Simpler & More General Minimization for Weighted Finite-State Automata

Jason Eisner
Johns Hopkins University
May 28, 2003 — HLT-NAACL

First half of talk is setup - reviews past work. Second half gives outline of the new results.

The Minimization Problem

Input: A DFA (deterministic finite-state automaton)
Output: An equiv. DFA with as few states as possible
Complexity: $O(|\text{arcs}| \log |\text{states}|)$ (Hopcroft 1971)

Represents the language \{aab, abb, bab, bbb\}
The Minimization Problem

Input: A DFA (deterministic finite-state automaton)
Output: An equiv. DFA with as few states as possible
Complexity: $O(|arcs| \log |states|)$ (Hopcroft 1971)

Can’t always work backward from final state like this.
A bit more complicated because of cycles.
Don’t worry about it for this talk.

Real-World NLP: Automata With Weights or Outputs

- Want to compute functions on strings: $\Sigma^* \rightarrow K$
  - After all, we’re doing language and speech!
  - Finite-state machines can often do the job
  - Easy to build, easy to combine, run fast
  - Build them with weighted regular expressions
    - To clean up the resulting DFA, minimize it to merge redundant portions
    - This smaller machine is faster to intersect/compose
    - More likely to fit on a hand-held device
    - More likely to fit into cache memory

- How do we minimize such DFAs?
  - Didn’t Mohri already answer this question?
  - Only for special cases of the output set $K$!
  - Is there a general recipe?
  - What new algorithms can we cook with it?
Weight Algebras

- Specify a weight algebra \((K, \otimes)\)
- Define DFAs over \((K, \otimes)\)
- Arcs have weights in set \(K\)
- A path’s weight is also in \(K\);
multiply its arc weights with \(\otimes\)
- Examples:
  - (strings, concatenation)
  - (scores, addition)
  - (probabilities, multiplication)
  - (real weights, multiplication)
  - (objectives & gradients, product-rule multiplication)
  - (bit vectors, conjunction)

- Finite-state computation of functions
- Concatenate strings
- Add scores
- Multiply probabilities

Examples:
- (strings, concatenation)
- (scores, addition)
- (probabilities, multiplication)
- (real weights, multiplication)
- (objectives & gradients, product-rule multiplication)
- (bit vectors, conjunction)

Finitestate computation of functions
Concatenate strings
Add scores
Multiply probabilities

Specify a weight algebra \((K, \otimes)\)
Define DFAs over \((K, \otimes)\)
Arcs have weights in set \(K\)
A path’s weight is also in \(K\);
multiply its arc weights with \(\otimes\)

Q: Semiring is \((K, \oplus, \otimes)\). Why aren’t you talking about \(\oplus\) too?
A: Minimization is about DFAs. At most one path per input.
So no need to \(\oplus\) the weights of multiple accepting paths.

Shifting Outputs Along Paths

- Doesn’t change the function computed:

  \[
  \begin{align*}
  a &= w, \\
  b &= w, \quad c = w
  \end{align*}
  \]

  \[
  \begin{align*}
  d &= z
  \end{align*}
  \]

  \[
  \begin{align*}
  abd &\rightarrow wwx, \\
  acd &\rightarrow wz
  \end{align*}
  \]

  \[
  \begin{align*}
  a &= w, \\
  b &= x, \quad c = z
  \end{align*}
  \]

  \[
  \begin{align*}
  d &= z
  \end{align*}
  \]

  \[
  \begin{align*}
  abd &\rightarrow wwx, \\
  acd &\rightarrow wz
  \end{align*}
  \]

- Doesn’t change the function computed:

  \[
  \begin{align*}
  a &= w, \\
  b &= w, \quad c = w
  \end{align*}
  \]

  \[
  \begin{align*}
  d &= z
  \end{align*}
  \]

  \[
  \begin{align*}
  abd &\rightarrow wwx, \\
  acd &\rightarrow wz
  \end{align*}
  \]

  \[
  \begin{align*}
  a &= w, \\
  b &= w, \quad c = w
  \end{align*}
  \]

  \[
  \begin{align*}
  d &= z
  \end{align*}
  \]

  \[
  \begin{align*}
  abd &\rightarrow wwx, \\
  acd &\rightarrow wz
  \end{align*}
  \]
Shifting Outputs Along Paths

- Doesn’t change the function computed:

\[ \begin{align*}
\text{ab}d & \rightarrow \text{ww}x \\
\text{ac}d & \rightarrow \text{ww}z
\end{align*} \]

Shifting Outputs Along Paths

- Doesn’t change the function computed:

\[ \begin{align*}
\text{ab}d & \rightarrow \text{w}x \\
\text{ac}d & \rightarrow \text{w}z
\end{align*} \]

