Article Submitted to Computer Speech and Language

Weighted Finite-State Transducers in Speech Recognition

Mehryar Mohri[1], Fernando Pereira[2] and Michael Riley[1]

[1]*AT&T Labs – Research*
180 Park Avenue, Florham Park, NJ 07932-0971, USA
[2]*Computer and Information Science Dept., University of Pennsylvania*
558 Moore-GRW, 200 South 33rd Street, Philadelphia, PA 19104 USA

Abstract

We survey the use of weighted finite-state transducers (WFSTs) in speech recognition. We show that WFSTs provide a common and natural representation for HMM models, context-dependency, pronunciation dictionaries, grammars, and alternative recognition outputs. Furthermore, general transducer operations combine these representations flexibly and efficiently. Weighted determinization and minimization algorithms optimize their time and space requirements, and a weight pushing algorithm distributes the weights along the paths of a weighted transducer optimally for speech recognition.

As an example, we describe a North American Business News (NAB) recognition system built using these techniques that combines the HMMs, full cross-word triphones, a lexicon of forty thousand words, and a large trigram grammar into a single weighted transducer that is only somewhat larger than the trigram word grammar and that runs NAB in real-time on a very simple decoder. In another example, we show that the same techniques can be used to optimize lattices for second-pass recognition. In a third example, we show how general automata operations can be used to assemble lattices from different recognizers to improve recognition performance.

1. Introduction

Much of current large-vocabulary speech recognition is based on models such as HMMs, tree lexicons, or n-gram language models that can be represented by *weighted finite-state transducers*. Even when richer models are used, for instance context-free grammars for spoken-dialog applications, they are often restricted, for efficiency reasons, to regular subsets, either by design or by approximation [Pereira and Wright, 1997, Nederhof, 2000, Mohri and Nederhof, 2001].

A finite-state transducer is a finite automaton whose state transitions are labeled with both input and output symbols. Therefore, a path through the transducer encodes a mapping from an input symbol sequence to an output symbol sequence. A *weighted* transducer puts weights on transitions in addition to the input and output symbols. Weights may encode probabilities, durations, penalties, or any other quantity that accumulates along paths to compute the overall weight of mapping an input sequence to an output sequence. Weighted transducers are thus a natural choice to represent the probabilistic finite-state models prevalent in speech processing.

We present a survey of the recent work done on the use of weighted finite-state transducers (WFSTs) in speech recognition [Mohri et al., 2000, Pereira and Riley, 1997, Mohri, 1997, Mohri et al., 1996, Mohri and Riley, 1998, Mohri et al., 1998, Mohri and Riley, 1999]. We show that common methods for combining and optimizing probabilistic models in speech processing can be generalized and efficiently implemented by translation to mathematically well-defined operations on weighted transducers. Furthermore, new optimization opportunities arise from viewing all symbolic levels of ASR modeling as weighted transducers. Thus, weighted finite-state transducers define a common framework with shared algorithms for the representation and use of the models in speech recognition that has important algorithmic and software engineering benefits.

We start by introducing the main definitions and notation for weighted finite-state acceptors and transducers used in this work. We then present introductory speech-related examples and describe the most important weighted transducer operations relevant to speech applications. Finally, we give examples of the application of transducer representations and operations on transducers to large-vocabulary speech recognition, with results that meet certain optimality criteria.

2. Weighted Finite-State Transducer Definitions and Algorithms

The definitions that follow are based on the general algebraic notion of *semiring* [Kuich and Salomaa, 1986]. The semiring abstraction permits the definition of automata representations and algorithms over a broad class of weight sets and algebraic operations.

A semiring K consists of a set $\mathbb{K}$ equipped with an associative and commutative operation $\oplus$ and an associative operation $\otimes$, with identities $\overline{0}$ and $\overline{1}$, respectively, such that $\otimes$ distributes over $\oplus$, and $\overline{0} \otimes a = a \otimes \overline{0} = \overline{0}$. In other words, a semiring is similar to the more familiar ring algebraic structure (such as the ring of polynomials over the reals), except that the additive operation $\oplus$ may not have an inverse. For example, $(\mathbb{N}, +, \cdot, 0, 1)$ is a semiring.

The weights used in speech recognition often represent probabilities; the corresponding semiring is then the *probability semiring* $(\mathbb{R}, +, \cdot, 0, 1)$. For numerical stability, implementations may replace probabilities with $-\log$ probabilities. The appropriate semiring is then the image by $-\log$ of the semiring $(\mathbb{R}, +, \cdot, 0, 1)$

and is called the *log semiring*. When using $-\log$ probabilities with a Viterbi (best path) approximation, the appropriate semiring is the *tropical semiring* $(\mathbb{R}_+ \cup \{\infty\}, \min, +, \infty, 0)$.

In the following definitions, we assume an arbitrary semiring $(\mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1})$. We will give examples with different semirings to illustrate the variety of useful computations that can be carried out in this framework by a judicious choice of semiring.

2.1. Weighted Acceptors

Models such as HMMs used in speech recognition are special cases of *weighted finite-state acceptors* (WFSAs). A WFSA $A = (\Sigma, Q, E, i, F, \lambda, \rho)$ over the semiring $\mathbb{K}$ is given by an alphabet or label set Σ, a finite set of states Q, a finite set of *transitions* $E \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times \mathbb{K} \times Q$, an *initial state* $i \in Q$, a set of *final states* $F \subseteq Q$, an *initial weight* λ and a *final weight function* ρ.

A transition $t = (p[t], \ell[t], w[t], n[t]) \in E$ can be represented by an arc from the *source* or *previous state* $p[t]$ to the *destination* or *next state* $n[t]$, with the *label* $\ell[t]$ and *weight* $w[t]$. In speech recognition, the transition weight $w[t]$ typically represents a probability or a $-\log$ probability.

A path in A is a sequence of consecutive transitions $t_1 \cdots t_n$ with $n[t_i] = p[t_{i+1}]$, $i = 1, \ldots, n-1$. Transitions labeled with the *empty symbol* ϵ consume no input. A *successful path* $\pi = t_1 \cdots t_n$ is a path from the initial state i to a final state $f \in F$. The label of the path π is the result of the concatenation of the labels of its constituent transitions:

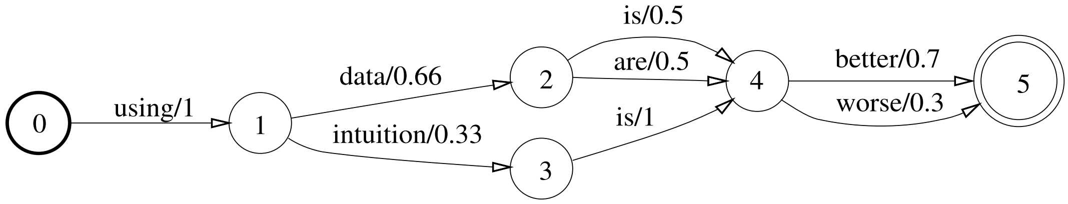
$$\ell[\pi] = \ell[t_1] \cdots \ell[t_n]$$

The weight associated to π is the $\otimes$-product of the initial weight, the weights of its constituent transitions and the final weight $\rho(n[t_n])$ of the state reached by π:

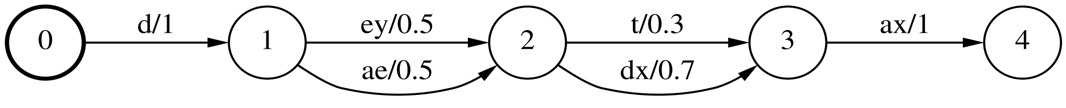
$$w[\pi] = \lambda \otimes w[t_1] \otimes \cdots \otimes w[t_n] \otimes \rho(n[t_n])$$

A symbol sequence x is accepted by A if there exists a successful path π labeled with x: $\ell[\pi] = x$. The weight associated by A to the sequence x is then the $\oplus$-sum of the weights of all the successful paths π labeled with x. Thus, a WFSA provides a mapping from symbol sequences to weights [Salomaa and Soittola, 1978, Berstel and Reutenauer, 1988, Kuich and Salomaa, 1986].

