SPEECH RECOGNITION WITH WEIGHTED FINITE-STATE TRANSDUCERS

Mehryar Mohri[1,3] *Fernando Pereira*[2] *Michael Riley*[3]

[1] Courant Institute
251 Mercer Street
New York, NY 10012
mohri@cims.nyu.edu

[2] University of Pennsylvania
200 South 33rd Street
Philadelphia, PA 19104
pereira@cis.upenn.edu

[3] Google Research
76 Ninth Avenue
New York, NY 10011
riley@google.com

ABSTRACT

This chapter describes a general representation and algorithmic framework for speech recognition based on weighted finite-state transducers. These transducers provide a common and natural representation for major components of speech recognition systems, including hidden Markov models (HMMs), context-dependency models, pronunciation dictionaries, statistical grammars, and word or phone lattices. General algorithms for building and optimizing transducer models are presented, including composition for combining models, weighted determinization and minimization for optimizing time and space requirements, and a weight pushing algorithm for redistributing transition weights optimally for speech recognition. The application of these methods to large-vocabulary recognition tasks is explained in detail, and experimental results are given, in particular for the North American Business News (NAB) task, in which these methods were used to combine 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. Another example demonstrates that the same methods can be used to optimize lattices for second-pass recognition.

1. INTRODUCTION

Much of current large-vocabulary speech recognition is based on models such as hidden Markov models (HMMs), lexicons, or n-gram statistical 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, or *string*, to an output string. 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 string to an output string. Weighted transducers are thus a natural choice to represent the probabilistic finite-state models prevalent in speech processing.

We present a detailed view of the use of weighted finite-state transducers 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 speech recognition 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 begin with an overview in Section 2, which informally introduces weighted finite-state transducers and algorithms, and motivates the methods by showing how they are applied to speech recognition. This section may suffice for those only interested in a brief tour of these methods. In the subsequent two sections, we give a more detailed and precise account. Section 3 gives formal definitions of weighted finite-state transducer concepts and corresponding algorithm descriptions. Section 4 gives a detailed description of how to apply these methods to large-vocabulary speech recognition and shows performance results. These sections are appropriate for those who wish to understand the algorithms more fully or wish to replicate the results.

2. OVERVIEW

We start with an informal overview of weighted automata and transducers, outlines of some of the key algorithms that support the ASR applications described in this chapter – *composition*, *determinization*, and *minimization*, and their application to speech recognition.

2.1. Weighted Acceptors

Weighted finite automata (or weighted acceptors) are used widely in automatic speech recognition (ASR). Figure 1 gives simple, familiar examples of weighted automata as used in ASR. The automaton in Figure 1(a) is a toy finite-state *language model*. The legal word strings are specified by the words along each complete path, and their probabilities by the product of the corresponding transition probabilities. The automaton in Figure 1(b) gives the possible pronunciations of one word, `data`, used in the language model. Each legal pronunciation is the phone strings along a complete path, and its probability is given by the product of the corresponding transition probabilities. Finally, the automaton in Figure 1(c) encodes a typical left-to-right, three-distribution-HMM structure for one phone, with the labels along a complete path specifying legal strings of acoustic distributions for that phone.

These automata consist of a set of states, an initial state, a set of final states (with final weights), and a set of transitions between states. Each transition has a source state, a destination state, a label and a weight. Such automata are called *weighted finite-state acceptors* (WFSA), since they *accept* or *recognize* each string that can be read along a path from the start state to a final state. Each accepted string is assigned a weight, namely the accumulated weights along accepting paths for that string, including final weights. An acceptor as a whole represents a set of strings, namely those that it accepts. As a weighted acceptor, it also associates to each accepted string the accumulated weights of their accepting paths.

Speech recognition architectures commonly give the run-time decoder the task of combining and optimizing automata 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 at this point to reduce path redundancy and thus improve search efficiency, especially for large vocabulary recognition [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. In addition, these automata combinations and optimizations are applied in a pre-programmed order to a pre-specified number of levels.

2.2. Weighted Transducers

Our approach uses finite-state transducers, rather than acceptors, to represent the n-gram grammars, pronunciation dictionaries, context-dependency specifications, HMM topology, word, phone or HMM segmentations, lattices and n-best output lists encountered in ASR. The transducer representation provides general methods for combining models and optimizing them, leading to both simple and flexible ASR decoder designs.

A weighted finite-state transducer (WFST) is

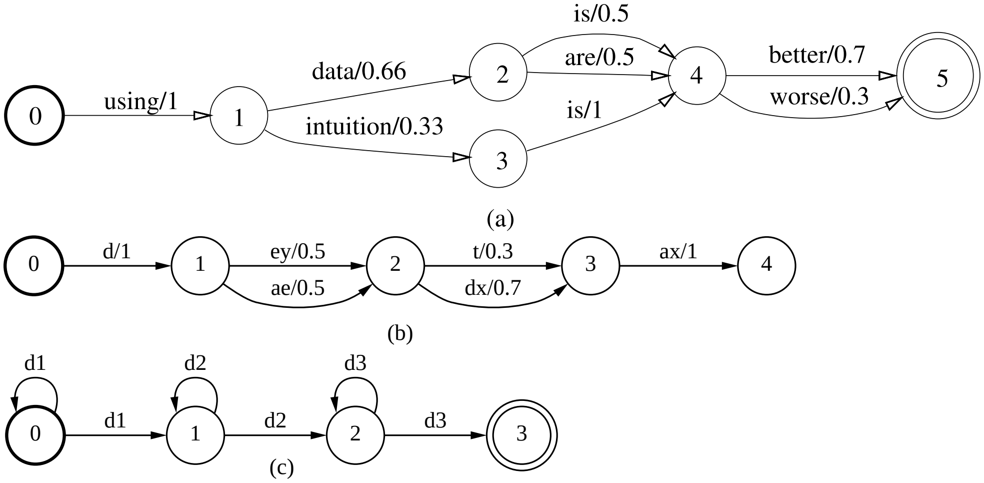


Figure 1: Weighted finite-state acceptor examples. By convention, the states are represented by circles and marked with their unique number. The initial *state* is represented by a bold circle, final states by double circles. The label l and weight w of a transition are marked on the corresponding directed arc by l/w. When explicitly shown, the final weight w of a final state f is marked by f/w.

quite similar to a weighted acceptor except that it has an input label, an output label and a weight on each of its transitions. The examples in Figure 2 encode (a superset of) the information in the WFSAs of Figure 1(a)-(b) as WFSTs. Figure 2(a) represents the same language model as Figure 1(a) by giving each transition identical input and output labels. This adds no new information, but is a convenient way we use often to treat acceptors and transducers uniformly.

Figure 2(b) represents a toy pronunciation lexicon as a mapping from phone strings to words in the lexicon, in this example `data` and `dew`, with probabilities representing the likelihoods of alternative pronunciations. It *transduces* a phone string that can be read along a path from the start state to a final state to a word string with a particular weight. The word corresponding to a pronunciation is output by the transition that consumes the first phone for that pronunciation. The transitions that consume the remaining phones output no further symbols, indicated by the null symbol ϵ as the transition's output label. In general, an ϵ input label marks a transition that consumes no input, and an ϵ output label marks a transition that produces no output.

This transducer has more information than the WFSA in Figure 1(b). 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 1(c) can be combined into a single transducer that preserves phone model identity. This illustrates the key advantage of a transducer over an acceptor: the transducer can represent a relationship between two levels of representation, for instance between phones and words or between HMMs and context-independent phones. More precisely, a transducer specifies a binary relation between strings: two strings are in the relation when there is a path from an initial to a final state in the transducer that has the first string as the sequence of input labels along the path, and the second string as the sequence of output labels along the path (ϵ symbols are left out in both input and output). In general, this is a relation rather than a function since the same input string might be transduced to different strings along two distinct paths. For a weighted transducer, each string pair is also associated with a weight.

We rely on a common set of weighted transducer operations to combine, optimize, search and prune them [Mohri et al., 2000]. Each operation implements a single, well-defined function that has its foundations in the mathematical theory of ratio-

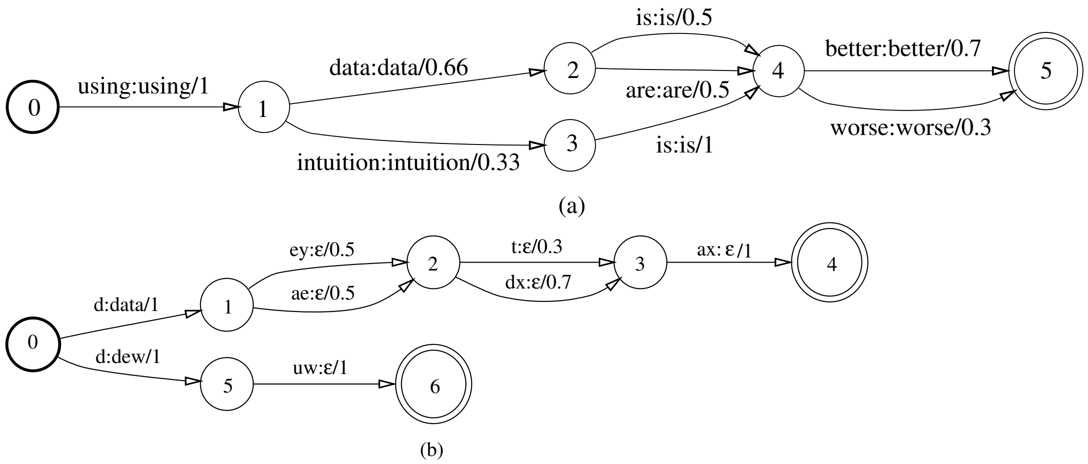


Figure 2: Weighted finite-state transducer examples. These are similar to the weighted acceptors in Figure 1 except output labels are introduced on each transition. The input label i, the output label o, and weight w of a transition are marked on the corresponding directed arc by $i : o/w$.

nal 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., 2000].

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.

Where possible, we provided *lazy* (also called *on-demand*) implementations of algorithms. Any finite-state object `fsm` can be accessed with the three main methods `fsm.start()`, `fsm.final(state)`, and `fsm.transitions(state)` that return the start state, the final weight of a state, and the transitions leaving a state, respectively. This interface can be implemented for concrete automata in an obvious way: the methods simply return the requested information from a stored representation of the automaton. However, the interface can also be given lazy implementations. For example, the lazy union of two automata returns a new lazy `fsm` object. When the object is first constructed, the lazy implementation just initializes internal book-keeping data. It is only when the interface methods request the start state, the final weights, or the transitions (and their destination states) leaving a state, that this information is actually computed, and optionally cached inside the object for later reuse. This approach has the advantage that if only a part of the result of an operation is needed (for example in a pruned search), then the unused part is never computed, saving time and space. We refer the interested reader to the library documentation and an overview of the library [Mohri et al., 2000] for further details on lazy finite-state objects.