Shifting Outputs Along Paths

- Doesn’t change the function computed:

\[ \begin{align*}
\text{ab}d & \rightarrow \text{w}x \\
\text{ac}d & \rightarrow \text{w}z
\end{align*} \]

Shifting Outputs Along Paths

- Doesn’t change the function computed:

\[ \begin{align*}
\text{ab}d & \rightarrow \text{w}x \\
\text{ac}d & \rightarrow \text{w}z
\end{align*} \]
Shifting Outputs Along Paths

- State sucks back a prefix from its out-arcs.

\[
\begin{align*}
\text{a:}w & \rightarrow \text{b:}x \\
\text{c:}z & \rightarrow \text{d:}z
\end{align*}
\]

\[
\begin{align*}
\text{abd} & \rightarrow \text{wwx} \\
\text{acd} & \rightarrow \text{wwz} \\
\ldots \text{ebd} & \rightarrow \text{wwx} \\
\ldots \text{ecd} & \rightarrow \text{wwz}
\end{align*}
\]
Shifting Outputs Along Paths

- Here, not all the out-arcs start with $w$
- But all the out-paths start with $w$
- Do pushback at later states first:

Shifting Outputs Along Paths (Mohri)

- Here, not all the out-arcs start with $w$
- But all the out-paths start with $w$
- Do pushback at later states first: now we’re ok!
Shifting Outputs Along Paths (Mohri)

- Actually, push back at all states at once

\[ \lambda(q) \]

Add \( \lambda(q) \) to end of q's in-arcs

Remove \( \lambda(q) \) from start of q's out-arcs

Minimizing Weighted DFAs (Mohri)

Mergeable because they accept the same suffix language: \( \{ab, bb\} \)
Minimizing Weighted DFAs (Mohri)

Still accept same suffix language, but produce different outputs on it

Fix by shifting outputs leftward ...

Now mergeable - they have the same suffix function:
ab → yz or wwy
acd → zzz or wwww

But still no easy way to detect mergeability.
Now mergeable - they have the same suffix function:
ab → yz
acd → zzz

Now mergeable - they have the same suffix function:
ab → yz
acd → zzz

Now mergeable - they have the same suffix function:
ab → yz
acd → zzz

Treat each label "a:yz" as a single atomic symbol

Now mergeable - they have the same suffix function:
ab → yz
acd → zzz

Now mergeable - they have the same suffix function:
ab → yz
acd → zzz

Treat each label "a:yz" as a single atomic symbol

Now mergeable - they have the same suffix function:
ab → yz
acd → zzz

Treat each label "a:yz" as a single atomic symbol
Minimizing Weighted DFAs (Mohri)

Treat each label “$a:yz$” as a single atomic symbol
Use unweighted minimization algorithm!

Now mergeable - they have the same suffix function:
$ab \rightarrow yz$
$acd \rightarrow zzz$

Minimizing Weighted DFAs (Mohri)

Treat each label “$a:yz$” as a single atomic symbol
Use unweighted minimization algorithm!

Now mergeable - they have the same suffix language:
$(a:yz \, b:zzz, b:zzz \, b:zzz)$

Summary of weighted minimization algorithm:
1. Compute $\lambda(q)$ at each state $q$
2. Push each $\lambda(q)$ back through state $q$; this changes arc weights
3. Merge states via unweighted minimization


- Mohri treated two versions of $(K, \otimes)$
- $(K, \otimes) = (\text{strings, concatenation})$
  - $\lambda(q)$ = longest common prefix of all paths from $q$
  - Rather tricky to find
- $(K, \otimes) = (\text{nonnegative reals, addition})$
  - $\lambda(q)$ = minimum weight of any path from $q$
  - Find it by Dijkstra’s shortest-path algorithm


- Mohri treated two versions of $(K, \otimes)$
- $(K, \otimes) = (\text{strings, concatenation})$
  - $\lambda(q)$ = longest common prefix of all paths from $q$
  - Rather tricky to find
- $(K, \otimes) = (\text{nonnegative reals, addition})$
  - $\lambda(q)$ = minimum weight of any path from $q$
  - Find it by Dijkstra’s shortest-path algorithm

- Mohri treated two versions of \((K, \otimes)\)
  - \((K, \otimes) = \text{(strings, concatenation)}\)
    - \(\lambda(q)\) = longest common prefix of all paths from \(q\)
      - Rather tricky to find
  - \((K, \otimes) = \text{(nonnegative reals, addition)}\)
    - \(\lambda(q)\) = minimum weight of any path from \(q\)
      - Find it by Dijkstra’s shortest-path algorithm

---


- Mohri treated two versions of \((K, \otimes)\)
  - \((K, \otimes) = \text{(strings, concatenation)}\)
    - \(\lambda(q)\) = longest common prefix of all paths from \(q\)
      - Rather tricky to find
  - \((K, \otimes) = \text{(nonnegative reals, addition)}\)
    - \(\lambda(q)\) = minimum weight of any path from \(q\)
      - Find it by Dijkstra’s shortest-path algorithm