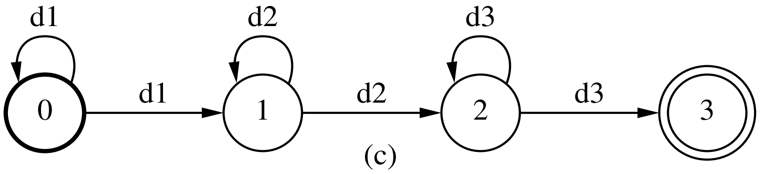
Figure 1 gives some simple, familiar examples of WFSAs as used in speech recognition. The automaton in Figure 1a is a toy finite-state *language model*. The legal word sequences are specified by the words along each complete path, and their probabilities by the product of the corresponding transition probabilities. The transducer in Figure 1b gives the possible pronunciations of one of the word `data` used in the language model. Each legal pronunciation is the sequence of phones along a complete path, and its probability is given by the product of the corresponding transition probabilities. Finally, the transducer in Figure 1c



(a)



(b)



(c)

Figure 1: Weighted finite-state acceptor examples. By convention, the states are represented by circles and marked with their unique number. An initial *state* is represented by a bold circle, final states by double circles. The label and weight of a transition t are marked on the corresponding directed arc by $\ell[t]/w[t]$. The final weight $\rho(f)$ of a final state $f \in F$ is marked by $f/\rho(f)$ or just omitted when $\rho(f) = \overline{1}$ as in these examples. In all the figures in this paper the initial weight is not marked because $\lambda = \overline{1}$.

encodes a typical left-to-right, three distribution-HMM structure for one phone, with the labels along a complete path specifying legal sequences of acoustic distributions for that phone.

2.2. Weighted Transducers

Weighted finite-state transducers (WFSTs) generalize WFSAs by replacing the single transition label by a pair (i, o) of an input label i and an output label o. While a weighted acceptor associates symbol sequences and weights, a WFST associates pairs of symbol sequences and weights, that is, it represents a weighted binary relation between symbol sequences [Salomaa and Soittola, 1978, Berstel and Reutenauer, 1988, Kuich and Salomaa, 1986].*

Formally, a WFST $T = (\Sigma, \Omega, Q, E, i, F, \lambda, \rho)$ over the semiring $\mathbb{K}$ is given by an *input alphabet* Σ, an *output alphabet* Ω, a finite set of states Q, a finite set of *transitions* $E \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times (\Omega \cup \{\epsilon\}) \times \mathbb{K} \times Q$, an *initial state* $i \in Q$, a set of *final states* $F \subseteq Q$, an *initial weight* λ and a *final weight function* ρ.

A transition $t = (p[t], \ell_i[t], \ell_o[t], w[t], n[t])$ can be represented by an arc from the *source state* $p[t]$ to the *destination state* $n[t]$, with the *input label* $\ell_i[t]$, the *output label* $\ell_o[t]$ and the *weight* $w[t]$. The definitions of path, path input label

*In general, several paths may relate a given input sequence to possibly distinct output sequences.

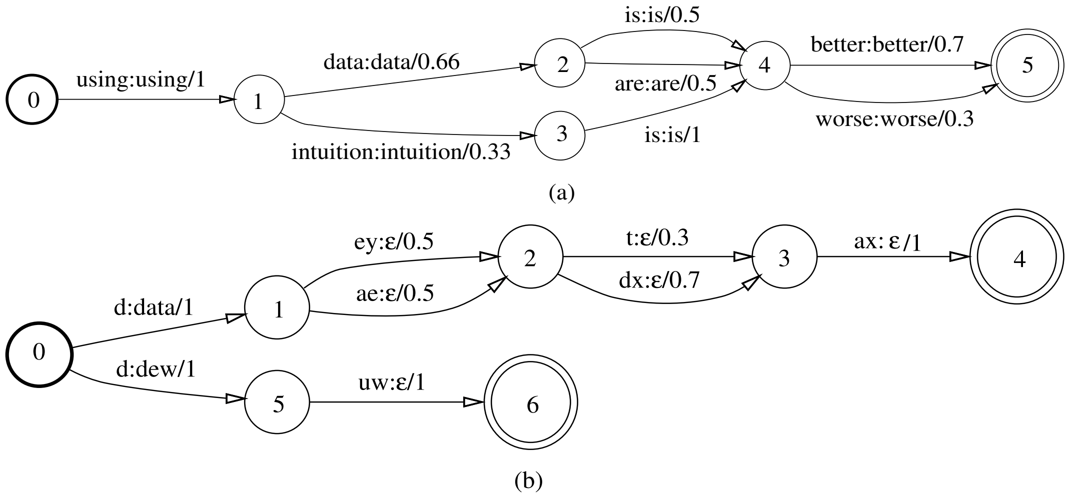


Figure 2: Weighted finite-state transducer examples.

and path weight are those given earlier for acceptors. A path's output label is the concatenation of output labels of its transitions.

The examples in Figure 2 encode (a superset of) the information in the WFSAs of Figure 1a-b as WFSTs. Figure 2a represents the same language model as Figure 1a by giving each transition identical input and output labels. This adds no new information, but is a convenient way of interpreting any acceptor as a transducer that we will use often.

Figure 2b represents a toy pronunciation lexicon as a mapping from phone sequences to words in the lexicon, in this example `data` and `dew`, with probabilities representing the likelihoods of alternative pronunciations. Since a word pronunciation may be a sequence of several phones, the path corresponding to each pronunciation has ϵ-output labels on all but the word-initial transition. This transducer has more information than the WFSA in Figure 1b. Since words are encoded by the output label, it is possible to combine the pronunciation transducers for more than one word without losing word identity. Similarly, HMM structures of the form given in Figure 1c can can be combined into a single transducer that preserves phone model identity while sharing distribution subsequences whenever possible.

2.3. Weighted Transducer Algorithms

Speech recognition architectures commonly give the run-time decoder the task of combining and optimizing transducers such as those in Figure 1. The decoder finds word pronunciations in its lexicon and substitutes them into the grammar. Phonetic tree representations may be used to improve search efficiency at this point [Ortmanns et al., 1996]. The decoder then identifies the correct context-dependent models to use for each phone in context, and finally substitutes them to create an HMM-level transducer. The software that performs these operations is usually tied to particular model topologies. For example, the context-dependent models might have to be triphonic, the grammar might be restricted to trigrams, and the alternative pronunciations might have to be enumerated in the lexicon. Further, these transducer combinations and optimizations are applied in a pre-programmed order to a pre-specified number of levels.

Our approach, in contrast, uses a uniform transducer representation for n-gram grammars, pronunciation dictionaries, context-dependency specifications, HMM topology, word, phone or HMM segmentations, lattices and n-best output lists. We then rely on a common set of weighted transducer operations to combine, optimize, search and prune these automata [Mohri et al., 2000]. Each operation implements a single, well-defined function that has its foundations in the mathematical theory of rational power series [Salomaa and Soittola, 1978, Berstel and Reutenauer, 1988, Kuich and Salomaa, 1986]. Many of those operations are the weighted transducer generalizations of classical algorithms for unweighted acceptors. We have brought together those and a variety of auxiliary operations in a comprehensive weighted finite-state machine software library (FsmLib) available for non-commercial use from the AT&T Labs – Research Web site [Mohri et al., 1997].

Basic *union*, *concatenation*, and *Kleene closure* operations combine transducers in parallel, in series, and with arbitrary repetition, respectively. Other operations convert transducers to acceptors by projecting onto the input or output label set, find the best or the n best paths in a weighted transducer, remove unreachable states and transitions, and sort acyclic automata topologically. We refer the interested reader to the library documentation and an overview paper [Mohri et al., 2000] for further details on those operations. Here, we will focus on a few key operations that support the ASR applications described in later sections.