We now discuss the key transducer operations that are used in our speech applications for model combination, redundant path removal, and size reduction, respectively.

2.3. Composition

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-

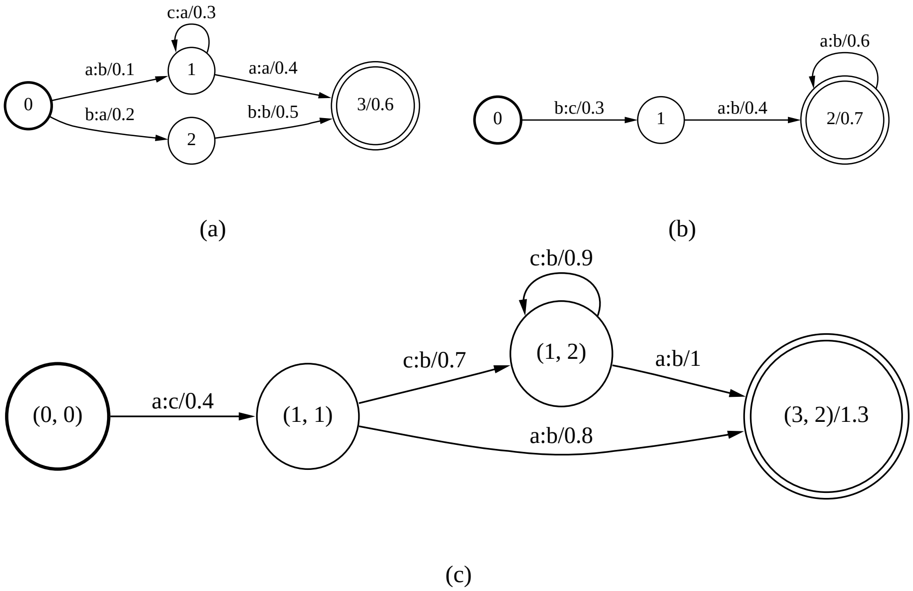


Figure 3: Example of transducer composition.

word transducer whose word strings 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.

As previously noted, a transducer represents a binary relation between strings. The composition of two transducers represents their relational composition. In particular, the composition $T = T_1 \circ T_2$ of two transducers T_1 and T_2 has exactly one path mapping string u to string w for each pair of paths, the first in T_1 mapping u to some string v and the second in T_2 mapping v to w. The weight of a path in T is computed from the weights of the two corresponding paths in T_1 and T_2 with the same operation that computes the weight of a path from the weights of its transitions. If the transition weights represent probabilities, that operation is the product. If instead the weights represent log probabilities or negative log probabilities as is common in ASR for numerical stability, the operation is the sum. More generally, the weight operations for a weighted transducer can be specified by a *semiring* [Salomaa and Soittola, 1978, Berstel and Reutenauer, 1988, Kuich and Salomaa, 1986], as discussed in more detail in Section 3.

The weighted composition algorithm generalizes the classical state-pair construction for finite automata intersection [Hopcroft and Ullman, 1979] to weighted acceptors and transducers. The states of the composition T are pairs of a T_1 state and a T_2 state. T satisfies the following conditions: (1) its initial state is the pair of the initial state of T_1 and the initial state of T_2; (2) its final states are pairs of a final state of T_1 and a final state of T_2, and (3) there is a transition t from (q_1, q_2) to (r_1, r_2) for each pair of transitions t_1 from q_1 to r_1 and t_2 from q_2 to r_2 such that the output label of t_1 matches the input label of t_2. The transition t takes its input label from t_1, its output label from t_2, and its weight is the combination of the weights of t_1 and t_2 done with the same operation that combines weights along a path. Since this computation is *local* — it involves only the transitions leaving two states being paired — it can be given a lazy implementation in which the composition is generated only as needed by other operations on the composed automaton. Transitions with ϵ-labels in T_1 or T_2 must be treated specially as discussed in Section 3. Figure 3 shows two simple transducers, Figure 3(a) and Figure 3(b), and the result of their composition, Figure 3(c). The weight of a path in the resulting transducer is the sum of the weights of the matching paths in T_1 and T_2 (as when the weights represent negative 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.

2.4. Determinization

In a *deterministic automaton*, each state has at most one transition with any given input label and there are no input ϵ-labels. Figure 4(a) gives an example of a non-deterministic weighted acceptor: at state 0, for instance, there are two transitions with the same label a. The automaton in Figure 4(b), on the other hand, is deterministic.

The key advantage of a deterministic automaton over equivalent nondeterministic ones is its irredundancy: it contains at most one path matching any given input string, thereby reducing the time and space needed to process the string. This is particularly important in ASR due to pronunciation lexicon redundancy in large vocabulary tasks. The familiar tree lexicon in ASR is a deterministic pronunciation representation [Ortmanns et al., 1996].

To benefit from determinism, we use a determinization algorithm that transforms a non-deterministic weighted automaton into an equivalent deterministic 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 string and weights to each input string; the distribution of the weight or output labels along paths need not be the same in the two transducers.

If we apply the weighted determinization algorithm to the union of a set of chain automata, each representing a single word pronunciation, we obtain a tree-shaped automaton. However, the result of this algorithm on more general automata may not be a tree, and in fact may be much more compact than a tree. The algorithm can produce results for a broad class of automata with cycles, which have no tree representation.

Weighted determinization generalizes the classical subset method for determinizing unweighted finite automata [Aho et al., 1986], Unlike in the unweighted case, not all weighted automata can be determinized. Conditions for determinizability are discussed in Section 3.3. Fortunately, most weighted automata used in speech processing can be either determinized directly or easily made determinizable with simple transformations, as we discuss in sections 3.3 and 4.1. In particular, any acyclic weighted automaton can always be determinized.

To eliminate redundant paths, weighted determinization needs to calculate the combined weight of all paths with the same labeling. When each path represents a disjoint event with probability given by its weight, the combined weight, representing the probability of the common labeling for that set of paths, would be the sum of the weights of the paths. Alternatively, we may just want to keep the most probable path, as is done in shortest path algorithms, leading to the so-called *Viterbi approximation*. When weights are negative log probabilities, these two alternatives correspond respectively to log summation and taking the minimum. In the general case, we use one operation, denoted $\otimes$, for combining weights along paths and for composition, and a second operation, denoted $\oplus$, to combine identically labeled paths. Some common choices of $(\oplus, \otimes)$ include $(\max, +)$, $(+, *)$, $(\min, +)$, and $(-\log(e^{-x} + e^{-y}), +)$. In speech applications, the first two are appropriate for probabilities, the last two for the corresponding negative log probabilities. More generally, as we will see in Section 3, many of the weighted automata algorithms apply when the two operations define an appropriate semiring. The choices $(\min, +)$, and $(-\log(e^{-x}+e^{-y}), +)$ are called the *tropical* and *log* semirings, respectively.

Our discussion and examples of determinization and, later, minimization will be illustrated with weighted acceptors. The *string* semiring, whose two operations are longest common prefix and concatenation, can be used to treat the output strings as weights. By this method, the transducer case can be handled as well; see Mohri [1997] for details.

We will now work through an example of determinization with weights in the tropical semiring. Figure 4(b) shows the weighted determinization of automaton A_1 from Figure 4(a). In general, the de-

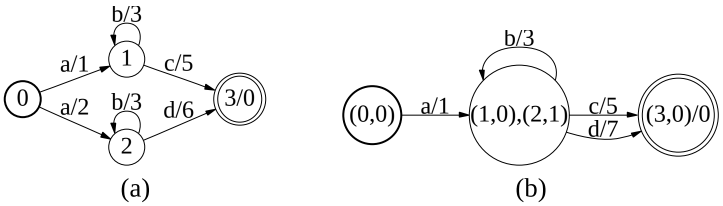


Figure 4: Determinization of weighted automata. (a) Weighted automaton over the tropical semiring A. (b) Equivalent weighted automaton B obtained by determinization of A.

terminization 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 ab in A_1, with weights $\{1+3=4, 2+3=5\}$. The minimum 4 is also the weight associated by A_2 to the string ab.

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, in Figure 4, the initial subset for automaton B 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, in Figure 4, the transition leaving $(0, 0)$ in B labeled with a is obtained from the two transitions labeled with a leaving state 0 in A: its weight is the minimum of the weight of those two transitions, and its destination state is the subset $S' = \{(1, 1-1=0), (2, 2-1=1)\}$. The algorithm is described in more in detail in Section 3.3.

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 that creates states and transitions only as needed.

2.5. Minimization

Given a deterministic automaton, we can reduce its size by minimization, which can save both space and time. Any deterministic unweighted 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, which extends the classical 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.

As we will see in Section 3.5, weighted minimization is quite efficient, indeed as efficient as classical deterministic finite automata (DFA) minimization.

We can view the deterministic weighted automaton A in Figure 5(a) 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. However, since the pairs for different transitions are

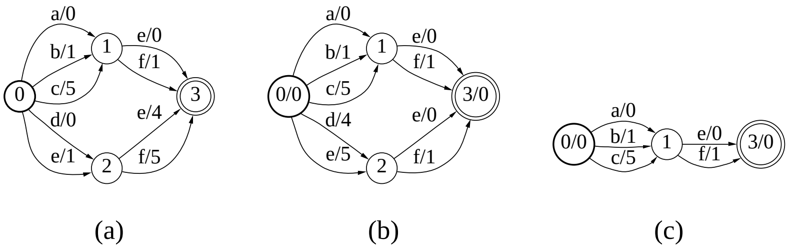


Figure 5: Weight pushing and minimization. (a) Deterministic weighted automaton A. (b) Equivalent weighted automaton B obtained by weight pushing in the tropical semiring. (c) Minimal weighted automaton equivalent to A.

all distinct, classical minimization would have no effect on A.

The size of A can still be reduced by using true weighted minimization. This algorithm works in two steps: the first step *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. Weight pushing is useful not only as a first step of minimization but also to redistribute weight among transitions to improve search, especially pruned search. The algorithm is described in more detail in Section 3.4, and analyzed in [Mohri, 2002]. Its applications to speech recognition are discussed in [Mohri and Riley, 2001].

