_2) \) with output label \( y \).
- \( p(R_1, x, R_2) \): paths in \( p(R_1, x, R_2) \) with input label \( x \).
- \( p(R_1, R_2) \): set of all paths from \( R_1 \) to \( R_2 \) to \( R_2 \) to \( R_1 \).

Sets of paths

Definitions and Notation – Paths

\( \text{Path } \not \equiv \)
\[
(\rho, x, I) \cup \\
([\nu]d \otimes [\nu]m \otimes ([\nu]d) \gamma \bigoplus = (\rho, x)[L]
\]

\[\text{Transducer } L \quad \bullet\]

\[
(\rho, x, I) \cup \\
([\nu]d \otimes [\nu]m \otimes ([\nu]d) \gamma \bigoplus = (x)[V]
\]

\[\text{Automaton } A \quad \bullet\]

2. Machines

\[\begin{align*}
\mathbb{A} & \leftarrow A : d \quad \text{final weight function} \\
\mathbb{A} & \leftarrow I : \gamma \quad \text{initial weight function}
\end{align*}\]

\[\begin{align*}
\text{Weight Functions:} & \quad \bullet \\
\mathbb{D} \times \mathbb{A} \times (\{\varepsilon\} \cap \mathbb{V}) \times (\{\varepsilon\} \cap \mathbb{X}) \times \mathbb{D} \supseteq \mathbb{E} \quad \text{Transitions:} \quad \bullet \\
\mathbb{D} \quad \text{States: } \mathbb{D}, \text{ initial states } I, \text{ final states } \mathbb{F} \quad \bullet \\
\mathbb{A} \quad \text{Alphabets: input } \mathbb{V}, \text{ output } \mathbb{X} \quad \bullet
\end{align*}\]

I. General Definitions

Definitions and Notation – Automata and Transducers
Complexity and Implementation

- Lazy implementation

\((|E| + |O|)O \text{ or } ((|O| + |E|) + (|1| + |1|)O)\)

- Complexity (linear): I = I + I = x + x 

as with the tropical semiring (locally closed semirings).

Eq. cycles = 0 (regular transducers), or semiring condition: e.g. weight of

Conditions on the closure operation:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Definition and Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Closure</td>
<td>((h \cdot x)_u[L] \bigoplus \bigotimes \infty = (h \cdot x) [[*_L]])</td>
</tr>
<tr>
<td>Product</td>
<td>((2h \cdot x)[2L] \otimes (1h \cdot 1x)[1L] \bigoplus \bigotimes )</td>
</tr>
<tr>
<td>Sum</td>
<td>((h \cdot x)[2L] \oplus (h \cdot x)[1L] = (h \cdot x)[2L \oplus 1L])</td>
</tr>
</tbody>
</table>
Graphical Representation - FSA

Program - Illustration - Sum
Graphical Representation:

Program:

Closeup - Illustration
### Complexity and Implementation

| Lazy Implementation (see table) | Complexity (linear): $(|\mathcal{A}| + |\mathcal{O}|)O$ |
|---------------------------------|-----------------------------------------------------|

<table>
<thead>
<tr>
<th>Lazy Implementation</th>
<th>Some Elementary Unary Operations - Algorithms</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Yes</strong></td>
<td><strong>Definition and Notation</strong></td>
</tr>
<tr>
<td></td>
<td><em>Projection</em></td>
</tr>
<tr>
<td></td>
<td>$(\tilde{h}^\prime x)[L] \oplus = (x)[\mathcal{V}]$</td>
</tr>
<tr>
<td><strong>Yes</strong></td>
<td><em>Inversion</em></td>
</tr>
<tr>
<td></td>
<td>$(x^\prime \tilde{h})[L] = (\tilde{h}^\prime x)[1-L]$</td>
</tr>
<tr>
<td><strong>No</strong></td>
<td><em>Reversal</em></td>
</tr>
<tr>
<td></td>
<td>$(\tilde{h}^\prime x)[L] = (\tilde{h}^\prime x)[1-L]$</td>
</tr>
</tbody>
</table>
Graphical Representation:

Program: \texttt{fsm.exe} > C.fsm

Reversal - Illustration
Graphical Representation:

Program: fsm2invert A, fsw > C, fsw

Inversion—Illustration
### Complexity and Implementation

<table>
<thead>
<tr>
<th>Deterministic</th>
<th>$A$ weighted $x$</th>
<th>Difference $D$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\forall$</td>
<td>$(x)[\forall \cup 1 \forall] = (x)[\forall \setminus 1 \forall]$</td>
<td>$\forall$</td>
</tr>
<tr>
<td>$K$</td>
<td>$(x)[\forall] \otimes (x)[1 \forall] = (x)[\forall \cup 1 \forall]$</td>
<td>Intersection</td>
</tr>
<tr>
<td>$K$</td>
<td>$(h \cdot x)[\forall L] \otimes (z \cdot x)[1 L] \bigoplus = (h \cdot x)[\forall L \circ 1 L]$</td>
<td>Composition</td>
</tr>
</tbody>
</table>