2.3.1. Composition and Intersection

As previously noted, a transducer represents a binary relation between symbol sequences. The composition of two transducers represents their relational composition. In particular, the composition $T = R \circ S$ of two transducers R and S has exactly one path mapping sequence u to sequence w for each pair of paths, the first in R mapping u to some sequence v and the second in S mapping v to w. The weight of a path in T is the $\otimes$-product of the weights of the corresponding paths in R and S [Salomaa and Soittola, 1978, Berstel and Reutenauer, 1988, Kuich and Salomaa, 1986].

Composition is the transducer operation for combining different levels of representation. For instance, a pronunciation lexicon can be composed with a word-level grammar to produce a phone-to-word transducer whose word sequences are restricted to the grammar. A variety of ASR transducer combination techniques, both context-independent and context-dependent, are conveniently and efficiently implemented with composition.

Our composition algorithm generalizes the classical state-pair construction for

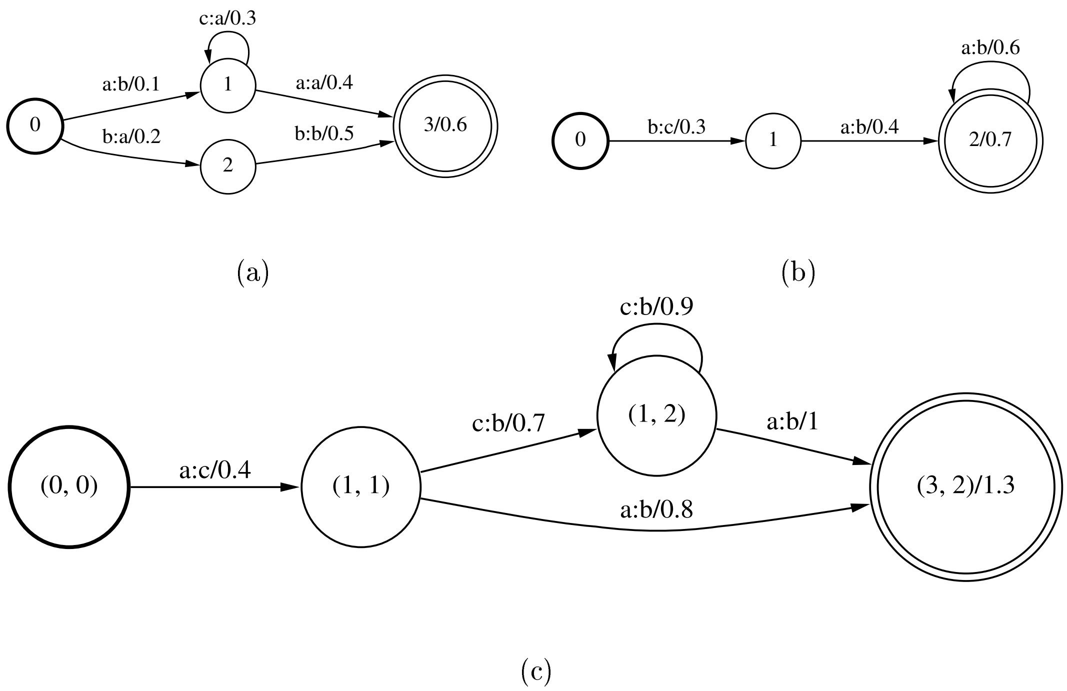


Figure 3: Example of transducer composition.

finite automata intersection [Hopcroft and Ullman, 1979] to weighted acceptors and transducers. The composition $R \circ S$ of transducers R and S has pairs of an R state and an S state as states, and satisfies the following conditions: (1) its initial state is the pair of the initial states of R and S; (2) its final states are pairs of a final state of R and a final state of S, and (3) there is a transition t from (r, s) to (r', s') for each pair of transitions t_R from r to r' and t_S from s to s' such that the output label of t matches the input label of t'. The transition t takes its input label from t_R, its output label from t_S, and its weight is the $\otimes$-product of the weights of t_R and t_S when the weights correspond to probabilities. Since this computation is *local* — it involves only the transitions leaving two states being paired — it can be given a *lazy* (or *on-demand*) implementation in which the composition is generated only as needed by other operations on the composed automaton. Transitions with ϵ labels in R or S must be treated specially as discussed elsewhere [Mohri et al., 1996, 2000].

Figure 3 shows two simple transducers over the tropical semiring, Figure 3a and Figure 3b, and the result of their composition, Figure 3c. The weight of a path in the resulting transducer is the sum of the weights of the matching paths in R and S since in this semiring $\otimes$ is defined as the usual addition (of log probabilities).

Since we represent weighted acceptors by weighted transducers in which the input and output labels of each transition are identical, the intersection of two weighted acceptors is just the composition of the corresponding transducers.

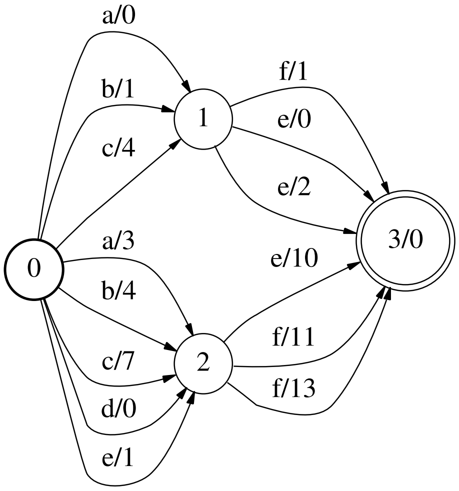


Figure 4: Non-deterministic weighted acceptor A_1.

2.3.2. Determinization

A weighted transducer is *deterministic* or *sequential* if and only if each of its states has at most one transition with any given input label and there are no input epsilon labels. Figure 4 gives an example of a non-deterministic weighted acceptor: at state 0, for instance, there are several transitions with the same label a.

Weighted determinization, which generalizes the classical subset method for determinizing finite automata [Aho et al., 1986], applies to a weighted automaton and outputs an equivalent deterministic weighted automaton. Two weighted acceptors are *equivalent* if they associate the same weight to each input string; weights may be distributed differently along the paths of two equivalent acceptors. Two weighted transducers are equivalent if they associate the same output sequence and weights to each input sequence; the distribution of the weight or output labels along paths needn't be the same in the two transducers.

In contrast to the unweighted case, not all weighted automata can be determinized, as explained rigorously elsewhere [Mohri, 1997]. Fortunately, most weighted automata used in speech processing can be either determinized directly or easily made determinizable by simple transformations, as we shall discuss later. In particular, any acyclic weighted automaton can be determinized.

Our discussion and examples of determinization and, later, minimization will be illustrated with weighted acceptors. The more general weighted transducer case can be shown to be equivalent to this case by interpreting weight-output label pairs as new 'weights' combined by the appropriate semiring [Mohri, 1997]. Determinization and minimization of finite-state transducers can also be used to give an efficient and compact representation of a lexicon [Mohri, 1996].

The critical advantage of a deterministic automaton over equivalent nondeterministic ones is its irredundancy: it contains at most one path matching any

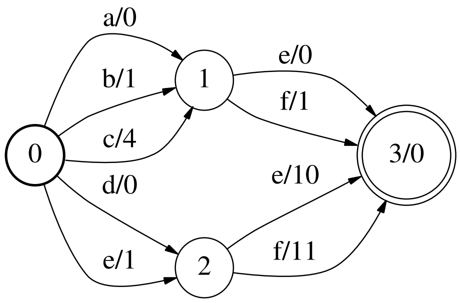


Figure 5: Equivalent weighted automaton A_2 obtained by weighted determinization of A_1.

given input sequence, thereby reducing the time and space needed to process an input sequence.

To eliminate redundant paths, weighted determinization needs to calculate the combined weight of all the paths for a given input sequence, which will depend on the semiring used. We describe determinization in the case of the tropical semiring; this account carries over easily to other semirings.

Figure 5 shows the weighted determinization in the tropical semiring of automaton A_1 from Figure 4. In general, the determinization of a weighted automaton is equivalent to the original, that is, it associates the same weight to each input string. For example, there are two paths corresponding to the input string ae in A_1, with weights $\{0 + 0 = 0, 3 + 10 = 13\}$. The minimum 0 is also the weight associated by A_2 to the string ae.