Pushing is a special case of *reweighting* . We describe reweighting in the case 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, let i be A's initial state and assume for convenience A has a single final state f.[1] Let $V : Q \rightarrow \mathbb{R}$ be an arbitrary *potential* function on states. Update each weight $w[t]$ of the transition t from state $p[t]$ to $n[t]$ as follows:

$$w[t] \leftarrow w[t] + (V(n[t]) - V(p[t])), \tag{1}$$

and the final weight $\rho(f)$ as follows:

$$\rho(f) \leftarrow \rho(f) + (V(i) - V(f)). \tag{2}$$

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:

$$(V(f) - V(i)) + (V(i) - V(f)) = 0. \tag{3}$$

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 5(b) shows the result of pushing for the input A. Thanks to pushing, the size of the automaton can then be reduced using classical minimization. Figure 5(c) illustrates the result of the final step of the algorithm. No approximation or heuristic is used: the resulting automaton C is equivalent to A.

2.6. Speech Recognition Transducers

As an illustration of these methods applied to speech recognition, we describe how to construct a single, statically-compiled and optimized recognition transducer that maps from context-dependent phones to

[1] 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 from each previously final state f with weight $\rho(f)$ an ϵ-transition with the weight $\rho(f)$ to the super-final state.

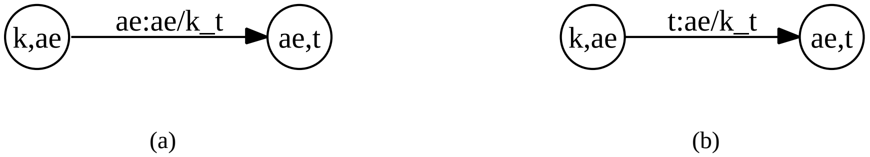


Figure 6: Context-dependent triphone transducer transition: (a) non-deterministic, (b) deterministic.

words. This is an attractive choice for tasks that have fixed acoustic, lexical, and grammatical models since the static transducer can be searched simply and efficiently with no recognition-time overhead for model combination and optimization.

Consider the pronunciation lexicon in Figure 2(b). Suppose we form the union of this transducer with the pronunciation transducers for the remaining words in the grammar G of Figure 2(a) by creating a new super-initial state and connecting an ϵ-transition from that state to the former start states of each pronunciation transducer. We 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 word string from that vocabulary to their corresponding pronunciations. Thus,

$$L \circ G \tag{4}$$

gives a transducer that maps from phones to word strings 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. In particular, the application of the familiar triphone models in ASR to the context-independent transducer, producing a context-dependent transducer, can be performed simply with composition.

To do so, we first construct a context-dependency transducer that maps from context-independent phones to context-dependent triphones. This transducer has a state for every pair of phones and a transition for every context-dependent model. In particular, if ae/k_t represents the triphonic model for ae with left context k and right context t,[2] then there is a transition in the context-dependency transducer from state (k, ae) to state (ae, t) with output label ae/k_t. For the input label on this transition, we could choose the center phone ae as depicted in Figure 6(a). This will correctly implement the transduction; but the transducer will be non-deterministic. Alternately, we can choose the right phone t as depicted in Figure 6(b). This will also correctly implement the transduction, but the result will be deterministic. To see why these are correct, realize that when we enter a state, we have read (in the deterministic case) or must read (in the non-deterministic case) the two phones that label the state. Therefore, the source state and destination state of a transition determine the triphone context. In Section 4, we give the full details of the triphonic context-dependency transducer construction and further demonstrate its correctness.

The above context-dependency transducer maps from context-independent phones to context-dependent triphones. We can invert the relation by interchanging the transducer's input and output labels to create a new transducer that maps from context-dependent triphones to context-independent phones. We do this inversion so we can left compose it with our context-independent recognition transducer $L \circ G$. If we let C represent a context-dependency transducer from context-dependent phones to context-independent phones, then:

$$C \circ (L \circ G) \tag{5}$$

gives a transducer that maps from context-dependent

[2] This use of / to indicate "in the context of" in a triphone symbol offers a potential ambiguity with our use of / to separate a transition's weight from its input and output symbols. However, since context-dependency transducers are never weighted in this chapter, the confusion is not a problem in what follows, so we chose to stay with the notation of previous work rather than changing it to avoid the potential ambiguity.

phones to word strings restricted to the grammar G. To complete our example, we optimize this transducer. Given our discussion of the benefits of determinization and minimization, we might try to apply those operations directly to the composed transducer:

$$N = \min(\det(C \circ (L \circ G))). \tag{6}$$

This assumes the recognition transducer can be determinized, which will be true if each of the component transducers can be determinized. If the context-dependency C is constructed as we have described and if the grammar G is an n-gram language model, then they will be determinizable. However, L may not be determinizable. In particular, if L has ambiguities, namely homophones (two distinct words that have the same pronunciation), then it can not be determinized as is. However, we can introduce auxiliary phone symbols at word ends to disambiguate homophones to create a transformed lexicon $\tilde{L}$. We also need to create a modified context-dependency transducer $\tilde{C}$ that additionally pairs the context-independent auxiliary symbols found in the lexicon with new context-dependent auxiliary symbols (which are later rewritten to epsilons after all optimizations). We leave the details to Section 4. The following expression specifies the optimized transducer:

$$N = \min(\det(\tilde{C} \circ (\tilde{L} \circ G))). \tag{7}$$

In Section 4, we give illustrative experimental results with a fully-composed, optimized (and *factored*) recognition transducer that maps from context-dependent units to words for the North American Business News (NAB) DARPA task. This transducer runs about $18\times$ faster than its unoptimized version and has only about $1.4\times$ times as many transitions as its word-level grammar. We have found similar results with DARPA Broadcast News and Switchboard.

3. ALGORITHMS

We now describe in detail the weighted automata and transducer algorithms introduced informally in Section 2 that are relevant to the design of speech recognition systems. We start with definitions and notation used in specifying and describing the algorithms.

3.1. Preliminaries

As noted earlier, all of our algorithms work with weights that are combined with operations satisfying the *semiring* conditions. A semiring $(\mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1})$ is specified by a set of values $\mathbb{K}$, two binary operations $\oplus$ and $\otimes$, and two designated values $\overline{0}$ and $\overline{1}$. The operation $\oplus$ is associative, commutative, and has $\overline{0}$ as identity. The operation $\otimes$ is associative, has identity $\overline{1}$, distributes with respect to $\oplus$, and has $\overline{0}$ as annihilator: for all $a \in \mathbb{K}, a \otimes \overline{0} = \overline{0} \otimes a = \overline{0}$. If $\otimes$ is also commutative, we say that the semiring is *commutative*. All the semirings we discuss in the rest of this chapter are commutative.

Real numbers with addition and multiplication satisfy the semiring conditions, but of course they also satisfy several other important conditions (for example, having additive inverses), which are not required for our transducer algorithms. Table 3.1 lists some familiar (commutative) semirings. In addition to the Boolean semiring, and the probability semiring used to combine probabilities, two semirings often used in text and speech processing applications are the *log semiring* which is isomorphic to the probability semiring via the negative-$\log$ mapping, and the *tropical semiring* which is derived from the log semiring using the *Viterbi approximation*.

A semiring $(\mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1})$ is *weakly left-divisible* if for any x and y in $\mathbb{K}$ such that $x \oplus y \neq \overline{0}$, there exists at least one z such that $x = (x \oplus y) \otimes z$. The $\otimes$-operation is *cancellative* if z is unique and we can write: $z = (x \oplus y)^{-1} \otimes x$. A semiring is *zero-sum-free* if for any x and y in $\mathbb{K}$, $x \oplus y = \overline{0}$ implies $x = y = \overline{0}$.

For example, the tropical semiring is weakly left-divisible with $z = x - \min(x, y)$, which also shows that $\otimes$ for this semiring is cancellative. The probability semiring is also weakly left-divisible with $z = \frac{x}{x+y}$. Finally, the tropical semiring, the probability semiring, and the log semiring are zero-sum-free.

For any $x \in \mathbb{K}$, let x^n denote

$$x^n = \underbrace{x \otimes \cdots \otimes x}_{n}. \tag{8}$$

When the infinite sum $\bigoplus_{n=0}^{+\infty} x^n$ is well defined and in $\mathbb{K}$, the closure of an element $x \in \mathbb{K}$ is defined as $x^* = \bigoplus_{n=0}^{+\infty} x^n$. A semiring is *closed* when infinite sums such as the one above, are well defined and if associativity, commutativity, and distributivity ap-

Table 1: *Semiring examples.* $\oplus_{\log}$ *is defined by:* $x \oplus_{\log} y = -\log(e^{-x} + e^{-y})$.

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

ply to countable sums (Lehmann [1977] and Mohri [2002] give precise definitions). The Boolean and tropical semirings are closed, while the probability and log semirings are not.

A *weighted finite-state transducer* $T = (\mathcal{A}, \mathcal{B}, Q, I, F, E, \lambda, \rho)$ over a semiring $\mathbb{K}$ is specified by a finite input alphabet $\mathcal{A}$, a finite output alphabet $\mathcal{B}$, a finite set of states Q, a set of initial states $I \subseteq Q$, a set of final states $F \subseteq Q$, a finite set of transitions $E \subseteq Q \times (\mathcal{A} \cup \{\epsilon\}) \times (\mathcal{B} \cup \{\epsilon\}) \times \mathbb{K} \times Q$, an initial state weight assignment $\lambda : I \to \mathbb{K}$, and a final state weight assignment $\rho : F \to \mathbb{K}$. $E[q]$ denotes the set of transitions leaving state $q \in Q$. $|T|$ denotes the sum of the number of states and transitions of T.

Weighted automata (or weighted acceptors) are defined in a similar way by simply omitting the input or output labels. The *projection* operations $\Pi_1(T)$ and $\Pi_2(T)$ obtain a weighted automaton from a weighted transducer T by omitting respectively the input or the output labels of T.

Given a transition $e \in E$, $p[e]$ denotes its origin or previous state, $n[e]$ its destination or next state, $i[e]$ its input label, $o[e]$ its output label, and $w[e]$ its weight. A *path* $\pi = e_1 \cdots e_k$ is a sequence of consecutive transitions: $n[e_{i-1}] = p[e_i]$, $i = 2, \ldots, k$. The path π is a *cycle* if $p[e_1] = n[e_k]$. An ϵ-*cycle* is a cycle in which the input and output labels of all transitions are ϵ.

