Weighted Finite-State Transducers in Speech Recognition

Part I. Mathematical Foundation and Algorithms

Mehryar Mohri and Michael Riley (*)
AT&T Labs - Research
{mohri, riley}@research.att.com

(*) with contributions from Fernando Pereira and Mark-Jan Nederhof.
Why Weighted Finite-State Transducers?

1. **Efficiency and Generality of Classical Automata Algorithms**
   - Efficient algorithms for a variety of problems (e.g. string-matching, compilers, Unix, design of controllability systems in aircrafts).
   - General algorithms: rational operations, intersection.

2. **Weights**
   - Handling uncertainty: text, handwritten text, speech, image, biological sequences.
   - Increased generality: finite-state transducers, multiplicity.

3. **Applications**
   - Text: pattern-matching, indexation, compression.
   - Speech: Large-vocabulary speech recognition, speech synthesis.
   - Image: image compression, filters.
Software Libraries

- **FSM Library**: Finite-State Machine Library – general software utilities for building, combining, optimizing, and searching weighted automata and transducers.
  

- **GRM Library**: Grammar Library – general software collection for constructing and modifying weighted automata and transducers representing grammars and statistical language models.
  
FSM Library

The FSM utilities construct, combine, minimize, and search *weighted finite-states machines (FSMs)*.

- **User Program Level:** Programs that read from and write to files or pipelines, *fsm(1):*

  \[ \text{fsmintersect in1.fsm in2.fsm >out.fsm} \]

- **C(++) Library Level:** Library archive of C(++) functions that implements the user program level, *fsm(3):*

  \[
  \begin{align*}
  \text{Fsm in1} & = \text{FSMLoad("in1.fsm");} \\
  \text{Fsm in2} & = \text{FSMLoad("in2.fsm");} \\
  \text{Fsm out} & = \text{FSMIntersect(fsm1, fsm2);} \\
  \text{FSMDump("out.fsm", out);} 
  \end{align*}
  \]

- **Definition Level:** Specification of *labels*, of *costs*, and of kinds of FSM representations.
FSM File Types

- **Textual Format:** Used for manual inputting and viewing of FSMs
  - Acceptor Files
  - Transducer Files
  - Symbols Files

- **Binary Format:** ‘Compiled’ representation used by all FSM utilities.
Compiling, Printing, and Drawing FSMs

• Compiling
  fsmcompile -s tropical -iA.sym <A.txt >A.fsm
  fsmcompile -s log -iA.sym -oA.sym -t <T.txt >T.fsm

• Printing
  fsmprint -iA.sym <A.fsm >A.txt
  fsmprint -iA.sym -oA.sym <T.fsm >T.txt

• Drawing
  fsmdraw -iA.sym <A.fsm | dot -Tps >A.ps
  fsmdraw -iA.sym -oA.sym <T.fsm | dot -Tps >T.ps
Weight Sets: Semirings

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

<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>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>

with \(\oplus_{\log}\) defined by: \(x \oplus_{\log} y = -\log(e^{-x} + e^{-y})\).
Automata/Acceptors

• **Graphical Representation** (A.ps):

```
red/0.5
0 -- green/0.3 --> 1
```

```
1 -- blue/0
```

```
2/0.8
yellow/0.6
```

• **Acceptor File** (A.txt):

```
0 0 red .5
0 1 green .3
1 2 blue
1 2 yellow .6
2 .8
```

• **Symbols File** (A.sym):

```
red  1
green 2
blue  3
yellow 4
```
Transducers

• **Graphical Representation** *(T.ps):*

  red:yellow/0.5

  \[ \begin{array}{c}
  0 \quad \text{green:blue/0.3} \quad 1 \\
  \end{array} \]

  blue:green/0

  yellow:red/0.6

  \[ \begin{array}{c}
  2/0.8 \\
  \end{array} \]

• **Transducer File** *(T.txt):*

  0 0 red yellow .5
  0 1 green blue .3
  1 2 blue green
  1 2 yellow red .6
  2 .8