### Definitions

Some Fundamental Binary Operations - Algorithms
Graphical Representation:

- Program: \texttt{fscompose \texttt{A.fsm} \texttt{B.fsm} \texttt{C.fsm}}
- Composition — Illustration
Replace $T_1 \circ T_2$ by $T_1 \circ F \circ T_2$.

Solution - Filter $F$ for Composition
Graphical Representation:

Program: fsm_intersect A.fsm B.fsm > C.fsm

Illustration - Intersection
Graphical Representation:

Program: difference A.fsm B.fsm > C.fsm

Difference - Illustration
Optimization Algorithms

<table>
<thead>
<tr>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimization</td>
<td>Minimizes using the deterministic algorithm.</td>
</tr>
<tr>
<td>Pushing</td>
<td>E.g., not all weighted automata or transducers can be deterministic.</td>
</tr>
</tbody>
</table>

There are specific semiring conditions for the use of these

- Creates equivalent minimal deterministic machines
- Creates equivalent pushdown/stochastic machines
- Creates equivalent deterministic machines
- Removes ε-transitions
- Removes non-accessible/non-coaccessible states
- Removal
- Connection
- Operation

Optimization Algorithms - Overview
- No natural lazy implementation.

- Complexity (linear): $O(|E| + |V|)$.

Complexity and Implementation

1. Depth-first search of $T_1$ from $I_1$.
2. Mark accessible and coaccessible states.
3. Keep marked states and corresponding transitions.

Description

- Output: weighted transducer $T_2 \equiv T_1$ with all states connected.
- Input: weighted transducer $T_1$.

Definition

Connection - Algorithm
**Graphical Representation:**

Program: fsm connect A, fsm \to C, fsm

Connection - Illustration
\section*{$\epsilon$-Removal – Algorithm}

- **Definition**
  - Input: weighted transducer $T_1$ with $\epsilon$-transitions.
  - Output: weighted transducer $T_2 \equiv T_1$ with no $\epsilon$-transition.

- **Description** (two stages):
  1. **Computation of $\epsilon$-closures**: for any state $p$, states $q$ that can be reached from $p$ via $\epsilon$-paths and the total weight of the $\epsilon$-paths from $p$ to $q$.

     \[ C[p] = \{(q, w) : q \in \epsilon[p], d[p, q] = w \neq 0\} \]

     with:

     \[ d[p, q] = \bigoplus_{\pi \in P(p, \epsilon, q)} w[\pi] \]

  2. **Removal of $\epsilon$’s**: actual removal of $\epsilon$-transitions and addition of new transitions.

    \[ \Rightarrow \text{All-pair K-shortest-distance problem in } T_\epsilon \text{ (} T \text{ reduced to its } \epsilon \text{-transitions)}. \]
Complementation

- Lazy implementation: Integration with on-the-fly weighted determinization
  
  \(|\mathcal{O}| \log \sqrt{\mathcal{O}} + |\mathcal{A}| \sqrt{\mathcal{O}}|O\) *

  \(((\otimes L + \oplus L)|\mathcal{A}| \sqrt{\mathcal{O}} + |\mathcal{O}|)O : \mathcal{L}^2 \mathcal{O} \mathcal{L}

  - Complexity:
    
    - Time complexity (cubic): \(\mathcal{O}\)
    
    - Space complexity (quadratic): \(\mathcal{O}\)

  - See references

  - Within decomposition of \(L\) into strongly connected components

  - Closest semimetrics: Floyd-Warshall or Gaussian-Jordan elimination algorithm

  - Distance algorithm [see references]

  - \(L^2\)-ApproXimation: Generic sparse shortest-path algorithm

  - All-pair shortest-distance algorithm in \(L\)

Complementation and Implementation
Graphical Representation:  

Program: Tηwmpēsition T. Jswm > Tηp. Jswm

- Removal - Illusstration
All acyclic machines are deterministic.

All unweighted automata are deterministic.

$M$ is determinizable if the determinization algorithm applies to $M$.

Semiring weakly left divisible semirings.

\begin{align*}
\text{Conditions} & \quad \text{Descriptions}
\end{align*}

1. Generalization of subset construction

- Same input label.
- Initial state and no two transitions leaving the same state share the
- Output: $M_2 \equiv M_1$, subsequence of deterministic or transducer $M_1$. Has a unique

\begin{align*}
\text{Definition} - \text{Algorithm} & \quad \text{Algorithm} - \text{Definition}
\end{align*}
– Lazy implementation.
– Complexity: exponential.