The functions n, p, and w on transitions can be extended to paths by setting $n[\pi] = n[e_k]$ and $p[\pi] = p[e_1]$, and by defining the weight of a path as the $\otimes$-product of the weights of its constituent transitions: $w[\pi] = w[e_1] \otimes \cdots \otimes w[e_k]$. More generally, w is extended to any finite set of paths R by setting $w[R] = \bigoplus_{\pi \in R} w[\pi]$; if the semiring is closed, this is defined even for infinite R. We denote by $P(q, q')$ the set of paths from q to q' and by $P(q, x, y, q')$ the set of paths from q to q' with input label $x \in \mathcal{A}^*$ and output label $y \in \mathcal{B}^*$. For an acceptor, we denote by $P(q, x, q')$ the set of paths with input label x. These definitions can be extended to subsets $R, R' \subseteq Q$ by $P(R, R') = \cup_{q \in R,\, q' \in R'} P(q, q')$, $P(R, x, y, R') = \cup_{q \in R,\, q' \in R'} P(q, x, y, q')$, and, for an acceptor, $P(R, x, R') = \cup_{q \in R,\, q' \in R'} P(q, x, q')$. A transducer T is *regulated* if the weight associated by T to any pair of input-output strings (x, y), given by

$$T(x, y) = \bigoplus_{\pi \in P(I, x, y, F)} \lambda[p[\pi]] \otimes w[\pi] \otimes \rho[n[\pi]], \quad (9)$$

is well defined and in $\mathbb{K}$. If $P(I, x, y, F) = \emptyset$, then $T(x, y) = \overline{0}$. A weighted transducer without ϵ-cycles is regulated, as is any weighted transducer over a closed semiring. Similarly, for a regulated acceptor, we define

$$T(x) = \bigoplus_{\pi \in P(I, x, F)} \lambda[p[\pi]] \otimes w[\pi] \otimes \rho[n[\pi]]. \quad (10)$$

The transducer T is *trim* if every state occurs in some path $\pi \in P(I, F)$. In other words, a trim transducer has no useless states. The same definition applies to acceptors.

3.2. Composition

As we outlined in Section 2.3, composition is the core operation for relating multiple levels of representation in ASR. More generally, composition is the fundamental algorithm used to create complex weighted transducers from simpler ones [Salomaa and Soittola,

```
Weighted-Composition($T_1, T_2$)
 1  $Q \leftarrow I_1 \times I_2$
 2  $S \leftarrow I_1 \times I_2$
 3  while $S \neq \emptyset$ do
 4      $(q_1, q_2) \leftarrow$ Head($S$)
 5      Dequeue($S$)
 6      if $(q_1, q_2) \in I_1 \times I_2$ then
 7          $I \leftarrow I \cup \{(q_1, q_2)\}$
 8          $\lambda(q_1, q_2) \leftarrow \lambda_1(q_1) \otimes \lambda_2(q_2)$
 9      if $(q_1, q_2) \in F_1 \times F_2$ then
10          $F \leftarrow F \cup \{(q_1, q_2)\}$
11          $\rho(q_1, q_2) \leftarrow \rho_1(q_1) \otimes \rho_2(q_2)$
12      for each $(e_1, e_2) \in E[q_1] \times E[q_2]$ such that $o[e_1] = i[e_2]$ do
13          if $(n[e_1], n[e_2]) \notin Q$ then
14              $Q \leftarrow Q \cup \{(n[e_1], n[e_2])\}$
15              Enqueue($S, (n[e_1], n[e_2])$)
16          $E \leftarrow E \cup \{((q_1, q_2), i[e_1], o[e_2], w[e_1] \otimes w[e_2], (n[e_1], n[e_2]))\}$
17  return $T$
```

Figure 7: Pseudocode of the composition algorithm.

1978, Kuich and Salomaa, 1986], and generalizes the composition algorithm for unweighted finite-state transducers [Eilenberg, 1974-1976, Berstel, 1979]. Let $\mathbb{K}$ be a commutative semiring and let T_1 and T_2 be two weighted transducers defined over $\mathbb{K}$ such that the input alphabet $\mathcal{B}$ of T_2 coincides with the output alphabet of T_1. Assume that the infinite sum $\bigoplus_{z \in \mathcal{B}^*} T_1(x, z) \otimes T_2(z, y)$ is well defined and in $\mathbb{K}$ for all $(x, y) \in \mathcal{A}^* \times \mathcal{C}^*$, where $\mathcal{A}$ is the input alphabet of T_1 and $\mathcal{C}$ is the output alphabet of T_2. This will be the case if $\mathbb{K}$ is closed, or if T_1 has no ϵ-input cycles or T_2 has no ϵ-output cycles. Then, the result of the composition of T_1 and T_2 is a weighted transducer denoted by $T_1 \circ T_2$ and specified for all x, y by:

$$(T_1 \circ T_2)(x, y) = \bigoplus_{z \in \mathcal{B}^*} T_1(x, z) \otimes T_2(z, y). \tag{11}$$

There is a general and efficient composition algorithm for weighted transducers [Salomaa and Soittola, 1978, Kuich and Salomaa, 1986]. States in the composition $T_1 \circ T_2$ of two weighted transducers T_1 and T_2 are identified with pairs of a state of T_1 and a state of T_2. Leaving aside transitions with ϵ inputs or outputs, the following rule specifies how to compute a transition of $T_1 \circ T_2$ from appropriate transitions of T_1 and T_2:

$$\begin{aligned} &(q_1, a, b, w_1, r_1) \text{ and } (q_2, b, c, w_2, r_2) \\ &\Longrightarrow ((q_1, q_2), a, c, w_1 \otimes w_2, (r_1, r_2)). \end{aligned} \tag{12}$$

Figure 7 gives the pseudocode of the algorithm in the ϵ-free case.

The algorithm takes as input two weighted transducers

$$\begin{aligned} T_1 &= (\mathcal{A}, \mathcal{B}, Q_1, I_1, F_1, E_1, \lambda_1, \rho_1) \quad \text{and} \\ T_2 &= (\mathcal{B}, \mathcal{C}, Q_2, I_2, F_2, E_2, \lambda_2, \rho_2), \end{aligned} \tag{13}$$

and outputs a weighted finite-state transducer $T = (\mathcal{A}, \mathcal{C}, Q, I, F, E, \lambda, \rho)$ implementing the composition of T_1 and T_2. E, I, and F are all initialized to the empty set and grown as needed.

The algorithm uses a queue S containing the set of pairs of states yet to be examined. The queue discipline of S is arbitrary, and does not affect the termination of the algorithm. The state set Q is initially the set of pairs of initial states of the original transducers, as is S (lines 1-2). Each time through the loop in lines 3-16, a new pair of states (q_1, q_2) is extracted from S (lines 4-5). The initial weight of (q_1, q_2) is computed by $\otimes$-multiplying the initial weights of q_1 and q_2 when they are both initial states (lines 6-8). Similar steps are followed for final states (lines 9-11). Then, for each pair of matching transitions (e_1, e_2), a new transition is created according to the rule specified earlier (line 16). If the destination state $(n[e_1], n[e_2])$ has not been found previously, it is added to Q and inserted in S (lines 14-15).

In the worst case, all transitions of T_1 leaving a state q_1 match all those of T_2 leaving state q_1', thus the space and time complexity of composition is quadratic: $O(|T_1||T_2|)$. However, a lazy implementation of composition can be used to construct just the part of the composed transducer that is needed.

More care is needed when T_1 has output ϵ labels and T_2 input ϵ labels. Indeed, as illustrated by Figure 8, a straightforward generalization of the ϵ-free case would generate redundant ϵ-paths and, in the case of non-idempotent semirings, would lead to an incorrect result. The weight of the matching paths of the original transducers would be counted p times, where p is the number of redundant paths in the composition.

To solve this problem, all but one ϵ-path must be filtered out of the composition. Figure 8 indicates in boldface one possible choice for that path, which in this case is the shortest. Remarkably, that filtering mechanism can be encoded as a finite-state transducer.

Let $\tilde{T_1}$ and $\tilde{T_2}$ be the weighted transducers obtained from T_1 and T_2 respectively by replacing the output ϵ labels of T_1 with ϵ_2 and the input ϵ labels of T_2 with ϵ_1. Consider the filter finite-state transducer F represented in Figure 9. Then $\tilde{T_1} \circ F \circ \tilde{T_2} = T_1 \circ T_2$. Since the two compositions in $\tilde{T_1} \circ F \circ \tilde{T_2}$ do not involve ϵ labels, the ϵ-free composition already described can be used to compute the resulting transducer.

Intersection (or *Hadamard product*) of weighted automata and composition of finite-state transducers are both special cases of composition of weighted transducers. Intersection corresponds to the case where input and output labels of transitions are identical and composition of unweighted transducers is obtained simply by omitting the weights.

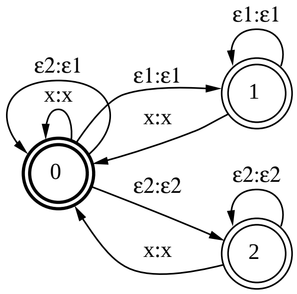


Figure 9: Filter for composition F.

3.3. Determinization

We now describe the generic determinization algorithm for weighted automata that we used informally when working through the example in Section 2.4. This algorithm is a generalization of the classical subset construction for NFAs (unweighted nondeterministic finite automata). The determinization of unweighted or weighted finite-state transducers can both be viewed as special instances of the generic algorithm presented here but, for simplicity, we will focus on the weighted acceptor case.

A weighted automaton is *deterministic* (also known as *subsequential*) if it has a unique initial state and if no two transitions leaving any state share the same input label. The *determinization* algorithm we now present applies to weighted automata over a cancellative weakly left-divisible semiring that satisfies a mild technical condition.[3] Figure 10 gives pseudocode for the algorithm.

A *weighted subset* p of Q is a set of pairs $(q, x) \in Q \times \mathbb{K}$. $Q[p]$ is the set of states q in p, $E[Q[p]]$ is the set of transitions leaving those states, and $i[E[Q[p]]]$ the set of input labels of those transitions.

The states of the result automaton are weighted subsets of the states of the original automaton. A state r of the result automaton that can be reached from the start state by path π is the weighted set of pairs $(q, x) \in Q \times \mathbb{K}$ such that q can be reached from an initial state of the original automaton by a path σ with $i[\sigma] = i[\pi]$ and $\lambda[p[\sigma]] \otimes w[\sigma] =$

[3] If $x \in P(I, Q)$, then $w[P(I, x, Q)] \neq \overline{0}$, which is satisfied by trim automata over the tropical semiring or any other zero-sum-free semiring.