In the classical subset construction for determinizing unweighted automata, all the states reachable by a given input from the initial state are placed in the same subset. In the weighted case, transitions with the same input label can have different weights, but only the minimum of those weights is needed and the leftover weights must be kept track of. Thus, the subsets in weighted determinization contain pairs (q, w) of a state q of the original automaton and a leftover weight w.

The initial subset is $\{(i, 0)\}$, where i is the initial state of the original automaton. For example, for automaton A_1 the initial subset is $\{(0, 0)\}$. Each new subset S is processed in turn. For each element a of the input alphabet Σ labeling at least one transition leaving a state of S, a new transition t leaving S is constructed in the result automaton. The input label of t is a and its weight is the minimum of the sums $w + l$ where w is s's leftover weight and l is the weight of an a-transition leaving a state s in S. The destination state of t is the subset S' containing those pairs (q', w') in which q' is a state reached by a transition labeled with a from a state of S and w' is the appropriate leftover weight.

For example, state 0 in A_2 corresponds to the initial subset $\{(0, 0)\}$ constructed by the algorithm. The A_2 transition leaving 0 and labeled with a is obtained from the two transitions labeled with a leaving the state 0 in A_1: its weight is

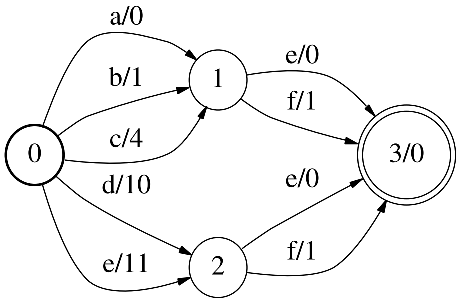


Figure 6: Equivalent weighted automaton B_2 obtained by *weight pushing* from A_2.

the minimum of the weight of those two transitions, and its destination state is the subset $S' = \{(1, 0-0=0), (2, 3-0=3)\}$, numbered 1 in A_2.

It is clear that the transitions leaving a given state in the determinization of an automaton can be computed from the subset for the state and the transitions leaving the states in the subset, as is the case for the classical non-deterministic finite automata (NFA) determinization algorithm. In other words, the weighted determinization algorithm is local like the composition algorithm, and can thus be given a lazy implementation.

2.3.3. Minimization

Any deterministic automaton can be minimized using classical algorithms [Aho et al., 1974, Revuz, 1992]. In the same way, any deterministic weighted automaton A can be minimized using our minimization algorithm [Mohri, 1997].

The resulting weighted automaton B is equivalent to the automaton A, and has the least number of states and the least number of transitions among all deterministic weighted automata equivalent to A.

Weighted minimization is quite efficient, as efficient in fact as classical deterministic finite automata (DFA) minimization: linear in the acyclic case ($O(m+n)$), and $O(m \log n)$ in the general case, where n is the number of states and m the number of transitions

We can view the deterministic weighted automaton A_2 as an unweighted automaton by interpreting each pair (a, w) of a label a and a weight w as a single label. We can then apply the standard DFA minimization algorithm to this automaton. But, since the pairs for different transitions are all distinct, classical minimization would have no effect on A_2.

The size of A_2 can still be reduced by using true weighted minimization. This algorithm works in two steps: the first steps *pushes* weight among transitions, † and the second applies the classical minimization algorithm to the result with each distinct label-weight pair viewed as a distinct symbol, as described above.

Pushing is a special case of *reweighting*. We describe reweighting in the case

†The weight pushing algorithm is described and analyzed in detail in [Mohri, 1998] and its applications to speech recognition are discussed in [Mohri and Riley, 2001a].

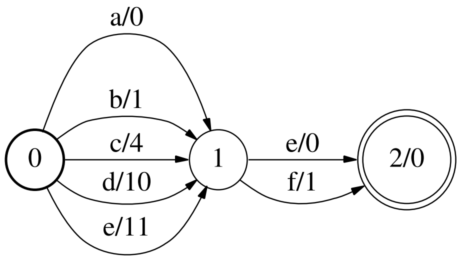


Figure 7: Equivalent weighted automaton A_3 obtained by weighted minimization from A_2.

of the tropical semiring; similar definitions can be given for other semirings. A (non-trivial) weighted automaton can be reweighted in an infinite number of ways that produce equivalent automata. To see how, assume for convenience that the automaton A has a single final state f_A.[‡] Let $V : Q \to \mathbb{R}$ be an arbitrary *potential* function on states. Update each transition weight as follows:

$$w[t] \leftarrow w[t] + (V(n[t]) - V(p[t]))$$

and each final weight as follows:

$$\rho(f_A) \leftarrow \rho(f_A) + (V(i_A) - V(f_A))$$

It is easy to see that with this reweighting, each potential internal to any successful path from the initial state to the final state is added and then subtracted, making the overall change in path weight:

$$(V(f_A) - V(i_A)) + (V(i_A) - V(f_A)) = 0$$

Thus, reweighting does not affect the total weight of a successful path and the resulting automaton is equivalent to the original.

To push the weight in A towards the initial state as much as possible, a specific potential function is chosen, the one that assigns to each state the lowest path weight from that state to the final state. After pushing, the lowest cost path (excluding the final weight) from every state to the final state will thus be 0.

Figure 6 shows the result of pushing for the input A_2. Thanks to pushing, the size of the automaton can then be reduced using classical minimization. Figure 7 illustrates the result of the final step of the algorithm. No approximation or heuristic is used: the resulting automaton A_3 is equivalent to A_2.

3. Weighted Finite-State Transducer Applications

We now describe several applications of weighted finite-state transducer algorithms to speech recognition.

[‡]Any automaton can be transformed into an equivalent automaton with a single final state by adding a super-final state, making all previously final states non-final, and adding an ϵ transition from each of the previously final states f to the super-final state with weight $\rho(f)$.

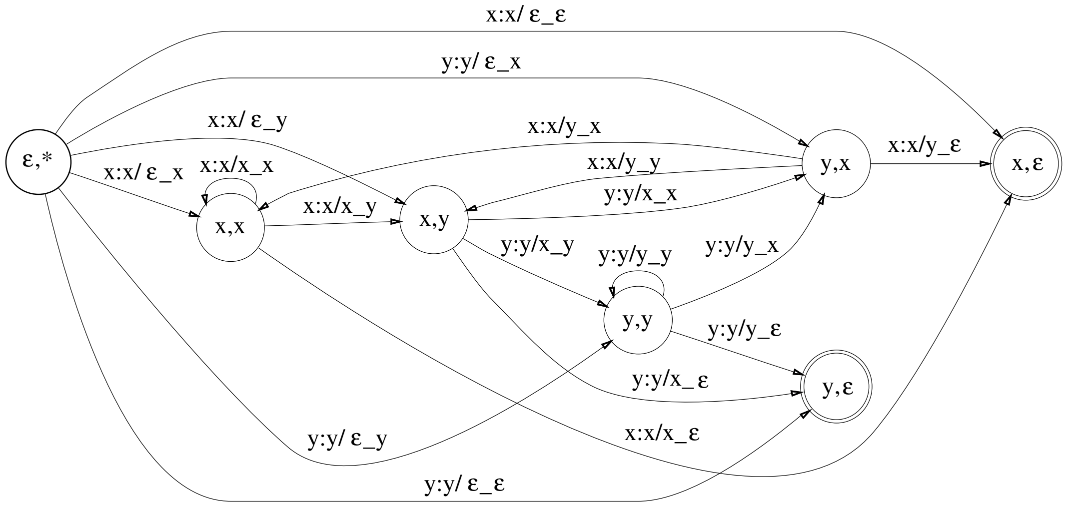


Figure 8: Context-dependent triphone transducer.