• **Symbols File** *(T.syms):*

  red 1
  green 2
  blue 3
  yellow 4
Definitions and Notation – Paths

- **Path** $\pi$
  - Origin or previous state: $p[\pi]$.
  - Destination or next state: $n[\pi]$.
  - Input label: $i[\pi]$.
  - Output label: $o[\pi]$.

```
\begin{center}
\begin{tikzpicture}
  \node[circle,draw] (p) at (0,0) {$p[\pi]$};
  \node[circle,draw] (n) at (2,0) {$n[\pi]$};
  \draw (p) -- node[below]{$i[\pi]:o[\pi]$} (n);
\end{tikzpicture}
\end{center}
```

- **Sets of paths**
  - $P(R_1, R_2)$: set of all paths from $R_1 \subseteq Q$ to $R_2 \subseteq Q$.
  - $P(R_1, x, R_2)$: paths in $P(R_1, R_2)$ with input label $x$.
  - $P(R_1, x, y, R_2)$: paths in $P(R_1, x, R_2)$ with output label $y$. 
Definitions and Notation – Automata and Transducers

1. General Definitions
   - Alphabets: input $\Sigma$, output $\Delta$.
   - States: $Q$, initial states $I$, final states $F$.
   - Transitions: $E \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times (\Delta \cup \{\epsilon\}) \times \mathbb{K} \times Q$.
   - Weight functions:
     initial weight function $\lambda : I \rightarrow \mathbb{K}$
     final weight function $\rho : F \rightarrow \mathbb{K}$.

2. Machines
   - Automaton $A = (\Sigma, Q, I, F, E, \lambda, \rho)$ with for all $x \in \Sigma^*$:
     $$\llbracket A \rrbracket(x) = \bigoplus_{\pi \in P(I, x, F)} \lambda(p[\pi]) \otimes w[\pi] \otimes \rho(n[\pi])$$
   - Transducer $T = (\Sigma, \Delta, Q, I, F, E, \lambda, \rho)$ with for all $x \in \Sigma^*$, $y \in \Delta^*$:
     $$\llbracket T \rrbracket(x, y) = \bigoplus_{\pi \in P(I, x, y, F)} \lambda(p[\pi]) \otimes w[\pi] \otimes \rho(n[\pi])$$
Rational Operations – Algorithms

• Definitions

<table>
<thead>
<tr>
<th>OPERATION</th>
<th>DEFINITION AND NOTATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sum</td>
<td>([T_1 \oplus T_2](x, y) = [T_1](x, y) \oplus [T_2](x, y))</td>
</tr>
<tr>
<td>Product</td>
<td>([T_1 \otimes T_2](x, y) = \bigoplus_{x=x_1 y_2, y=y_1 y_2} [T_1](x_1, y_1) \otimes [T_2](x_2, y_2))</td>
</tr>
<tr>
<td>Closure</td>
<td>([T^*](x, y) = \bigoplus_{n=0}^{\infty} [T]^n(x, y))</td>
</tr>
</tbody>
</table>

• Conditions on the closure operation: condition on \(T\): e.g. weight of \(\epsilon\)-cycles = 0 (regulated transducers), or semiring condition: e.g. \(\overline{1} \oplus x = \overline{1}\) as with the tropical semiring (locally closed semirings).

• Complexity and implementation
  – Complexity (linear): \(O((|E_1| + |Q_1|) + (|E_2| + |Q_2|))\) or \(O(|Q| + |E|)\).
  – Lazy implementation.
Sum – Illustration

- **Program**: fsmunion A.fsm B.fsm >C.fsm

- **Graphical Representation**:

  ![Diagram of FSMs](image-url)
Product – Illustration

- **Program:**  fsmconcat A.fsm B.fsm >C.fsm
- **Graphical Representation:**

![Diagram](image-url)
Closure – Illustration

- **Program:** `fsmclosure B.fsm > C.fsm`
- **Graphical Representation:**

```
B.fsa

0 -> 1/0 (green/0.4)
0 -> 2/0.3 (blue/1.2)

C.fsa

0 -> 1/0 (green/0.4)
0 -> 2/0.3 (blue/1.2)
0 -> 3/0 (eps/0)
0 -> 3/0 (eps/0.3)
```
Some Elementary Unary Operations – Algorithms