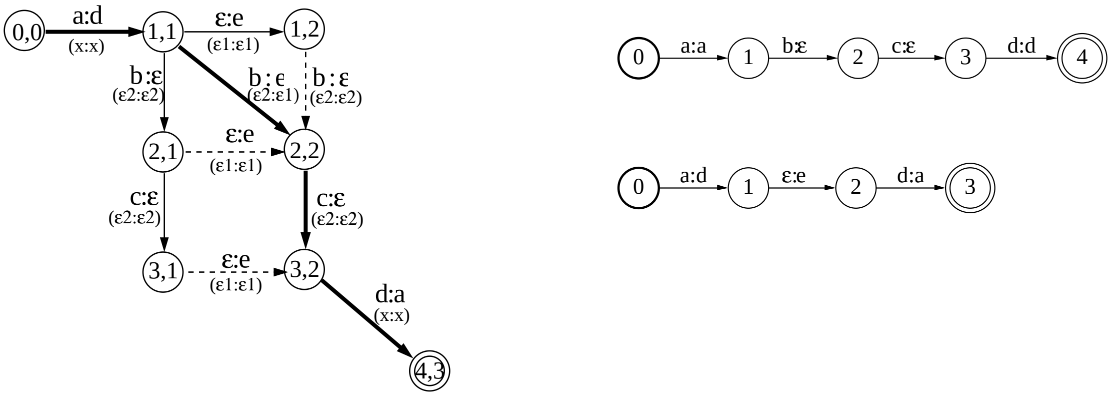


Figure 8: Redundant ϵ-paths. A straightforward generalization of the ϵ-free case could generate all the paths from $(1, 1)$ to $(3, 2)$ when composing the two simple transducers on the right-hand side.

$\lambda[p[\pi]] \otimes w[\pi] \otimes x$. Thus, x can be viewed as the *residual* weight at state q. The algorithm takes as input a weighted automaton $A = (\mathcal{A}, Q, I, F, E, \lambda, \rho)$ and, when it terminates, yields an equivalent deterministic weighted automaton $A' = (\mathcal{A}, Q', I', F', E', \lambda', \rho')$.

The algorithm uses a queue S containing the set of states of the resulting automaton A', yet to be examined. The sets Q', I', F', and E' are initially empty. The queue discipline for S can be arbitrarily chosen and does not affect the termination of the algorithm. The initial state set of A' is $I' = \{i'\}$ where i' is the weighted set of the initial states of A with the respective initial weights. Its initial weight is $\overline{1}$ (lines 1-2). S originally contains only the subset I' (line 3). Each time through the loop in lines 4-16, a new weighted subset p' is dequeued from S (lines 5-6). For each x labeling at least one of the transitions leaving a state p in the weighted subset p', a new transition with input label x is constructed. The weight w' associated to that transition is the sum of the weights of all transitions in $E[Q[p']]$ labeled with x pre-$\otimes$-multiplied by the residual weight v at each state p (line 8). The destination state of the transition is the subset containing all the states q reached by transitions in $E[Q[p']]$ labeled with x. The weight of each state q of the subset is obtained by taking the $\oplus$-sum of the residual weights of the states p $\otimes$-times the weight of the transition from p leading to q and by *dividing* that by w'. The new subset q' is inserted in the queue S when it is a new state (line 16). If any of the states in the subset q' is final, q' is made a final state and its final weight is obtained by summing the final weights of all the final states in q', pre-$\otimes$-multiplied by their residual weight v (line 14-15).

The worst case complexity of determinization is exponential even in the unweighted case. However, in many practical cases such as for weighted automata used in large-vocabulary speech recognition, this blow-up does not occur. It is also important to notice that just like composition, determinization has a natural lazy implementation in which only the transitions required by an application are expanded in the result automaton.

Unlike in the unweighted case, determinization does not halt on all input weighted automata. We say that a weighted automaton A is *determinizable* if the determinization algorithm halts for the input A. With a determinizable input, the algorithm outputs an equivalent deterministic weighted automaton.

The *twins property* for weighted automata characterizes determinizable weighted automata under some general conditions [Mohri, 1997]. Let A be a weighted automaton over a weakly left-divisible semiring $\mathbb{K}$. Two states q and q' of A are said to be *siblings* if there are strings x and y in $\mathcal{A}^*$ such that both q and q' can be reached from I by paths labeled with x and there is a cycle at q and a cycle at q' both labeled with y. When $\mathbb{K}$ is a commutative and cancellative semiring, two sibling states are said to be *twins* when for every string y:

$$w[P(q, y, q)] = w[P(q', y, q')]. \tag{14}$$

```
WEIGHTED-DETERMINIZATION($A$)
$i' \leftarrow \{(i, \lambda(i)) : i \in I\}$
$\lambda'(i') \leftarrow \overline{1}$
$S \leftarrow \{i'\}$
while $S \neq \emptyset$ do
    $p' \leftarrow$ HEAD($S$)
    DEQUEUE($S$)
    for each $x \in i[E[Q[p']]]$ do
        $w' \leftarrow \bigoplus \{v \otimes w : (p, v) \in p', (p, x, w, q) \in E\}$
        $q' \leftarrow \{(q, \bigoplus \{w'^{-1} \otimes (v \otimes w) : (p, v) \in p', (p, x, w, q) \in E\}) :$
            $q = n[e], i[e] = x, e \in E[Q[p']]\}$
        $E' \leftarrow E' \cup \{(p', x, w', q')\}$
        if $q' \notin Q'$ then
            $Q' \leftarrow Q' \cup \{q'\}$
            if $Q[q'] \cap F \neq \emptyset$ then
                $F' \leftarrow F' \cup \{q'\}$
                $\rho'(q') \leftarrow \bigoplus \{v \otimes \rho(q) : (q, v) \in q', q \in F\}$
            ENQUEUE($S, q'$)
return $T'$
```

Figure 10: Pseudocode of the weighted determinization algorithm [Mohri, 1997].

A has *the twins property* if any two sibling states of A are twins. Figure 11 shows a weighted automaton over the tropical semiring that does not have the twins property: states 1 and 2 can be reached by paths labeled with a from the initial state and have cycles with the same label b, but the weights of these cycles (3 and 4) are different.

The following theorems relate the twins property and weighted determinization [Mohri, 1997].

Theorem 1 *Let A be a weighted automaton over the tropical semiring. If A has the twins property, then A is determinizable.*

Theorem 2 *Let A be a trim unambiguous weighted automaton over the tropical semiring.*[4] *Then A is determinizable iff it has the twins property.*

There is an efficient algorithm for testing the twins property for weighted automata [Allauzen and Mohri, 2003]. Note that any acyclic weighted automaton over a zero-sum-free semiring has the twins property and is determinizable.

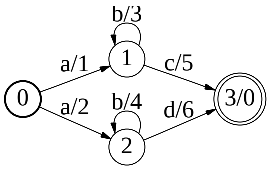


Figure 11: Non-determinizable weighted automaton over the tropical semiring. States 1 and 2 are non-twin siblings.

The *pre-determinization* algorithm can be used to make determinizable an arbitrary weighted transducer over the tropical semiring by inserting transitions labeled with special symbols [Allauzen and Mohri, 2004]. The algorithm makes use of a general twins property [Allauzen and Mohri, 2003] to insert new transitions when needed to guarantee that the resulting transducer has the twins property and thus is determinizable.

[4] A weighted automaton is said to be *unambiguous* if for any $x \in \Sigma^*$ it admits at most one accepting path labeled with x.

3.4. Weight Pushing

As discussed in Section 2.5, *weight pushing* is necessary in weighted minimization, and is also very useful to improve search. Weight pushing can also be used to test the equivalence of two weighted automata. Weight pushing is possible because the choice of the distribution of the total weight along each successful path of a weighted automaton does not affect the total weight of each successful path, and therefore preserves the definition of the automaton as a weighted set (weighted relation for a transducer).

Let A be a weighted automaton over a zero-sum-free and weakly left-divisible semiring $\mathbb{K}$. For any state $q \in Q$, assume that the following sum is well defined and in $\mathbb{K}$:

$$d[q] = \bigoplus_{\pi \in P(q,F)} (w[\pi] \otimes \rho[n[\pi]]). \tag{15}$$

The value $d[q]$ is the *shortest-distance* from q to F [Mohri, 2002]. This is well defined for all $q \in Q$ when $\mathbb{K}$ is a closed semiring. The weight-pushing algorithm consists of computing each shortest-distance $d[q]$ and of *reweighting* the transition weights, initial weights and final weights in the following way:

$$\begin{aligned} w[e] &\leftarrow d[p[e]]^{-1} \otimes w[e] \otimes d[n[e]] \text{ if } d[p[e]] \neq \overline{0}, \\ \lambda[i] &\leftarrow \lambda[i] \otimes d[i], \\ \rho[f] &\leftarrow d[f]^{-1} \otimes \rho[f] \qquad\qquad\qquad \text{if } d[f] \neq \overline{0}, \end{aligned} \tag{16}$$

where e is any transition, i any initial state, and f any final state. Each of these operations can be done in constant time. Therefore, reweighting can be done in linear time $O(T_\otimes|A|)$ where $T_\otimes$ denotes the worst cost of an $\otimes$-operation. The complexity of the shortest-distances computation depends on the semiring [Mohri, 2002]. For the tropical semiring, $d[q]$ can be computed using a standard shortest-distance algorithm. The complexity is linear for acyclic automata, $O(|Q| + (T_\oplus + T_\otimes)|E|)$, where $T_\oplus$ denotes the worst cost of an $\oplus$-operation. For general weighted automata over the tropical semiring, the complexity is $O(|E| + |Q| \log |Q|)$.

For semirings like the probability semiring, a generalization of the Floyd-Warshall algorithm for computing all-pairs shortest-distances can be used [Mohri, 2002]. Its complexity is $\Theta(|Q|^3(T_\oplus + T_\otimes + T_*))$ where T_* denotes the worst cost of the closure operation. The space complexity of these algorithms is $\Theta(|Q|^2)$. Therefore, the Floyd-Warshall algorithm is impractical for automata with several hundred million states or transitions, which arise in large-vocabulary ASR. An approximate version of a generic single-source shortest-distance algorithm can be used instead to compute $d[q]$ efficiently [Mohri, 2002].

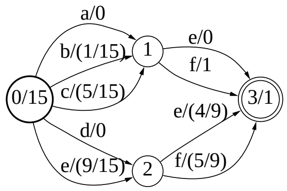


Figure 12: Weighted automaton C obtained from A of Figure 5(a) by weight pushing in the probability semiring.