3.1. Transducer Combination

Consider the pronunciation lexicon in Figure 2b. Suppose we form the union of this transducer with the pronunciation transducers for the remaining words in the grammar G of Figure 2a and then take its Kleene closure by connecting an ϵ-transition from each final state to the initial state. The resulting pronunciation lexicon L would pair any sequence of words from that vocabulary to their corresponding pronunciations. Thus,

$$L \circ G$$

gives a transducer that maps from phones to word sequences restricted to G.

We used composition here to implement a context-independent substitution. However, a major advantage of transducers in speech recognition is that they generalize naturally the notion of context-independent substitution of a label to the context-dependent case. The transducer of Figure 8 does not correspond to a simple substitution, since it describes the mapping from context-independent phones to context-dependent triphonic models, denoted by *phone/left context_right context*. Just two hypothetical phones x and y are shown for simplicity. Each state encodes the knowledge of the previous and next phones. State labels in the figure are pairs (a, b) of the past a and the future b, with ϵ representing the start or end of a phone sequence and $*$ an unspecified future. For instance, it is easy to see that the phone sequence xyx is mapped by the transducer to $x/\epsilon\_y\ y/x\_x\ x/y\_\epsilon$ via the unique state sequence $(\epsilon, *)(x, y)(y, x)(x, \epsilon)$. More generally, when there are n context-independent phones, this triphonic construction gives a transducer with $O(n^2)$ states and $O(n^3)$ transitions. A tetraphonic construction would give a transducer with $O(n^3)$ states and $O(n^4)$ transitions. In real applications, context-dependency transducers will benefit significantly from determinization and mini-

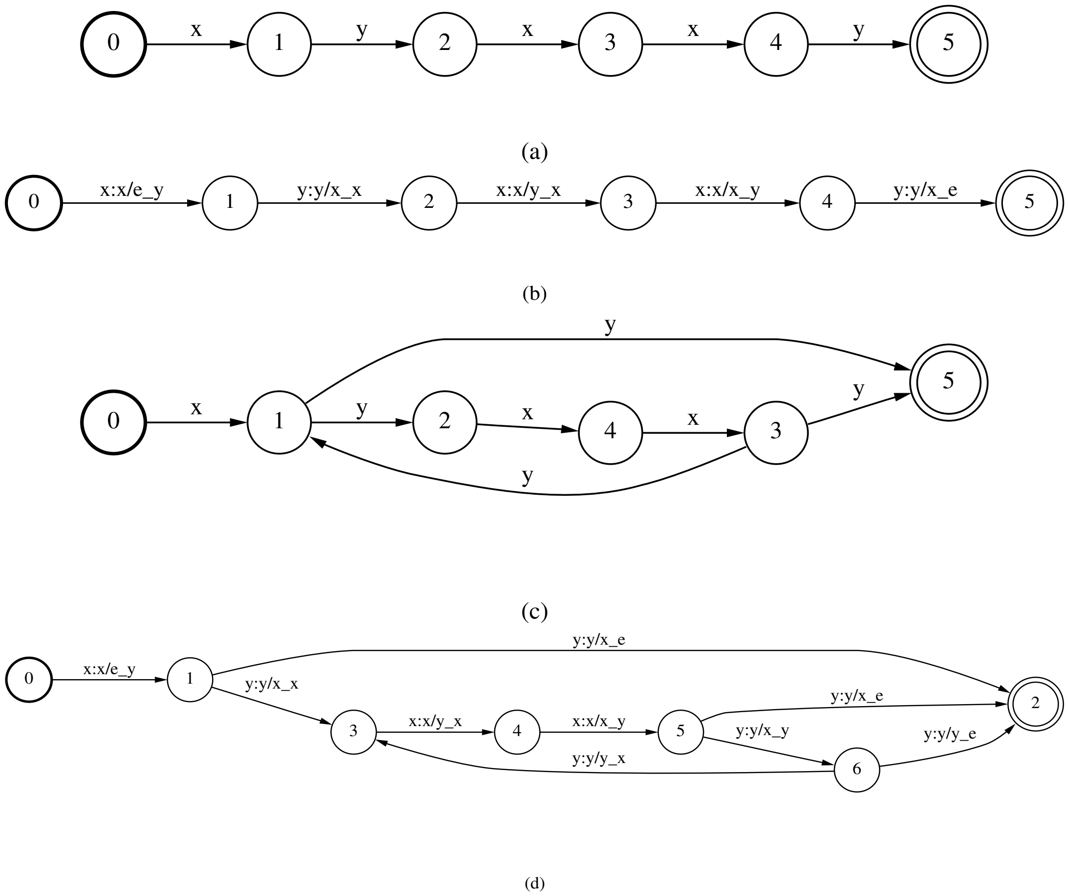


Figure 9: Context-dependent composition examples.

mization since many n-phones share the same HMM model due to the clustering of contexts used to alleviate data sparseness.

The following simple example shows the use of this context-dependency transducer. A context-independent string can be represented by the obvious single-path acceptor as in Figure 9a. This can then be composed with the context-dependency transducer in Figure 8.[§] The result is the transducer in Figure 9b, which has a single path labeled with the context-independent labels on the input side and the corresponding context-dependent labels on the output side.

The context-dependency transducer can be composed with more complex transducers than the trivial one in Figure 9a. For example, composing the context-dependency transducer with the transducer in Figure 9c results in the transducer in Figure 9d. By definition of relational composition, this must correctly replace the context-independent units with the appropriate context-dependent units on all of its paths. Therefore, composition provides a convenient and general mechanism for applying context-dependency to ASR transducers.

[§]Before composition, we promote the acceptor in Figure 9a to the corresponding transducer with identical input and output labels.

If we let C represent a context-dependency transducer from context-dependent phones to context-independent phones, then

$$C \circ L \circ G$$

gives a transducer that maps from context-dependent phones to word sequences restricted to the grammar G. Note that C is the inverse of a transducer such as in Figure 8; that is the input and output labels have been exchanged on all transitions. For notational convenience, we adopt this form of the context-dependency transducer when we use it in recognition cascades.

As we did for the pronunciation lexicon, we can represent the HMM set as H, the closure of the union of the individual HMMs (see Figure 1c). Note that we do not explicitly represent the HMM-state self-loops in H. Instead, we simulate those in the run-time decoder. With H in hand,

$$H \circ C \circ L \circ G$$

gives a transducer that maps from distributions to word sequences restricted to G.

We thus can use composition to combine all levels of our ASR transducers into an integrated transducer in a convenient, efficient and general manner. When these automata are statically provided, we can apply the optimizations discussed in the next section to reduce decoding time and space requirements. If the transducer needs to be modified dynamically, for example by adding the results of a database lookup to the lexicon and grammar in an extended dialogue, we adopt a hybrid approach that optimizes the fixed parts of the transducer and uses lazy composition to combine them with the dynamic portions during recognition [Mohri and Pereira, 1998].

3.2. Transducer Standardization

To optimize an integrated transducer, we use three additional steps; (a) determinization, (b) minimization, and (c) factoring.

3.2.1. Determinization

We use weighted transducer determinization at each step of the composition of each pair of transducers. The main purpose of determinization is to eliminate redundant paths in the composed transducer, thereby substantially reducing recognition time. In addition, its use in intermediate steps of the construction also helps to improve the efficiency of composition and to reduce transducer size.

In general, the transducer $L \circ G$ from phone sequences to words is not determinizable. This is clear in the presence of homophones. But, even without homophones, $L \circ G$ may not be determinizable because the first word of the output sequence might not be known before the entire input phone sequence is scanned. Such unbounded output delays make $L \circ G$ non-determinizable.

To make it possible to determinize $L \circ G$, we introduce an auxiliary phone symbol, denoted $\#_0$, marking the end of the phonetic transcription of each word. Other auxiliary symbols $\#_1 \ldots \#_{k-1}$ are used when necessary to distinguish homophones, as in the following example:

`r eh d` $\#_0$	*read*
`r eh d` $\#_1$	*red*

At most P auxiliary phones are needed, where P is the maximum degree of homophony. The pronunciation dictionary transducer with these auxiliary symbols added is denoted by $\tilde{L}$.