- **Definitions**

<table>
<thead>
<tr>
<th>Operation</th>
<th>Definition and Notation</th>
<th>Lazy Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reversal</td>
<td>$[\tilde{T}](x, y) = [T](\bar{x}, \bar{y})$</td>
<td>No</td>
</tr>
<tr>
<td>Inversion</td>
<td>$[T^{-1}](x, y) = [T](y, x)$</td>
<td>Yes</td>
</tr>
<tr>
<td>Projection</td>
<td>$<a href="x">A</a> = \bigoplus_{y} [T](x, y)$</td>
<td>Yes</td>
</tr>
</tbody>
</table>

- **Complexity and implementation**
  - Complexity (linear): $O(|Q| + |E|)$.
  - Lazy implementation (see table).
Reversal – Illustration

• **Program:**  fsmreverse A.fsm > C.fsm

• **Graphical Representation:**

```
0 -> 1 (red/0.5, green/0.3) -> 2 (blue/0, yellow/0.6) -> 3/0 (green/1.2) -> 4/0.3

0 -> 4 (eps/0, eps/0.3) -> 3 (green/1.2) -> 2 (blue/0, yellow/0.6) -> 1/0 (green/0.3)
```

A.fsa

C.fsa
Inversion – Illustration

- **Program:**  `fsminvert A.fsm >C.fsm`

- **Graphical Representation:**

```
A.fst

\begin{figure}
\centering
\begin{tikzpicture}
  \node[state] (0) {0};
  \node[state] (1) at (1,0) {1};
  \node[state] (2) at (2,0) {2};

  \draw[->] (0) -- node[below] {green:pig/0.3} (1);
  \draw[->] (1) -- node[above] {blue:cat/0} (2);
  \draw[->] (2) -- node[below] {yellow:dog/0.6} (1);
  \draw[->, bend left] (0) to node[above] {red:bird/0.5} (0);
\end{tikzpicture}
\end{figure}
```

```
C.fst

\begin{figure}
\centering
\begin{tikzpicture}
  \node[state] (0) {0};
  \node[state] (1) at (1,0) {1};
  \node[state] (2) at (2,0) {2};

  \draw[->] (0) -- node[below] {pig:green/0.3} (1);
  \draw[->] (1) -- node[above] {cat:blue/0} (2);
  \draw[->] (2) -- node[below] {dog:yellow/0.6} (1);
  \draw[->, bend left] (0) to node[above] {bird:red/0.5} (0);
\end{tikzpicture}
\end{figure}
```
Projection – Illustration

- **Program:**  `fsmproject -1 T.fsm >A.fsm`
- **Graphical Representation:**

```
T.fst

0 --green:pig/0.3--> 1 --blue:cat/0

0 --red/0.5----------> 1 --blue/0

A.fsa

0 --green/0.3---> 1 --blue:cat/0
```

red:bird/0.5

2/0.8
Some Fundamental Binary Operations – Algorithms

- **Definitions**

<table>
<thead>
<tr>
<th>OPERATION</th>
<th>DEFINITION AND NOTATION</th>
<th>CONDITION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Composition</td>
<td>([T_1 \circ T_2](x, y) = \bigoplus_z [T_1](x, z) \otimes [T_2](z, y))</td>
<td>(\mathbb{K}) commutative</td>
</tr>
<tr>
<td>Intersection</td>
<td>(<a href="x">A_1 \cap A_2</a> = <a href="x">A_1</a> \otimes <a href="x">A_2</a>)</td>
<td>(\mathbb{K}) commutative</td>
</tr>
<tr>
<td>Difference</td>
<td>(<a href="x">A_1 - A_2</a> = <a href="x">A_1 \cap \overline{A_2}</a>)</td>
<td>(A_2) unweighted &amp; deterministic</td>
</tr>
</tbody>
</table>

- **Complexity and implementation**
  - Complexity (quadratic): \(O((|E_1| + |Q_1|)(|E_2| + |Q_2|))\).
  - Path multiplicity in presence of \(\epsilon\)-transitions: \(\epsilon\)-filter.
  - Lazy implementation.
Composition – Illustration

- **Program:**  
  \texttt{fsmcompose A.fsm B.fsm >C.fsm}

- **Graphical Representation:**
**Multiplicity & \( \epsilon \)-Transitions – Problem**