Speaking informally, the algorithm pushes the weight on each path as much as possible towards the initial states. Figures 5(a)-(b) show weight pushing on the tropical semiring, while Figure 12 shows weight pushing for the same automaton but on the probability semiring.

Note that if $d[q] = \overline{0}$, then, since $\mathbb{K}$ is zero-sum-free, the weight of all paths from q to F is $\overline{0}$. Let A be a weighted automaton over the semiring $\mathbb{K}$. Assume that $\mathbb{K}$ is closed and that the shortest-distances $d[q]$ are all well defined and in $\mathbb{K} - \{\overline{0}\}$. In both cases, we can use the distributivity over the infinite sums defining shortest distances. Let e' (π') denote the transition e (path π) after application of the weight pushing algorithm. e' (π') differs from e (resp. π) only by its weight. Let λ' denote the new initial weight function, and ρ' the new final weight function. Then, the following proposition holds [Mohri, 1997, 2005].

Proposition 1 *Let $B = (\mathcal{A}, Q, I, F, E', \lambda', \rho')$ be the result of the weight pushing algorithm applied to the weighted automaton A, then*

1. *the weight of a successful path π is unchanged after weight pushing:*

$$\begin{aligned} &\lambda'[p[\pi']] \otimes w[\pi'] \otimes \rho'[n[\pi']] = \\ &\lambda[p[\pi]] \otimes w[\pi] \otimes \rho[n[\pi]]. \end{aligned} \tag{17}$$

2. *the weighted automaton B is* stochastic, *that is,*

$$\forall q \in Q, \bigoplus_{e' \in E'[q]} w[e'] = \overline{1}. \tag{18}$$

These two properties of weight pushing are illustrated by figures 5(a)-(b) and 12: the total weight of a successful path is unchanged after pushing; at each state of the weighted automaton of Figure 5(b), the minimum weight of the outgoing transitions is 0, and at each state of the weighted automaton of Figure 12, the weights of outgoing transitions sum to 1.

3.5. Minimization

Finally, we discuss in more detail the minimization algorithm introduced in Section 2.5. A deterministic weighted automaton is said to be *minimal* if there is no other deterministic weighted automaton with a smaller number of states that represents the same mapping from strings to weights. It can be shown that the minimal deterministic weighted automaton has also the minimal number of transitions among all equivalent deterministic weighted automata [Mohri, 1997].

Two states of a deterministic weighted automaton are said to be *equivalent* if exactly the same set of strings label the paths from these states to a final state, and the total weight of the paths for each string, including the final weight of the final state, is the same. Thus, two equivalent states of a deterministic weighted automaton can be merged without affecting the function realized by that automaton. A weighted automaton is minimal when it is not possible to create two distinct equivalent states after any pushing of the weights along its paths.

As outlined in Section 2.5, the general minimization algorithm for weighted automata consists of first applying the weight pushing algorithm to normalize the distribution of the weights along the paths of the input automaton, and then of treating each pair (label, weight) as a single label and applying classical (unweighted) automata minimization [Mohri, 1997]. The minimization of both unweighted and weighted finite-state transducers can also be viewed as instances of the algorithm presented here, but, for simplicity, we will not discuss that further here. The following theorem holds [Mohri, 1997].

Theorem 3 *Let A be a deterministic weighted automaton over a semiring $\mathbb{K}$. Assume that the conditions of application of the weight pushing algorithm hold. Then the execution of the following steps:*

1. *weight pushing,*
2. *(unweighted) automata minimization,*

lead to a minimal weighted automaton equivalent to A.

The complexity of automata minimization is linear in the case of acyclic automata $O(|Q| + |E|)$ and is $O(|E| \log |Q|)$ in the general case. In view of the complexity results of the previous section, for the tropical semiring, the time complexity of the weighted minimization algorithm is linear $O(|Q| + |E|)$ in the acyclic case and $O(|E| \log |Q|)$ in the general case.

Figure 5 illustrates the algorithm in the tropical semiring. Automaton A cannot be further minimized using the classical unweighted automata minimization since no two states are equivalent in that machine. After weight pushing, automaton B has two states, 1 and 2, that can be merged by unweighted automaton minimization.

Figure 13 illustrates the minimization of an automaton defined over the probability semiring. Unlike the unweighted case, a minimal weighted automaton is not unique, but all minimal weighted automata have the same graph topology, and only differ in the weight distribution along each path. The weighted automata B' and C' are both minimal and equivalent to A'. B' is obtained from A' using the algorithm described above in the probability semiring and it is thus a stochastic weighted automaton in the probability semiring.

For a deterministic weighted automaton, the $\oplus$ operation can be arbitrarily chosen without affecting the mapping from strings to weights defined by the automaton, because a deterministic weighted automaton has at most one path labeled by any given string. Thus, in the algorithm described in theorem 3, the weight pushing step can be executed in any semiring $\mathbb{K}'$ whose multiplicative operation matches that of $\mathbb{K}$. The minimal weighted automata obtained by pushing the weights in $\mathbb{K}'$ is also minimal in $\mathbb{K}$, since it can be interpreted as a (deterministic) weighted automaton over $\mathbb{K}$.

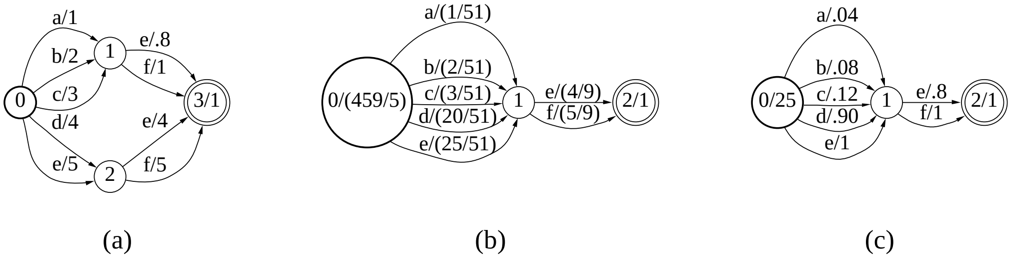


Figure 13: Minimization of weighted automata. (a) Weighted automaton A' over the probability semiring. (b) Minimal weighted automaton B' equivalent to A'. (c) Minimal weighted automaton C' equivalent to A'.

In particular, A' can be interpreted as a weighted automaton over the semiring $(\mathbb{R}_+, \max, \times, 0, 1)$. The application of the weighted minimization algorithm to A' in this semiring leads to the minimal weighted automaton C' of Figure 13(c). C' is also a *stochastic* weighted automaton in the sense that, at any state, the maximum weight of all outgoing transitions is one.

In the particular case of a weighted automaton over the probability semiring, it may be preferable to use weight pushing in the $(\max, \times)$-semiring since the complexity of the algorithm is then equivalent to that of classical single-source shortest-paths algorithms.[5] The corresponding algorithm is a special instance of a generic shortest-distance algorithm [Mohri, 2002].

4. APPLICATIONS TO SPEECH RECOGNITION

We now describe the details of the application of weighted finite-state transducer representations and algorithms to speech recognition as introduced in Section 2.

4.1. Speech Recognition Transducers

As described in Section 2, we will represent various models in speech recognition as weighted-finite state transducers. Four principal models are the word-level grammar G, the pronunciation lexicon L, the context-dependency transducer C, and the HMM transducer H. We will discuss now the construction of each these transducers. Since these will be combined by composition and optimized by determinization, we ensure they are efficient to compose and allow weighted determinization.

The word-level grammar G, whether handcrafted or learned from data, is typically a finite-state model in speech recognition. Hand-crafted finite-state models can be specified by regular expressions, rules or directly as automata. Stochastic n-gram models, common in large vocabulary speech recognition, can be represented compactly by finite-state models. For example, a bigram grammar has a state for every word w_i and a transition from state w_1 to state w_2 for every bigram $w_1 w_2$ that is seen in the training corpus. The transition is labeled with w_2 and has weight $-\log(\hat{p}(w_2|w_1))$, the negative log of the estimated transition probability. The weight of a bigram $w_1 w_3$ that is not seen in the training data can be estimated, for example, by backing-off to the unigram. That is, it has weight $-\log(\beta(w_1)\,\hat{p}(w_3))$, where $\hat{p}(w_3)$ is the estimated w_3 unigram probability and $\beta(w_1)$ is the w_1 backoff weight [Katz, 1987]. The unseen bigram could be represented as a transition from state w_1 to w_3 in the bigram automaton just as a seen bigram. However, this would result in $O(|V|^2)$ transitions in the automaton, where $|V|$ is the vocabulary size. A simple approximation, with the introduction of a *backoff* state b, avoids this. In this model, an unseen $w_1 w_3$ bigram is represented as two transitions: an ϵ-transition from state w_1 to

[5] This preference assumes the resulting distribution of weights along paths is not important. As discussed in the next section, the weight distribution that results from pushing in the $(+, \times)$ semiring has advantages when the resulting automaton is used in a pruned search.

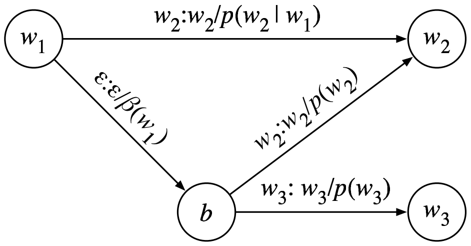


Figure 14: Word bigram transducer model: Seen bigram $w_1 w_2$ represented as a w_2-transition from state w_1 to state w_2; unseen bigram $w_1 w_3$ represented as an ϵ-transition from state w_1 to backoff state b and as a w_3 transition from state b to state w_3.

state b with weight $-\log(\beta(w_1))$ and a transition from state b to state w_3 with label w_3 and weight $-\log(\hat{p}(w_3))$. This configuration is depicted in Figure 14. This is an approximation since seen bigrams may also be read as backed-off unigrams. However, since the seen bigram typically has higher probability than its backed-off unigram, it is usually a good approximation. A similar construction is used for higher-order n-grams.

These grammars present no particular issues for composition. However, the backoff ϵ-transitions introduce non-determinism in the n-gram model. If fully determinized without ϵ-transitions, $O(|V|^2)$ transitions would result. However, we can treat the backoff ϵ labels as regular symbols during determinization, avoiding the explosion in the number of transitions.