For consistency, the context-dependency transducer C must also accept all paths containing these new symbols. For further determinizations at the context-dependent phone level and distribution level, each auxiliary phone must be mapped to a distinct context-dependent-level symbol. Thus, self-loops are added at each state of C mapping each auxiliary phone to a new auxiliary context-dependent phone. The augmented context-dependency transducer is denoted by $\tilde{C}$.

Similarly, each auxiliary context-dependent phone must be mapped to a new distinct distribution name. P self-loops are added at the initial state of H with auxiliary distribution name input labels and auxiliary context-dependent phone output labels to allow for this mapping. The modified HMM model is denoted by $\tilde{H}$.

It is straightforward to see that the addition of auxiliary symbols guarantees the determinizability of the transducer obtained after each composition, allowing the application of weighted transducer determinization at several stages in our construction.

First, $\tilde{L}$ is composed with G and determinized, yielding $det(\tilde{L} \circ G)$.¶ The benefit of this determinization is the reduction of the number of alternative transitions at each state to at most the number of distinct phones at that state, while the original transducer may have as many as V outgoing transitions at some states where V is the vocabulary size. For large tasks in which the vocabulary has 10^5 to 10^6 words, the advantages of this optimization are clear.

The context-dependency transducer might not be deterministic with respect to the context-independent phone labels. For example, the transducer shown in figure 8 is not deterministic since the initial state has several outgoing transitions with the same input label x or y. To build a small and efficient integrated transducer, it is important to first determinize the inverse of $\tilde{C}$. ‖

¶An n-gram language model G is often constructed as a deterministic weighted automaton with back-off states – in this context, the symbol ϵ is treated as a regular symbol for the definition of determinism. If this does not hold, G is first determinized [Mohri, 1997].

‖Triphonic or more generally n-phonic context-dependency models can be built directly with a deterministic inverse [Riley et al., 1997]. They can also be computed by compilation of context-dependent rules corresponding to each n-phonic context into finite-state transducers [Kaplan and Kay, 1994, Karttunen, 1995, Mohri and Sproat, 1996].

$\tilde{C}$ is then composed with the resulting transducer and determinized. Similarly, $\tilde{H}$ is composed with the context-dependent transducer and determinized. This last determinization increases sharing among HMM models that start with the same distributions. At each state of the resulting integrated transducer, there is at most one outgoing transition labeled with any given distribution name, reducing recognition time even more.

In a final step, we use the erasing operation π_ϵ that replaced the auxiliary distribution symbols by ϵ's. The complete sequence of operations is summarized by the following construction formula:

$$N = \pi_\epsilon(det(\tilde{H} \circ det(\tilde{C} \circ det(\tilde{L} \circ G))))$$

where parentheses indicate the order in which the operations are performed. The result N is an integrated recognition transducer that can be constructed even in very large-vocabulary tasks and leads to a substantial reduction in recognition time, as the experimental results below will show.

3.2.2. Minimization

Once we have determinized the integrated transducer, we can reduce it further by minimization. The auxiliary symbols are left in place, the minimization algorithm is applied, and then the auxiliary symbols are removed:

$$N = \pi_\epsilon(min(det(\tilde{H} \circ det(\tilde{C} \circ det(\tilde{L} \circ G)))))$$

Weighted minimization can be used in different semirings. Both minimization in the tropical semiring and minimization in the log semiring can be used in this context. It is not hard to prove that the results of these two minimizations have exactly the same number of states and transitions and only differ in how weight is distributed along paths. The difference in weights arises from differences in the definition of the key pushing operation for different semirings.

Weight pushing in the log semiring has a very large beneficial impact on the pruning efficacy of a standard Viterbi beam search. In contrast, weight pushing in the tropical semiring, which is based on lowest weights between paths described earlier, produces a transducer that may slow down beam-pruned Viterbi decoding many fold.

To push weights in the log semiring instead of the tropical semiring, the potential function is the $-\log$ of the total probability of paths from the each state to the (super-)final state rather than the lowest weight from the state to the (super-)final state. In other words, the transducer is pushed in terms of probabilities along all future paths from a given state rather than the highest probability over the single best path. By using $-\log$ probability pushing, we preserve a desirable property of the language model, namely that the weights of the transitions leaving each state be normalized as in a probabilistic automaton [Carlyle and Paz, 1971]. We have observed that probability pushing makes pruning more effective [Mohri and Riley, 2001a], and conjecture that this is because the acoustic likelihoods and the transducer probabilities are now *synchronized* to obtain the optimal likelihood ratio test for deciding whether to prune. We further conjecture that this reweighting is the best possible for pruning. A proof of these conjectures will require a careful mathematical analysis of pruning.

One step that has not been described yet is how to compute the reweighting potential function. If the lowest cost path potential function is used, classical single-source shortest path algorithms can be employed [Cormen et al., 1992]. However, adopting the sum of probability mass potential function required significant extensions of the classical algorithms, which are of independent interest [Mohri, 1998].

We have thus *standardized* the integrated transducer in our construction — it is the *unique* deterministic, minimal transducer for which the weights for all transitions leaving any state sum to 1 in probability, up to state relabeling. If one accepts that these are desirable properties of an integrated decoding transducer, then our methods obtain the *optimal* solution among all integrated transducers.

3.2.3. Factoring

For efficiency reasons, our decoder has a separate representation for variable-length left-to-right HMMs, which we will call the *HMM specification*. The integrated transducer of the previous section does not take good advantage of this since, having combined the HMMs into the recognition transducer proper, the HMM specification consists of trivial one-state HMMs. However, by suitably *factoring* the integrated transducer, we can again take good advantage of this feature.

A path whose states other than the first and last have at most one outgoing and one incoming transition is called a *chain*. The integrated recognition transducer just described may contain many chains after the composition with $\tilde{H}$, and after determinization. As mentioned before, we do not explicitly represent the HMM-state self-loops but simulate them in the run-time decoder. The set of all chains in N is denoted by $Chain(N)$.

The input labels of N name one-state HMMs. We can replace the input of each length-n chain in N by a single label naming an n-state HMM. The same label is used for all chains with the same input sequence. The result of that replacement is a more compact transducer denoted by F. The factoring operation on N leads to the following decomposition:

$$N = H' \circ F$$

where H' is a transducer mapping variable-length left-to-right HMM state distribution names to n-state HMMs. Since H' can be separately represented in the decoder's HMM specification, the actual recognition transducer is just F.

Chain inputs are in fact replaced by a single label only when this helps to reduce the size of the transducer. This can be measured by defining the *gain* of the replacement of an input sequence σ of a chain by:

$$G(\sigma) = \sum_{\pi \in Chain(N), i[\pi]=\sigma} |\sigma| - |o[\pi]| - 1$$

where $|\sigma|$ denotes the length of the sequence σ, $i[\pi]$ the input label and $o[\pi]$ the output label of a path π. The replacement of a sequence σ helps reduce the size of the transducer if $G(\sigma) > 0$.

Our implementation of the factoring algorithm allows one to specify the maximum number r of replacements done (the r chains with the highest gain are replaced), as well as the maximum length of the chains that are factored.

Factoring does not affect recognition time. It can however significantly reduce the size of the recognition transducer. We believe that even better factoring methods may be found in the future.

3.2.4. Experimental Results – First-Pass Transducers

We applied the techniques outlined in the previous sections to build an integrated, optimized recognition transducer for a 40,000-word vocabulary North American Business News (NAB) task.** The following models were used:

- Acoustic model of 7,208 distinct HMM states, each with an emission mixture distribution of up to twelve Gaussians.
- Triphonic context-dependency transducer C with 1,525 states and 80,225 transitions.
- 40,000-word pronunciation dictionary L with an average of 1.056 pronunciations per word and an out-of-vocabulary rate of 2.3% on the NAB Eval '95 test set.
- Trigram language model G with 3,926,010 transitions built by Katz's back-off method with frequency cutoffs of 2 for bigrams and 4 for trigrams. It was shrunk with an epsilon of 40 using the method of Seymore and Rosenfeld [1996], which retained all the unigrams, 22.3% of the bigrams and 19.1% of the trigrams. The perplexity on the NAB Eval '95 test set was 164.4 (142.1 before shrinking).