Redundant \( \epsilon \)-paths.
Solution – Filter $F$ for Composition

Replace $T_1 \circ T_2$ by $T_1 \circ F \circ T_2$. 
Intersection – Illustration

- **Program:**  `fsmintersect A.fsm B.fsm > C.fsm`
- **Graphical Representation:**

A.fsa

B.fsa

C.fsa
Difference – Illustration

- **Program:** `fsmdifference A.fsm B.fsm >C.fsm`
- **Graphical Representation:**

![Diagram of state machines](image)

Mohri & Riley Part I. Algorithms Fundamental Binary Operations 24
### Optimization Algorithms – Overview

#### Definitions

<table>
<thead>
<tr>
<th><strong>Operation</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Connection</td>
<td>Removes non-accessible/non-coaccessible states</td>
</tr>
<tr>
<td>$\epsilon$-Removal</td>
<td>Removes $\epsilon$-transitions</td>
</tr>
<tr>
<td>Determinization</td>
<td>Creates equivalent deterministic machine</td>
</tr>
<tr>
<td>Pushing</td>
<td>Creates equivalent pushed/stochastic machine</td>
</tr>
<tr>
<td>Minimization</td>
<td>Creates equivalent minimal deterministic machine</td>
</tr>
</tbody>
</table>

#### Conditions: There are specific semiring conditions for the use of these algorithms. E.g. not all weighted automata or transducers can be determinized using the determination algorithm.
Connection – Algorithm

• **Definition**
  – Input: weighted transducer $T_1$.
  – Output: weighted transducer $T_2 \equiv T_1$ with all states connected.

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

• **Complexity and implementation**
  – Complexity (linear): $O(|Q_1| + |E_1|)$.
  – No natural lazy implementation.
Connection – Illustration

• Program:  \texttt{fsmconnect A.fsm >C.fsm}

• Graphical Representation:

\begin{center}
\includegraphics[width=\textwidth]{connection_diagram.png}
\end{center}
\textbf{\(\epsilon\)-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.

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

  – All-pair shortest-distance algorithm in $T_\epsilon$.
    
    * $k$-Closed semirings (for $T_\epsilon$) or approximation: generic sparse shortest-distance algorithm [See references].

    * Closed semirings: Floyd-Warshall or Gauss-Jordan elimination algorithm with decomposition of $T_\epsilon$ into strongly connected components [See references],
      
      space complexity (quadratic): $O(|Q|^2 + |E|)$.

    time complexity (cubic): $O(|Q|^3(T_\oplus + T_\otimes + T_*)).$

  – Complexity:

    * Acyclic $T_\epsilon$: $O(|Q|^2 + |Q||E|(T_\oplus + T_\otimes)).$

    * General case (tropical semiring): $O(|Q||E| + |Q|^2 \log |Q|).$

  – Lazy implementation: integration with on-the-fly weighted determinization.
\( \epsilon \)-Removal – Illustration

- **Program:** \texttt{fsmrmepsilon T.fsm >TP.fsm}
- **Graphical Representation:**

\( \text{T.fsm} \)

\( \text{TP.fsm} \)

\( \epsilon \)-Distances
Determinization – Algorithm

- **Definition**
  - Input: *determinizable* weighted automaton or transducer $M_1$.
  - Output: $M_2 \equiv M_1$ subsequential or deterministic: $M_2$ has a unique initial state and no two transitions leaving the same state share the same input label.

- **Description**
  1. **Generalization of subset construction**: weighted subsets $(q_1, w_1), \ldots, (q_n, w_n)$, $w_i$ remainder weight at state $q_i$.
  2. **Weight of a transition in the result**: $\oplus$-sum of the original transitions pre-$\otimes$-multiplied by remainders.

- **Conditions**
  - Semiring: weakly left divisible semirings.
  - $M$ is determinizable $\equiv$ the determinization algorithm applies to $M$.
  - All unweighted automata are determinizable.
  - All acyclic machines are determinizable.
– Not all weighted automata or transducers are determinizable.
– Characterization based on the *twins property*.