As described in Section 2, we represent the pronunciation lexicon L as the Kleene closure of the union of individual word pronunciations as in Figure 2(b). In order for this transducer to efficiently compose with G, the output (word) labels must be placed on the initial transitions of the words; other locations would lead to delays in the composition matching, which could consume significant time and space.

In general, transducer L is not determinizable. This is clear in the presence of homophones. But, even without homophones, it may not be determinizable because the first word of the output string might not be known before the entire phone string is scanned. Such unbounded output delays make L non-determinizable.

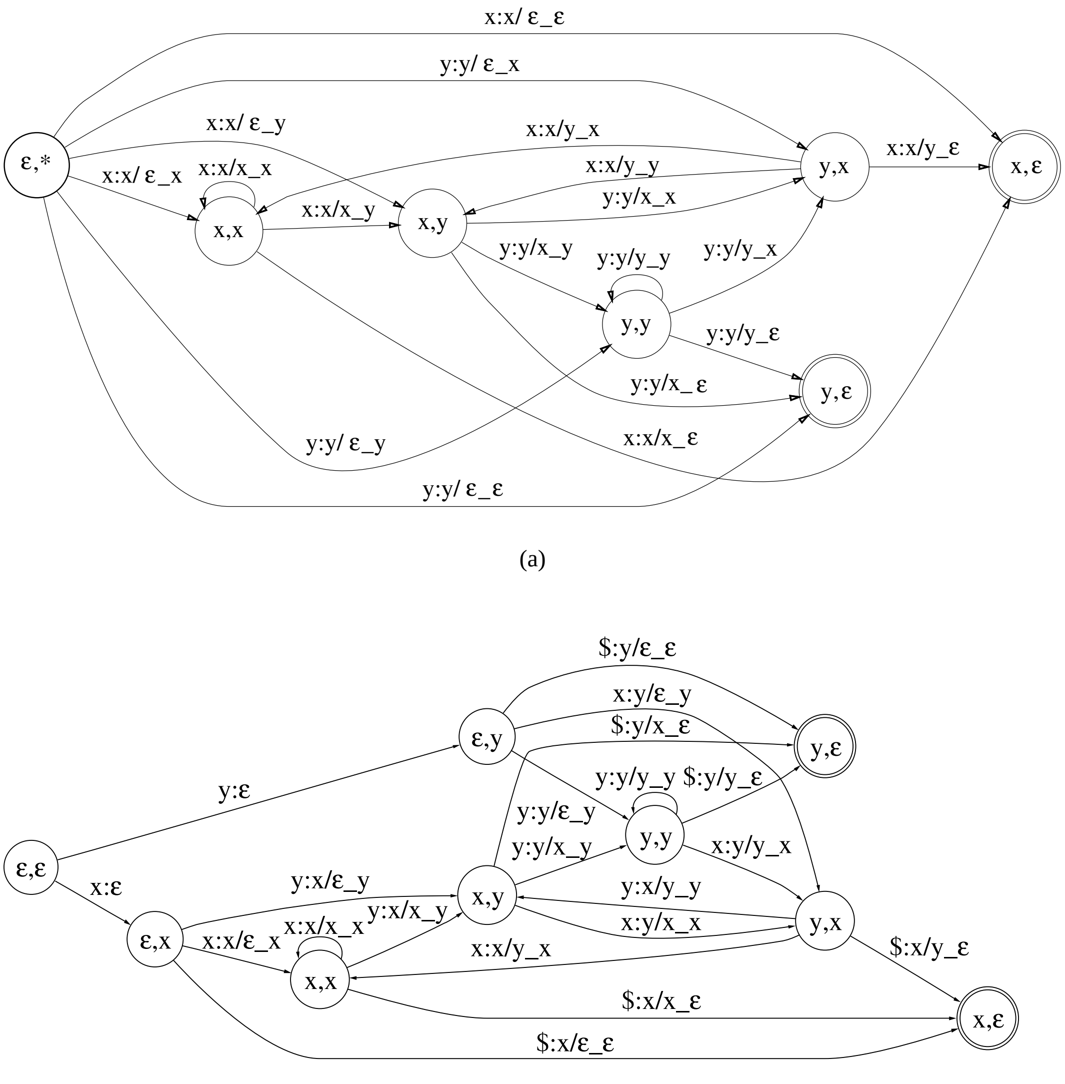
To make it possible to determinize L, we introduce an auxiliary phone symbol, denoted $\#_0$, marking the end of the phonetic transcription of each word. Other auxiliary symbols $\#_1 \ldots \#_{k-1}$ are used when necessary to distinguish homophones, as in the following example:

`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 $\tilde{L}$. Allauzen et al. [2004b] describe more general alternatives to the direct construction of $\tilde{L}$. In that work, so long as L correctly defines the pronunciation transduction, it can be transformed algorithmically to something quite similar to $\tilde{L}$, regardless of the initial disposition of the output labels or the presence of homophony.

As introduced in Section 2, we can represent the mapping from context-independent phones to context-dependent units with a finite-state transducer, with Figure 6 giving a transition of that transducer. Figure 15 gives complete context-dependency transducers where just two hypothetical phones x and y are shown for simplicity. The transducer in Figure 15(a) is non-deterministic, while the one in Figure 15(b) is deterministic. For illustration purposes, we will describe the non-deterministic version since it is somewhat simpler. As in Section 2, we denote the context-dependent units as *phone/left context_right context*. Each state in Figure 15(a) 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 string and $*$ an unspecified future. For instance, it is easy to see that the phone string 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.

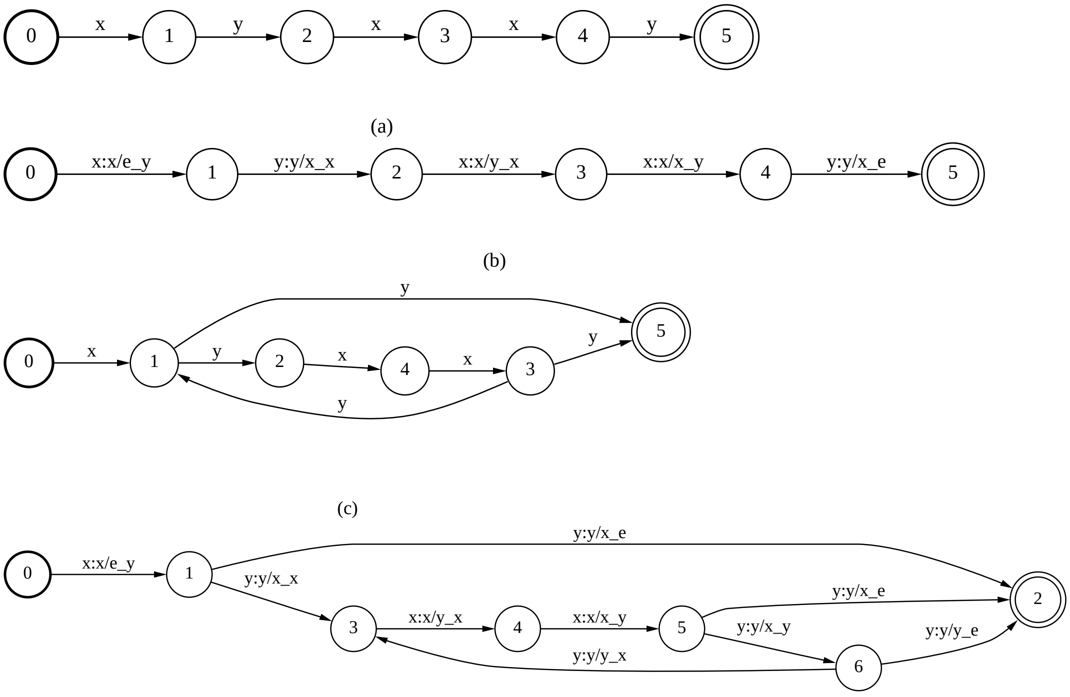
The following simple example shows the use



(a)

(b)

Figure 15: Context-dependent triphone transducers: (a) non-deterministic, (b) deterministic.



(a)

(b)

(c)

(d)

Figure 16: Context-dependent composition examples: (a) context-independent ‘string’, (b) context-dependency applied to *(a)*, (c) context-independent automaton, (d) context-dependency applied to *(c)*.

of this context-dependency transducer. A context-independent string can be represented by the obvious single-path acceptor as in Figure 16(a). This can then be composed with the context-dependency transducer in Figure 15.[6] The result is the transducer in Figure 16(b), 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 16(a). For example, composing the context-dependency transducer with the transducer in Figure 16(c) results in the transducer in Figure 16(d). 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.

The non-determinism of the transducer in Figure 15(a) introduces a single symbol matching delay in the composition with the lexicon. The deterministic transducer in Figure 15(b) composes without a matching delay, which makes it the better choice in applications. However, it introduces a single-phone shift between a context-independent phone and its corresponding context-dependent unit in the result. This shift requires the introduction of a final *subsequential* symbol $\$$ to pad out the context. In practice, this might be mapped to a silence phone or an ϵ-transition.

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 strings restricted to the grammar G. Note that C is the inverse of a transducer such as in Figure 15; 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.

For correctness, the context-dependency transducer C must also accept all paths containing the auxiliary symbols added to $\tilde{L}$ to make it determinizable. For 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 $\tilde{C}$.

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 1(c)). 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 strings restricted to G.

Each auxiliary context-dependent phone in $\tilde{C}$ must be mapped to a new distinct distribution name. 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 $\tilde{H}$.

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 [Riley et al., 1997, Mohri and Pereira, 1998].

4.2. Transducer Standardization

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

[6] Before composition, we promote the acceptor in Figure 16(a) to the corresponding transducer with identical input and output labels.

4.2.1. 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.

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.

$\tilde{C}$ 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 replace the auxiliary distribution symbols by ϵ's. The complete sequence of operations is summarized by the following construction formula:

$$N = \pi_\epsilon(\det(\tilde{H} \circ \det(\tilde{C} \circ \det(\tilde{L} \circ G)))). \qquad (19)$$

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 show.