---

In both cases:
- \(\lambda(q)\) = a “sum” over infinite set of path weights
- Must define this “sum” and an algorithm to compute it
- Doesn’t generalize automatically to other \((K, \otimes)\) ...
Generalizing the Strategy

- What properties must the $\lambda$ function have?
- For all $F: \Sigma^* \rightarrow \mathbb{K}$, $k \in \mathbb{K}$, $a \in \Sigma$:
  - **Shifting**: $\lambda(k \otimes F) = k \otimes \lambda(F)$
  - **Quotient**: $\lambda(F)$ is a left factor of $\lambda(a^{-1}F)$
  - **Final-quotient**: $\lambda(F)$ is a left factor of $F(e)$

Then pushing + merging is guaranteed to minimize the machine.

Generalizing the Strategy

- What properties must the $\lambda$ function have?
- For all $F: \Sigma^* \rightarrow \mathbb{K}$, $k \in \mathbb{K}$, $a \in \Sigma$:
  - **Shifting**: $\lambda(k \otimes F) = k \otimes \lambda(F)$
  - **Quotient**: $\lambda(F)$ is a left factor of $\lambda(a^{-1}F)$

Suffix functions can be written as $xx \otimes F$ and $yy \otimes F$:

- $axxza$: $bxxza$
- $ayyza$: $byyab$

Shifting property says:
When we remove the prefixes $\lambda(xx \otimes F)$ and $\lambda(yy \otimes F)$
we will remove $xx$ and $yy$ respectively.

Generalizing the Strategy

- What properties must the $\lambda$ function have?
- For all $F: \Sigma^* \rightarrow \mathbb{K}$, $k \in \mathbb{K}$, $a \in \Sigma$:
  - **Shifting**: $\lambda(k \otimes F) = k \otimes \lambda(F)$

Suffix functions can be written as $xx \otimes F$ and $yy \otimes F$:

- $axxza$: $bxxza$
- $ayyza$: $byyab$

Shifting property says:
When we remove the prefixes $\lambda(xx \otimes F)$ and $\lambda(yy \otimes F)$
we will remove $xx$ and $yy$ respectively.

Actually, remove $xx \otimes \lambda(F)$ and $yy \otimes \lambda(F)$.

Generalizing the Strategy

- What properties must the $\lambda$ function have?
- For all $F: \Sigma^* \rightarrow \mathbb{K}$, $k \in \mathbb{K}$, $a \in \Sigma$:
  - **Shifting**: $\lambda(k \otimes F) = k \otimes \lambda(F)$
  - **Quotient**: $\lambda(F)$ is a left factor of $\lambda(a^{-1}F)$

Quotient property says that this quotient exists
even if $\lambda(F_q)$ doesn't have a multiplicative inverse.

Then pushing + merging is guaranteed to minimize.
A New Specific Algorithm

- Mohri’s algorithms instantiate this strategy.
  - They use particular definitions of $\lambda$.
    - $\lambda(q)$ = longest common string prefix of all paths from $q$
    - $\lambda(q)$ = minimum numeric weight of all paths from $q$
    - Interpreted as infinite sums over path weights; ignore input symbols
    - Dividing by $\lambda$ makes suffix func canonical: path weights sum to 1
  - Now for a new definition of $\lambda$!
    - $\lambda(q)$ = weight of the shortest path from $q$,
      breaking ties lexicographically by input string
    - Choose just one path, based only on its input symbols;
      computation is simple, well-defined, independent of $(K, \otimes)$
    - Dividing by $\lambda$ makes suffix func canonical: shortest path has weight 1

A New Specific Algorithm

- New definition of $\lambda$:
  - $\lambda(q)$ = weight of the shortest path from $q$,
    breaking ties lexicographically by input string
  - Computation is simple, well-defined, independent of $(K, \otimes)$
  - Breadth-first search back from final states:

A New Specific Algorithm

- New definition of $\lambda$:
  - $\lambda(q)$ = weight of the shortest path from $q$,
    breaking ties lexicographically on input symbols
  - Computation is simple, well-defined, independent of $(K, \otimes)$
  - Breadth-first search back from final states:

Requires Multiplicative Inverses

- Does this definition of $\lambda$ have the necessary properties?
  - $\lambda(q)$ = weight of the shortest path from $q$,
    breaking ties alphabetically on input symbols
  - If we regard $\lambda$ as applying to suffix functions:
    - $\lambda(F) = F(\text{min domain}(F))$ with appropriate defn of “min”
  - Shifting: $\lambda(k \otimes F) = k \otimes \lambda(F)$
    - Trivially true
  - Quotient: $\lambda(F)$ is a left factor of $\lambda(a^{-1}F)$
  - Final-quotient: $\lambda(F)$ is a left factor of $F(\varepsilon)$
    - These are true provided that $(K, \otimes)$ contains multiplicative inverses.
    - I.e., okay if $(K, \otimes)$ is a semigroup; $(K, \oplus, \otimes)$ is a division semiring.