**Complexity and Implementation**
– Complexity: exponential.
– Lazy implementation.
Determinization of Weighted Automata – Illustration

- **Program:** `fsmdeterminize A.fsm > D.fsm`

- **Graphical Representation:**

```
0 -> 1/0 (b/1) -> 2 (a/1, b/4) -> 1/0 (b/1) -> 3/0 (b/3)
0 -> 3/0 (b/3)
2 -> 0 (a/3)
2 -> 1/0 (b/3)
1/0 -> 0 (b/1, b/5)
1/0 -> 3/0 (b/3)
3/0 -> 0 (b/3)
3/0 -> 1/0 (b/3)
```

```
{(0,0)} -> {((1,2),(2,0))/2} (a/1) -> {(1,0),(2,3)}/0 (b/1) -> {(0,0)} (b/1)
{(1,0),(2,3)}/0 -> {((0,0))} (b/3)
{((1,2),(2,0))/2} -> {((0,0))} (b/5)
{((1,0),(2,3))/0} -> {((0,0))} (b/5)
```
Determinization of Weighted Transducers – Illustration

- **Program:**  \texttt{fsmdeterminize T.fsm >D.fsm}
- **Graphical Representation:**

```
\begin{itemize}
  \item \texttt{a:a/0.1}
  \item \texttt{a:b/0.2}
  \item \texttt{a:c/0.5}
  \item \texttt{b:b/0.3}
  \item \texttt{b:eps/0.4}
  \item \texttt{a:eps/0.5}
  \item \texttt{c:eps/0.7}
  \item \texttt{a:b/0.6}
  \item \texttt{a:b/0.1}
  \item \texttt{a:eps/0.1}
  \item \texttt{b:a/0.3}
  \item \texttt{b:eps/0.1}
  \item \texttt{c:c/1.1}
  \item \texttt{a:b/0.5}
\end{itemize}
```
Pushing – Algorithm

• Definition
  – Input: weighted automaton or transducer $M_1$.
  – Output: $M_2 \equiv M_1$ such that the longest common prefix of all outgoing paths $= \epsilon$ or such that the $\oplus$-sum of the weights of all outgoing transitions $= \overline{T}$ modulo the string/weight at the initial state.

• Description (two stages):
  1. Single-source shortest distance computation: for each state $q$,
     \[
     d[q] = \bigoplus_{\pi \in P(q,F)} w[\pi]
     \]

  2. Reweighting: for each transition $e$ such that $d[p[e]] \neq \overline{0}$,
     \[
     w[e] \leftarrow (d[p[e]])^{-1} (w[e] \otimes d[n[e]])
     \]

• Conditions (automata case)
  – Weakly divisible semiring.
  – Zero-sum free semiring or zero-sum free machine.
• **Complexity**
  
  – Automata case
    * Acyclic case (linear): $O(|Q| + |E|(T_\oplus + T_\otimes))$.
    * General case (tropical semiring): $O(|Q| \log |Q| + |E|)$.
  
  – Transducer case: $O((|P_{max}| + 1)|E|)$. 
Weight Pushing – Illustration

- **Program:**  
  ```bash
  fsmpush -ic A.fsm >P.fsm
  ```

- **Graphical Representation:**
  - **Tropical semiring**

```plaintext
0 --a/0--> 1 --e/0--> 3/0
   |   |   |   |
   b/1 | c/4 | d/0 | e/1

0 --c/4--> 2 --e/10--> 3/0
   |   |   |   |
   d/0 | e/1 | f/11 |

0 --b/1--> 1 --f/1 --> 3/0
   |   |   |   |
   c/4 | e/10 | f/11
```

```plaintext
0 --b/1--> 1 --f/1 --> 3/0
   |   |   |   |
   c/4 | e/10 | f/11
```
Log semiring
Label Pushing – Illustration

- **Program:**  
  \texttt{fsmpush -il T.fsm >P.fsm}

- **Graphical Representation:**

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure.png}
\end{figure}
Minimization – Algorithm

• **Definition**
  – Input: deterministic weighted automaton or transducer \( M_1 \).
  – Output: deterministic \( M_2 \equiv M_1 \) with minimal number of states and transitions.

• **Description**: two stages
  1. **Canonical representation**: use pushing or other algorithm to standardize input automata.
  2. **Automata minimization**: encode pairs (label, weight) as labels and use classical unweighted minimization algorithm.