Characterization based on the Turing property.

– Not all weighted automata or transducers are determinizable.
Graphical Representation: Determinization of Weighted Automata — Illustration
Graphical Representation

Program: FROM TERMINATE LFSM → D LFSM

Determination of Weighted Transducers – Illustration
Zero-sum free semiring or zero-sum free machine.

Weakly divisible semiring.

**Conditions** (automata case)

\( ([\varepsilon]u)p \otimes [\varepsilon]m \) → \([\varepsilon]n \)

2. **Reweight:** For each transition \( e \) such that

\[
(\langle d,b \rangle_p \quad \text{for all} \quad p \in \mathcal{P}) \quad \Longrightarrow \quad [\mu]n \quad \text{for each state} \quad \mu.
\]

1. **Single-source shortest distance computation:** For each state \( \mu \),

**Description (two stages):**

- Transitions = \( \mu \) modulo the string/weight at the initial state.
- Input path = \( \varepsilon \) or such that the \( \oplus \)-sum of the weights of all outgoing
- Output: \( W^\mu \) such that the longest common prefix of all output-

**Pushing**

**Algorithm**
$O(|\mathcal{E}| + \log |\mathcal{O}|)$

- General case (tropical semiring)

$(\otimes L + \oplus L) |\mathcal{E}| + |\mathcal{O}|$ $O$

- Acyclic case (linear)

- Automata case

- Transducer case:

$O(|\mathcal{E}| (1 + |x^{mw} J|))$
Tropical semi-ring

Graphical representation

Program: \texttt{fsmpush -tc A.fsm > P.fsm}

Weight Pushing - Illustration
Graphical Representation

Program: \texttt{push -li \texttt{fsm} > \texttt{fsm}}

Label Pushing – Illustration
Optimization Algorithms

Part I: Algorithms

Mohri & Riley

\[ ((|x_{\text{unw}}| + |\mathcal{D}| \log |\mathcal{E}| + |\mathcal{D}| + S)O \]

- General case:
  - Acyclic case:
    - Transducer case:
      - General case (trivial semiring):
        - Acyclic case (linear):
          - Automata case

\textbf{Complexity}

Use classical unweighted minimization algorithm.

2. \textbf{Automata minimization: encode pairs (label, weight) as labels and}
   decode input automata.

1. \textbf{Canonical representation: use pushing or other algorithm to tran-}
   sitions.

- Output: deterministic \( \mathcal{M}_2 \equiv \mathcal{M}_1 \) with minimal number of states and
- Input: deterministic weighted automaton or transducer \( \mathcal{M}_1 \).

\textbf{Definition}

Minimization Algorithm
Graphical Representation:

Program: FSM

Minimization - Illustration
Optimization Algorithms

Part I Algorithms

Mohan R. Riley

42

- Second stage (quasi-linear): $O(m \cdot m')$

Second stage (quasi-linear):

- First stage: $O(\log (|A^0| + |A^1|) + (|A^2| + |A^1|))$

Complexity

- For testing the equivalence of unweighted automata:

  2. Test: encode partial (label, weight) as labels and use classical algorithm
decide input automata.

- Canonical representation: use pushing or other algorithms to stan-

  Description: two steps

  - Output: true if $A^1 \equiv A^2$, otherwise.

  Definition

  Equivalence - Algorithm
Graphical Representation:

Program: FSM_M

Equivalence - Illustration
On-the-fly weighted determinization.
the potentials.
General N-best paths algorithm augmented with the computation of
\(N\)-best strings algorithm

Bellman-Ford) or topological sort order (Lawler).

BFS and the specific queue disciplines: best-first (Dijkstra's, PIFO
semiring and the specific algorithm in the specific case of the tropical
algorithm)

Graph.

Works with any queue discipline and any semiring if closed for the

\[(a, b)_{p \in \mathcal{P}} \]

\([\mu]n \bigoplus = [b]p

Definition: For each state \(q^*\),

Generic single-source shortest-distance algorithm

Single-source shortest-distance algorithms — Algorithms
Graphical Representation

Program: fswbestpath [-n N] A.fsm > C.fsm

Single-Source Shortest-Distance Algorithms - Illustration
Graphical Representation:

Program: fstream -c1.0 afstream >cfstream

Running - Illustration
5. Expand \( W \) on-the-fly for each input string using lazy replacement and terminals \( A \).