We applied the transducer optimization steps as described in the previous section except that we applied the minimization and weight pushing after factoring the transducer. Table 1 gives the size of the intermediate and final transducers.

Observe that the factored transducer $min(F)$ has only about 40% more transitions than G. The HMM specification H' consists of 430,676 HMMs with an average of 7.2 states per HMM. It occupies only about 10% of the memory of

**Our speech recognition decoder library will soon be made available for non-commercial use. It will include among other utilities the construction and optimization of the recognition transducer described in the previous section and will be accessible from the AT&T Labs – Research Web site [Mohri and Riley, 2001b].

$min(F)$ in the decoder (due to the compact representation possible from its specialized topology). Thus, the overall memory reduction from factoring is substantial.

transducer	states	transitions
G	1,339,664	3,926,010
$L \circ G$	8,606,729	11,406,721
$det(L \circ G)$	7,082,404	9,836,629
$C \circ det(L \circ G))$	7,273,035	10,201,269
$det(H \circ C \circ L \circ G)$	18,317,359	21,237,992
F	3,188,274	6,108,907
$min(F)$	2,616,948	5,497,952

Table 1: Size of the first-pass recognition transducers in the NAB $40,000$-word vocabulary task.

transducer	$\times$ real-time
$C \circ L \circ G$	12.5
$C \circ det(L \circ G)$	1.2
$det(H \circ C \circ L \circ G)$	1.0
$min(F)$	0.7

Table 2: Recognition speed of the first-pass transducers in the NAB $40,000$-word vocabulary task at 83% word accuracy

We used these transducers in a simple, general-purpose, one-pass Viterbi decoder applied to the DARPA NAB Eval '95 test set. Table 3.2.4 shows the recognition speed on a Compaq Alpha 21264 processor for the various optimizations, where the word accuracy has been fixed at 83.0%. We see that the fully-optimized recognition transducer, $min(F)$, substantially speeds up recognition.

To obtain improved accuracy, we can widen the decoder beam [††] and/or use better models in the first pass. In particular, since the offline construction of the recognition transducer used here required approximately an order of magnitude more runtime memory than the size of resulting machine, we performed our initial experiments using a significant shrink of the LM. We are currently experimenting with much less shrunken NAB LMs having acquired more memory and improved the memory usage of our construction.

Alternatively, we can use a two-pass system to obtain improved accuracy, as described in the next section.

[††] These models have an asymptotic wide-beam accuracy of 85.3%.

3.2.5. Experimental Results – Rescoring Transducers

We have applied the optimization techniques to lattice rescoring for a 160,000-word vocabulary NAB task. The following models were used to build lattices in a first pass:

- Acoustic model of 5,520 distinct HMM states, each with an emission mixture distribution of up to four Gaussians.
- Triphonic context-dependency transducer C with 1,525 states and 80,225 transitions.
- 160,000-word pronunciation dictionary L with an average of 1.056 pronunciations per word and an out-of-vocabulary rate of 0.8% on the NAB Eval '95 test set.
- Bigram language model G with 1,238,010 transitions built by Katz's back-off method with frequency cutoffs of 2 for bigrams. It was shrunk with an epsilon of 160 using the method of Seymore and Rosenfeld [1996], which retained all the unigrams and 13.9% of the bigrams. The perplexity on the NAB Eval '95 test set was 309.9.

We used an efficient approximate lattice generation method [Ljolje et al., 1999] to generate word lattices. These word lattices were then used as the 'grammar' in a second rescoring pass. The following models were used in the second pass:

- Acoustic model of 7,208 distinct HMM states, each with an emission mixture distribution of up to twelve Gaussians. The model was adapted to each speaker using a single full-matrix MLLR transform [Leggetter and Woodland, 1995].
- Triphonic context-dependency transducer C with 1,525 states and 80,225 transitions.
- 160,000-word stochastic, TIMIT-trained, multiple-pronunciation lexicon L [Riley et al., 1999].
- 6-gram language model G with 40,383,635 transitions built by Katz's back-off method with frequency cutoffs of 1 for bigrams and trigrams, 2 for 4-grams, and 3 for 5-grams and 6-grams. It was shrunk with an epsilon of 5 using the method of Seymore and Rosenfeld, which retained all the unigrams, 34.6% of the bigrams, 13.6% of the trigrams, 19.5% of the 4-grams, 23.1% of the 5-grams, and 11.73% of the 6-grams. The perplexity on the NAB Eval '95 test set was 156.83.

We applied the transducer optimization steps described in the previous section but only to the level of $L \circ G$ (where G is each lattice). Table 3 shows the speed of second-pass recognition on a Compaq Alpha 21264 processor for these optimizations when the word accuracy is fixed at 88.0% on the DARPA Eval '95 test set.[‡‡] We see that the optimized recognition transducers again substantially speed up recognition. The median number of lattice states and arcs was reduced by $\sim 50\%$ by the optimizations.

transducer	x real-time
$C \circ L \circ G$	.18
$C \circ det(L \circ G)$	.13
$C \circ min(det(L \circ G))$	.02

Table 3: Recognition speed of the second-pass transducers in the NAB $160,000$-word vocabulary task at 88% word accuracy

3.3. Recognizer Combination

It is known that combining the output of different recognizers can improve recognition accuracy [Fiscus, 1997]. We achieve this simply by adding together the negative log probability estimates $-\log P_n(s,x)$ for sentence hypothesis s and utterance x from each of the n recognizer lattices and then select the lowest cost path in this combination. This can be implemented by taking the finite-state intersection of the lattices and then finding the lowest cost path using the acyclic single-source shortest path algorithm [Cormen et al., 1992]. (Recall that the finite-state intersection of two acceptors $A_1 \cap A_2$ is identical to the finite-state composition of $T_1 \circ T_2$ where T_1 and T_2 are the corresponding transducers with identical input and output labels).

Model	context	gender	cep. var. norm.
Mod1	5-phone	dep.	yes
Mod2	5-phone	dep.	no
Mod3	5-phone	indep.	yes
Mod4	5-phone	indep.	no
Mod5	3-phone	dep.	yes
Mod6	3-phone	indep.	no

Table 4: Acoustic models used in the LVCSR-200 task

We used this combination technique in the AT&T submission to the NIST Large Vocabulary Continuous Speech Recognition (LVCSR) 2000 evaluation [Ljolje et al., 2000]. For that system, we used six distinct acoustic models to generate six sets of word lattices. These acoustic models differed in their context-dependency level (triphone vs. pentaphone), whether they were gender-dependent

‡‡The recognition speed excludes the offline transducer construction time.

Word Error Rate (%)						
Model/pass	Mod1	Mod2	Mod3	Mod4	Mod5	Mod6
MLLR	30.3	30.2	30.8	30.7	31.4	32.6
Combined	30.3	29.6	28.9	28.8	28.7	28.6

Table 5: Word error rate on LVCSR-2000 task before and after model combination

and whether they were cepstral variance normalized, as specified in Table 4. All these models were MLLR-adapted. The system used a 40,000 word vocabulary and a 6-gram language model. Table 5 shows the word error rate on the LVCSR Eval '00 test set using each of these models. Also shown are the word error rates after the finite-state intersection of the lattices for the first n acoustic models, where $n = 2$ through 6.[†] As we can see, the six-fold model combination gives an absolute 1.6% word error rate reduction over the best single model.

4. Conclusion

We gave a brief overview of several weighted finite-state transducer algorithms and their application to speech recognition. The algorithms we described are very general. Similar techniques can be used in various other areas of speech processing such as speech synthesis [Sproat, 1997, Beutnagel et al., 1999], in other areas of language technology such as information extraction and phonological and morphological analysis [Kaplan and Kay, 1994, Karttunen, 1995], in optical character recognition, in biological sequence analysis and other pattern matching and string processing applications [Crochemore and Rytter, 1994], and in image processing [Culik II and Kari, 1997], just to mention some of the most active application areas.