• **Complexity**
  – Automata case
    * Acyclic case (linear): \( O(|Q| + |E|(T_\oplus + T_\otimes)) \).
    * General case (tropical semiring): \( O(|E| \log |Q|) \).
  – Transducer case
    * Acyclic case: \( O(S + |Q| + |E|(|P_{max}| + 1)) \).
    * General case: \( O(S + |Q| + |E|(\log |Q| + |P_{max}|)) \).
Minimization – Illustration

- **Program**: `fsmminimize D.fsm > M.fsm`
- **Graphical Representation:**

```plaintext
Mohri & Riley Part I. Algorithms Optimization Algorithms 41
```
Equivalence – Algorithm

• Definition
  – Input: deterministic weighted automata $A_1$ and $A_2$.
  – Output: TRUE if $A_2 \equiv A_1$, FALSE otherwise.

• Description: two stages
  1. Canonical representation: use pushing or other algorithm to standardize input automata.
  2. Test: encode pairs (label, weight) as labels and use classical algorithm for testing the equivalence of unweighted automata.

• Complexity
  – First stage: $O((|E_1| + |E_2|) + (|Q_1| + |Q_2|) \log(|Q_1| + |Q_2|))$ if using pushing in the tropical semiring.
  – Second stage (quasi-linear): $O(m \alpha(m, n))$ where $m = |E_1| + |E_2|$ and $n = |Q_1| + |Q_2|$, and $\alpha$ is the inverse of Ackermann’s function.
Equivalence – Illustration

- **Program:** `fsmequiv [-v] D.fsm M.fsm`
- **Graphical Representation:**

![Diagram of states and transitions](image)

D.fsa

\[=\]

M.fsa
Single-Source Shortest-Distance Algorithms – Algorithm

- **Generic single-source shortest-distance algorithm**
  - Definition: for each state $q$,
    \[
    d[q] = \bigoplus_{\pi \in P(q, F)} w[\pi]
    \]
  - Works with any queue discipline and any semiring $k$-closed for the graph.
  - Coincides with classical algorithms in the specific case of the tropical semiring and the specific queue disciplines: best-first (Dijkstra), FIFO (Bellman-Ford), or topological sort order (Lawler).

- **$N$-best strings algorithm**
  - General $N$-best paths algorithm augmented with the computation of the potentials.
  - On-the-fly weighted determinization.
Single-Source Shortest-Distance Algorithms – Illustration

- **Program:** fsmbestpath [-n N] A.fsm > C.fsm
- **Graphical Representation:**

```
0 --- green/0.3 --- 1       1 --- red/0.5 --- red/0.5 --- 2
     |                    |                   |
     |  blue/0            |  yellow/0.6       |
     |  green/0.3         |                    |
     |                    |  red/0.5          |
     |  red/0.5           |                    |
     |                    |  green/0.3        |
     |                    |  4/0.8            |
```

A.fsa

```
0 --- green/0.3 --- 1 --- blue/0 --- 2/0.8
```

C.fsa
Pruning – Illustration

- **Program:**  `fsmprune -c1.0 A.fsm >C.fsm`
- **Graphical Representation:**

```
A.fsa

0  green/0.3  1  blue/0  2  yellow/0.6  3  blue/0  4/0.8

C.fsa

0  green/0.3  1  blue/0  2  yellow/0.6  3  blue/0  4/0.8
```
Compilation of Weighted CFGs – Algorithm

• **Definition**
  – Input: weighted context-free grammar $G$.
  – Output: weighted automaton $A$ representing $G$.

• **Condition:** $G$ must be *strongly regular*, e.g. rules of each set $M$ of mutually recursive nonterminals are either all right-linear or all left-linear.

• **Description**
  1. Build the dependency graph $D_G$ of the input grammar $G$.
  2. Compute the strongly connected components (SCCs) of $D_G$.
  3. Construct weighted automaton $K(S)$ for each SCC $S$ and for each non-terminal $X \in S$ a weighted automaton $M(X)$ derived from $K(S)$.
  4. Create simple automaton $M_G$ accepting exactly the set of active non-terminals $A$.
  5. Expand $M_G$ on-the-fly for each input string using lazy replacement and editing.
• **Complexity and Implementation**
  
  – Compact intermediate representation of \( G \) by a weighted transducer \( T \).
  