Requires Multiplicative Inverses

- So $(K, \otimes)$ must contain multiplicative inverses (under $\otimes$).
  - Consider $(K, \oplus)$ = (nonnegative reals, addition):
    
\[ \lambda = \varepsilon \]
Requires Multiplicative Inverses

- So \((K, \odot)\) must contain multiplicative inverses (under \(\odot\)).
- Consider \((K, \odot) = (\text{nonnegative reals}, \text{addition})\):

\[
\begin{align*}
\lambda &= 1 \\
b &\odot 0 \\
c &\odot 3
\end{align*}
\]

\[
\lambda = 5
\]

Oops! -3 isn’t a legal weight.

Need to say \((K, \odot) = (\text{reals}, \text{addition})\).

Then subtraction always gives an answer.

Unlike Mohri, we might get negative weights in the output DFA ...

But unlike Mohri, we can handle negative weights in the input DFA (including negative weight cycles!).

How about transducers?

- \((K, \odot) = (\text{strings}, \text{concatenation})\)
- Must add multiplicative inverses, via inverse letters.

\[
\begin{align*}
\lambda &= xy \\
aw &\rightarrow wxy \\
bc &\rightarrow wxy \\
ac &\rightarrow wxz
\end{align*}
\]

Can actually make this work, though \(\odot\) no longer \(O(1)\)

- Still arguably simpler than Mohri
- But this time we’re a bit slower in worst case, not faster as before
- Can eliminate inverse letters after we minimize

Real Benefit – Other Semirings!

- Other \((K, \odot)\) of current interest do have mult inverses ... 
- So we now have an easy minimization algorithm for them.
- No algorithm existed before.
Back to the General Strategy

- What properties must the \( \lambda \) function have?
  - For all \( F: \Sigma^* \rightarrow K \), \( k \in K \), \( a \in \Sigma^* \):
    - Shifting: \( \lambda (k \cdot \otimes F) = k \cdot \lambda (F) \)
    - Quotient: \( \lambda (F) \) is a left factor of \( \lambda (a^{-1}F) \)
    - Final-quotient: \( \lambda (F) \) is a left factor of \( F(\varepsilon) \)
  - New algorithm and Mohri’s algs are special cases

- What if we don’t have mult. inverses?
- Does this strategy work in every \( (K, \otimes) \)?
- Does an appropriate \( \lambda \) always exist?
- No! No strategy always works.
- Minimization isn’t always well-defined!

Minimization Not Unique

- In previously studied cases, all minimum-state machines equivalent to a given DFA were essentially the same.
- But the paper gives several \( (K, \otimes) \) where this is not true!

Minimization Not Unique

- In previously studied cases, all minimum-state machines equivalent to a given DFA were essentially the same.
- But the paper gives several \( (K, \otimes) \) where this is not true!

- Mergeability may not be an equivalence relation on states.
- “Having a common residue” may not be an equivalence relation on suffix functions.
- Has to do with the uniqueness of prime factorization in \( (K, \otimes) \).
- (But had to generalize notion so didn’t assume \( \otimes \) was commutative.)
- Paper gives necessary and sufficient conditions ...

Non-Unique Minimization Is Hard

- Minimum-state automaton isn’t always unique.
- But can we find one that has min # of states?
- No: unfortunately NP-complete.
  - (reduction from Minimum Clique Partition)
- Can we get close to the minimum?
  - No: Min Clique Partition is inapproximable in polytime to within any constant factor (unless P=NP).
  - So we can’t even be sure of getting within a factor of 100 of the smallest possible.

Summary of Results

- Some weight semirings are “bad”:
  - Don’t let us minimize uniquely, efficiently, or approximately (even in (bit vectors, conjunction))
- Characterization of “good” weight semirings
- General minimization strategy for “good” semirings
  - Find a \( \lambda \) ... Mohri’s algorithms are special cases
- Easy minimization algorithm for division semirings
  - For additive weights, simpler & faster than Mohri’s
  - Can apply to transducers, with “inverse letters” trick
  - Applies in the other semirings of present interest
- fancy machine learning; parameter training; optimality theory
New definition of $\lambda$:

- $\lambda(q)$ = weight of the shortest path from $q$, breaking ties alphabetically on input symbols

Ranking of accepting paths by input string:

- $\varepsilon < b < bb < aab < aba < abb$

“genealogical order on strings”
we pick the minimum string accepted from state $q$