4. Create simple automaton \( W \) accepting exactly the set of active non-terminals \( X \in S \) a weighted automaton \( W(X) \) derived from \( K(S) \).

3. Construct weighted automaton \( K(S) \) for each SCC \( S \) and for each non-

2. Compute the strongly connected components (SCCs) of \( D_G \).

1. Build the dependency graph \( D_G \) of the input grammar \( G \).

**Description**

- Initially recursive nonterminals are either all right-linear or all left-linear.
- Condition: \( G \) must be strongly regular, e.g., rules of each set \( W \) of mu-

- Output: Weighted automaton \( A \) representing \( G \).
- Input: Weighted context-free grammar \( G \).

**Definition**

Compilation of Weighted CFGs – Algorithm
Lazy compilation algorithm: \( O(|L|) \).
- Compilation algorithm applies to \( L \) rather than \( G \).
- Compact intermediate representation of \( G \) by a weighted transducer \( T \).
\( \mathcal{G} \)

Automa
tion

(Weighted Automata \( \mathcal{K}(S) \))

\( \mathcal{G} \)

Graph

Dependancy

Compilation of Weighted CFGs – Illustartion

\( \lambda \leftarrow 4 \cdot \lambda \)

\( \lambda^q \leftarrow 3 \cdot \lambda \)

\( \lambda^p \leftarrow 7 \cdot X \)

\( \lambda X \leftarrow 1 \cdot Z \)
Graphical Representation

Program

Grammar:

\[ c \leftarrow a \cdot A \quad X \leftarrow 3 \cdot A \quad A^0 \leftarrow 2 \cdot X \quad A \cdot X \leftarrow 1 \cdot Z \]
\[ A \rightarrow a_0 B_1 a_1 B_2 a_2 \ldots B_m a_m \]

3. Replace each rule with left-hand side \( A \in \mathcal{M} \):

\[ e \rightarrow A \]

2. For each nonterminal \( A \in \mathcal{M} \), add new non-terminal \( A' \), new rules:

1. Let \( \mathcal{M} \) be a set of mutually recursive non-terminals.

   - Output: \( C', \) strongly regular approximation of \( C \) with \( \mathcal{T}(C') \subseteq \mathcal{T}(C) \).
   - Input: arbitrary weighted context-free grammar \( C \).

   - **Definition**

   Regular Approximation of Weighted CFGs – Algorithm
\[ (0 = m \text{ when } V^0 \Rightarrow V) \]

\[
\begin{align*}
V^0 & \leftarrow \underbrace{B}_m \\
B^{m-1} \leftarrow & \underbrace{B}_{m-1} \\
& \quad \ldots
\end{align*}
\]

\[
\begin{align*}
\exists q & \in ((W - N) \cap \mathcal{Q}) \ni w^0 \cdots w^m \in \mathcal{Q}, \quad W \in \underbrace{B}_{y}, \ldots \underbrace{B}_0 \text{ with } m > 0
\end{align*}
\]
mate grammar.

- Grammar compilation algorithm directly applies to the resulting approximate.
- Complexity of the simple variant of the algorithm (linear): $O(|G|)$.
- Ent.
- Readable and modifiable result, structure of original grammar still appears.
- At most one new non-terminal symbol for any non-terminal symbol of $G$.  

Complexity and Implementation
Context-Free Grammars

Regular Approximation

\[ \text{Program:} \]

<table>
<thead>
<tr>
<th>a</th>
<th>( \rightarrow )</th>
<th>( \top )</th>
<th>( \downarrow )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \downarrow )</td>
<td>( \rightarrow )</td>
<td>( \top )</td>
<td>( \downarrow )</td>
</tr>
<tr>
<td>( \uparrow )</td>
<td>( \rightarrow )</td>
<td>( \top )</td>
<td>( \downarrow )</td>
</tr>
<tr>
<td>( \downarrow )</td>
<td>( \rightarrow )</td>
<td>( \top )</td>
<td>( \downarrow )</td>
</tr>
<tr>
<td>( \downarrow )</td>
<td>( \rightarrow )</td>
<td>( \top )</td>
<td>( \downarrow )</td>
</tr>
<tr>
<td>( \downarrow )</td>
<td>( \rightarrow )</td>
<td>( \top )</td>
<td>( \downarrow )</td>
</tr>
</tbody>
</table>

Generation

Graphical Representation

Regular Approximation of Weighted CFGs - Illustration
Speech Processing Applications

- Speech Recognition (see Part II)
- Speech Synthesis
- Spoken-Dialue Applications

Components (of a complex system):
- Conveient combination and optimizaiton of different information sources
- Algorithms applying to machines of 500M transistors
- Algorithms based on a general algebraic framework (Semantics)

Conclusion