  – Compilation algorithm applies to \( T \) rather than \( G \).
  
  – Lazy compilation algorithm, complexity (linear): \( O(|T|) \).
### Compilation of Weighted CFGs – Illustration

<table>
<thead>
<tr>
<th>Grammar $G$</th>
<th>Dependency Graph</th>
<th>Weighted Automata $K(S)$</th>
<th>Activation Automaton $M_G$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Z .1 \rightarrow XY$</td>
<td><img src="diagram1" alt="Diagram" /></td>
<td>$K({Z})$: <img src="diagram2" alt="Diagram" /></td>
<td><img src="diagram3" alt="Diagram" /></td>
</tr>
<tr>
<td>$X .2 \rightarrow aY$</td>
<td></td>
<td>$K({X, Y})$: <img src="diagram4" alt="Diagram" /></td>
<td></td>
</tr>
<tr>
<td>$Y .3 \rightarrow bX$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$Y .4 \rightarrow c$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
• **Grammar G:** $Z \rightarrow XY \quad X \rightarrow aY \quad Y \rightarrow bX \quad Y \rightarrow c$

• **Program:**
```
grmread -i lab -w cfg.txt | grmcfcompile -i lab -s Z
```

• **Graphical Representation:**

![Diagram](attachment:image.png)
Regular Approximation of Weighted CFGs – Algorithm

• **Definition**
  - Input: arbitrary weighted context-free grammar $G$.
  - Output: $G'$ strongly regular approximation of $G$ with $L(G) \subseteq L(G')$.

• **Description**
  1. Let $M$ be a set of mutually recursive non-terminals.
  2. For each nonterminal $A \in M$, add new nonterminal $A' \not\in N$, new rule:
     \[ A' \rightarrow \epsilon \]
  3. Replace each rule with left-hand side $A \in M$:
     \[ A \rightarrow \alpha_0 B_1 \alpha_1 B_2 \alpha_2 \cdots B_m \alpha_m \]
with $m \geq 0$, $B_1, \ldots, B_m \in M$, $\alpha_0 \ldots \alpha_m \in (\Sigma \cup (N - M))^*$, by:

\[
\begin{align*}
A & \rightarrow \alpha_0 \ B_1 \\
B'_1 & \rightarrow \alpha_1 \ B_2 \\
B'_2 & \rightarrow \alpha_2 \ B_3 \\
\vdots \\
B'_{m-1} & \rightarrow \alpha_{m-1} \ B_m \\
B'_m & \rightarrow \alpha_m \ A'
\end{align*}
\]

$(A \rightarrow \alpha_0 \ A'$ when $m = 0)$. 

• **Complexity and Implementation**

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

<table>
<thead>
<tr>
<th>Grammar $G$</th>
<th>Regular Approximation</th>
<th>Graphical Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$E \rightarrow E + T$</td>
<td>$E' \rightarrow \epsilon$</td>
<td><img src="image" alt="Graph" /></td>
</tr>
<tr>
<td>$E \rightarrow T$</td>
<td>$T' \rightarrow \epsilon$</td>
<td></td>
</tr>
<tr>
<td>$T \rightarrow T * F$</td>
<td>$F' \rightarrow \epsilon$</td>
<td></td>
</tr>
<tr>
<td>$T \rightarrow F$</td>
<td>$E \rightarrow E$</td>
<td></td>
</tr>
<tr>
<td>$F \rightarrow (E)$</td>
<td>$E' \rightarrow + T$</td>
<td></td>
</tr>
<tr>
<td>$F \rightarrow a$</td>
<td>$T' \rightarrow E'$</td>
<td></td>
</tr>
</tbody>
</table>

- **Program:**

  grmcfapproximate -i lab -o nlab cfg.fsm > ncfg.txt

  grmread -i nlab ncfg.txt | grmcfcompile -i nlab -s E >M
Conclusion

• **Generality and Efficiency**
  – Algorithms based on a general algebraic framework (semirings).
  – Algorithms applying to machines of 500M transitions.
  – Convenient combination and optimization of different information sources (components) of a complex system.

• **Speech Processing Applications**
  – Automatic Speech Recognition (see Part II).
  – Speech Synthesis.
  – Spoken-Dialoɡ Applications.