5. Acknowledgments

We thank Andrej Ljolje for providing the acoustic models and Don Hindle and Richard Sproat for providing the language models used in our experiments.

References

Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The design and analysis of computer algorithms.* Addison Wesley, Reading, MA, 1974.

Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers, Principles, Techniques and Tools.* Addison Wesley, Reading, MA, 1986.

[†]If a particular lattice used in the intersection gives an empty result (no paths in common), that acoustic model's lattice is skipped for that utterance.

Jean Berstel and Christophe Reutenauer. *Rational Series and Their Languages.* Springer-Verlag, Berlin-New York, 1988.

Mark Beutnagel, Mehryar Mohri, and Michael Riley. Rapid Unit Selection from a Large Speech Corpus for Concatenative Speech Synthesis. In *Proceedings of the 6th European Conference on Speech Communication and Technology (Eurospeech '99)*, Budapest, Hungary, 1999.

J. W. Carlyle and Azaria Paz. Realizations by Stochastic Finite Automaton. *Journal of Computer and System Sciences*, 5:26–40, 1971.

Thomas Cormen, Charles Leiserson, and Ronald Rivest. *Introduction to Algorithms.* The MIT Press, Cambridge, MA, 1992.

Maxime Crochemore and Wojciech Rytter. *Text Algorithms.* Oxford University Press, 1994.

Karel Culik II and Jarkko Kari. Digital Images and Formal Languages. In Grzegorz Rozenberg and Arto Salomaa, editors, *Handbook of Formal Languages*, pages 599–616. Springer, 1997.

J. Fiscus. Post-Processing System to Yield Reduced Word Error Rates: Recognizer Output Voting Error Reduction (ROVER). In *Proceedings of the 1997 IEEE ASRU Workshop*, pages 347–354, Santa Barbara, CA, 1997.

John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison Wesley, Reading, MA, 1979.

Ronald M. Kaplan and Martin Kay. Regular Models of Phonological Rule Systems. *Computational Linguistics*, 20(3), 1994.

Lauri Karttunen. The Replace Operator. In 33^{rd} *Meeting of the Association for Computational Linguistics (ACL 95), Proceedings of the Conference, MIT, Cambridge, Massachussetts.* ACL, 1995.

Werner Kuich and Arto Salomaa. *Semirings, Automata, Languages.* Number 5 in EATCS Monographs on Theoretical Computer Science. Springer-Verlag, Berlin, Germany, 1986.

C. Leggetter and Phil Woodland. Maximum Likelihood Linear Regession for Speaker Adaptation of Continuous Density HMMs. *Computer Speech and Language*, 9(2):171–186, 1995.

Andre Ljolje, Donald Hindle, Michael Riley, and Richard Sproat. The AT&T LVCSR-2000 System. In *Proceedings of the NIST Large Vocabulary Conversational Speech Recognition Workshop*, College Park, Maryland, 2000.

Andrej Ljolje, Fernando Pereira, and Michael Riley. Efficient General Lattice Generation and Rescoring. In *Proceedings of the European Conference on Speech Communication and Technology (Eurospeech '99)*, Budapest, Hungary, 1999.

M. Mohri and F.C.N. Pereira. Dynamic Compilation of Weighted Context-Free Grammars. In *36th Annual Meeting of the ACL and 17th International Conference on Computational Linguistics*, volume 2, pages 891–897, 1998.

Mehryar Mohri. On some Applications of Finite-State Automata Theory to Natural Language Processing. *Journal of Natural Language Engineering*, 2: 1–20, 1996.

Mehryar Mohri. Finite-State Transducers in Language and Speech Processing. *Computational Linguistics*, 23(2), 1997.

Mehryar Mohri. General Algebraic Frameworks and Algorithms for Shortest-Distance Problem. Technical Memorandum 981210-10TM, AT&T Labs - Research, 62 pages, 1998.

Mehryar Mohri and Mark-Jan Nederhof. Regular Approximation of Context-Free Grammars through Transformation. In Jean claude Junqua and Gertjan van Noord, editors, *Robustness in Language and Speech Technology*, pages 153–163. Kluwer Academic Publishers, The Netherlands, 2001.

Mehryar Mohri, Fernando Pereira, and Michael Riley. Weighted Automata in Text and Speech Processing. In *ECAI-96 Workshop, Budapest, Hungary*. ECAI, 1996.

Mehryar Mohri, Fernando Pereira, and Michael Riley. General-purpose Finite-State Machine Software Tools. *http://www.research.att.com/sw/tools/fsm*, AT&T Labs – Research, 1997.

Mehryar Mohri, Fernando Pereira, and Michael Riley. The Design Principles of a Weighted Finite-State Transducer Library. *Theoretical Computer Science*, 231:17–32, January 2000.

Mehryar Mohri and Michael Riley. Network Optimizations for Large Vocabulary Speech Recognition. *Speech Communication*, 25(3), 1998.

Mehryar Mohri and Michael Riley. Integrated Context-Dependent Networks in Very Large Vocabulary Speech Recognition. In *Proceedings of the 6th European Conference on Speech Communication and Technology (Eurospeech '99)*, Budapest, Hungary, 1999.

Mehryar Mohri and Michael Riley. A Weight Pushing Algorithm for Large Vocabulary Speech Recognition. In *Proceedings of the 7th European Conference* *on Speech Communication and Technology (Eurospeech '01)*, Aalborg, Denmark, September 2001a.

Mehryar Mohri and Michael Riley. DCD Library – Speech Recognition Decoder Library. *http://www.research.att.com/sw/tools/dcd*, AT&T Labs – Research, 2001b.

Mehryar Mohri, Michael Riley, Don Hindle, Andrej Ljolje, and Fernando Pereira. Full Expansion of Context-Dependent Networks in Large Vocabulary Speech Recognition. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP '98)*, Seattle, Washington, 1998.

Mehryar Mohri and Richard Sproat. An Efficient Compiler for Weighted Rewrite Rules. In 34*th Meeting of the Association for Computational Linguistics (ACL '96), Proceedings of the Conference, Santa Cruz, California*. ACL, 1996.

Mark-Jan Nederhof. Practical Experiments with Regular Approximation of Context-free Languages. *Computational Linguistics*, 26(1), 2000.

S. Ortmanns, Hermann Ney, and A. Eiden. Language-Model Look-Ahead for Large Vocabulary Speech Recognition. In *Proceedings of the International Conference on Spoken Language Processing (ICSLP'96)*, pages 2095–2098. University of Delaware and Alfred I. duPont Institute, 1996.

Fernando Pereira and Michael Riley. *Finite State Language Processing*, chapter Speech Recognition by Composition of Weighted Finite Automata. The MIT Press, 1997.

Fernando Pereira and Rebecca Wright. Finite-State Approximation of Phrase-Structure Grammars. In E. Roche and Y. Schabes, editors, *Finite-State Language Processing*, pages 149–173. MIT Press, 1997.

Dominique Revuz. Minimisation of Acyclic Deterministic Automata in Linear Time. *Theoretical Computer Science*, 92:181–189, 1992.

Michael Riley, William Byrne, Michael Finke, Sanjeev Khudanpur, Andrej Ljolje, John McDonough, Harriet Nock, Murat Saraclar, Charles Wooters, and George Zavaliagkos. Stochastic pronunciation modelling form hand-labelled phonetic corpora. *Speech Communication*, 29:209–224, 1999.

Michael Riley, Fernando Pereira, and Mehryar Mohri. Transducer Composition for Context-Dependent Network Expansion. In *Proceedings of Eurospeech'97*. Rhodes, Greece, 1997.

Arto Salomaa and Matti Soittola. *Automata-Theoretic Aspects of Formal Power Series*. Springer-Verlag, New York, 1978.

Kristie Seymore and Roni Rosenfeld. Scalable Backoff Language Models. In *Proceedings of ICSLP*, Philadelphia, Pennsylvania, 1996.

Richard Sproat. Multilingual Text Analysis for Text-to-Speech Synthesis. *Journal of Natural Language Engineering*, 2(4):369–380, 1997.