4.2.2. 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:

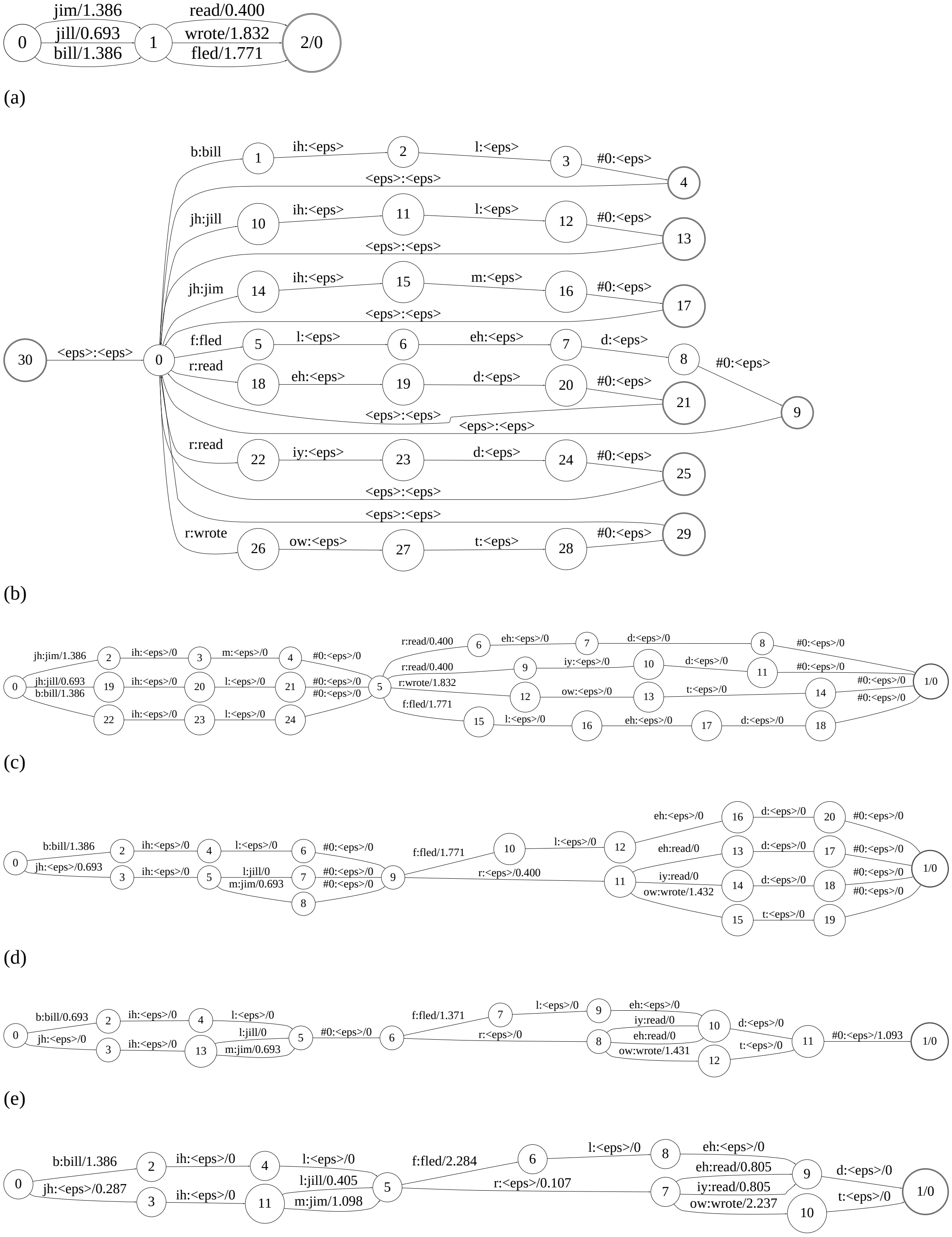
$$N = \pi_\epsilon(\min(\det(\tilde{H} \circ \det(\tilde{C} \circ \det(\tilde{L} \circ G))))). \qquad (20)$$

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 weight 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 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 are normalized as in a probabilistic automaton [Carlyle and Paz, 1971]. We have observed that probability pushing makes pruning more effective [Mohri and Riley, 2001], 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.

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.



(a)

(b)

(c)

(d)

(e)

(f)

Figure 17: Recognition transducer construction: (a) grammar G, (b) lexicon $\tilde{L}$, (c) $\tilde{L} \circ G$, (d) $\det(\tilde{L} \circ G)$, (e) $\min_{\text{tropical}}(\det(\tilde{L} \circ G))$, (f) $\min_{\log}(\det(\tilde{L} \circ G))$.

Figure 17 illustrates the steps in this construction. For simplicity, we consider a small toy grammar and show the construction only down to the context-independent phone level. Figure 17(a) shows the toy grammar G and Figure 17(b) shows the lexicon $\tilde{L}$. Note the word labels on the lexicon are on the initial transitions and that disambiguating auxiliary symbols have been added at the word ends. Figure 17(c) shows their composition $\tilde{L} \circ G$. Figure 17(d) shows the resulting determinization, $\mathrm{det}(\tilde{L} \circ G)$; observe how phone redundancy is removed. Figures 17(e)-(f) show the minimization step, $\min(\mathrm{det}(\tilde{L} \circ G))$; identical futures are combined. In Figure 17(e), the minimization uses weight pushing over the tropical semiring, while in Figure 17(f), the log semiring is used.

4.2.3. 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 $\mathrm{chain}(N)$.

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 string. The result of that replacement is a more compact transducer denoted by F. The factoring operation on N leads to the following decomposition:

$$N = H' \circ F, \qquad (21)$$

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.

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

Table 2: Size of the first-pass recognition transducers in the NAB $40,000$-word vocabulary task.

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 string σ of a chain by:

$$G(\sigma) = \sum_{\pi \in \mathrm{chain}(N), i[\pi]=\sigma} |\sigma| - |o[\pi]| - 1, \qquad (22)$$

where $|\sigma|$ denotes the length of the string σ, $i[\pi]$ the input label and $o[\pi]$ the output label of a path π. The replacement of a string 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.

4.2.4. Experimental Results – First-Pass Transducers

We used the techniques discussed in the previous sections to build many recognizers. To illustrate effectiveness of the techniques and explain some practical details, we discuss here an integrated, optimized recognition transducer for a $40,000$-word vocabulary

transducer	× 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

(a)

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

(b)

Table 3: (a) Recognition speed of the first-pass transducers in the NAB $40,000$-word vocabulary task at 83% word accuracy. (b) Recognition speed of the second-pass transducers in the NAB $160,000$-word vocabulary task at 88% word accuracy.

North American Business News (NAB) task. The following models are 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, 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. Perplexity on the NAB Eval '95 test set is 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 2 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 $\min(F)$ in the decoder (due to the compact representation possible from its specialized topology). Thus, the overall memory reduction from factoring is substantial.

We used these transducers in a simple, general-purpose, one-pass Viterbi decoder applied to the DARPA NAB Eval '95 test set. Table 4.2.4(a) 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 might widen the decoder beam [7], use a larger vocabulary, or use a less shrunken language model. Figure 18(a) shows the affect of vocabulary size (with a bigram LM and optimization only to the $L \circ G$ level). We see that beyond 40,000 words, there is little benefit to increasing the vocabulary either in real-time performance or asymptotically. Figure 18(b) shows the affect of the language model shrinking parameter. These curves were produced by Stephan Kanthak of RWTH using our transducer construction, but RWTH's acoustic models, as part of a comparison with lexical tree methods [Kanthak et al., 2002]. As we can see, decreasing the shrink parameter from 40 as used above to 10 has a significant affect, while further reducing it to 5 has very little affect. An alternative to using a larger LM is to use a two-pass system to obtain improved accuracy, as described in the next section. This has the advantage it allows quite compact shrunken bigram LMs in the first-pass, while the second pass performs as well as the larger-model single pass systems.

While our examples here have been on NAB, we have also applied these methods to Broadcast News [Saraclar et al., 2002], Switchboard, and various AT&T-specific large-vocabulary tasks [Allauzen et al., 2004b]. In our experience, fully-optimized and

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

factored recognition transducers provide very fast decoding while often having well less than twice the number of transitions as their word-level grammars.

4.2.5. Experimental Results – Rescoring Transducers

The weighted transducer approach is also easily applied to multipass recognition. To illustrate this, we now show how to implement lattice rescoring for a $160,000$-word vocabulary NAB task. The following models are 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 is 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. Perplexity on the NAB Eval '95 test set is 309.9.

We used an efficient approximate lattice generation method [Ljolje et al., 1999] to generate word lattices. These word lattices are then used as the 'grammar' in a second rescoring pass. The following models are 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 is 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 is 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. Perplexity on the NAB Eval '95 test set is 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 4.2.4(b) 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.[8] We see that the optimized recognition transducers again substantially speed up recognition. The median number of lattice states and arcs is reduced by ~ 50% by the optimizations.

5. CONCLUSION

We presented an overview of weighted finite-state transducer methods and their application to speech recognition. The methods are quite general, and can also be applied in other areas of speech and language processing, including information extraction, speech synthesis [Sproat, 1997, Allauzen et al., 2004a], phonological and morphological analysis [Kaplan and Kay, 1994, Karttunen, 1995], in optical character recognition, 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.

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.

The first author's work was partially funded by the New York State Office of Science Technology and Academic Research (NYSTAR). The second author's work was

[8] The recognition speed excludes the offline transducer construction time.

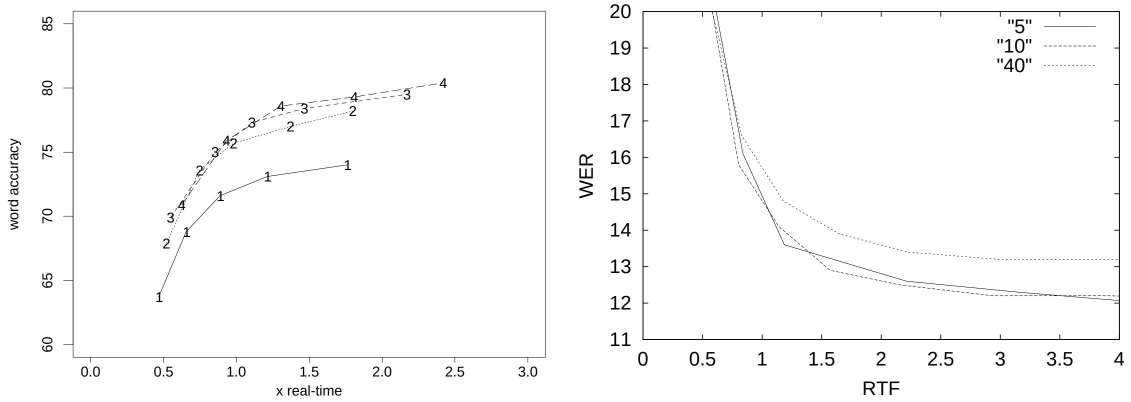


Figure 18: (a) Effect of vocabulary size: NAB bigram recognition results for vocabularies of (1) 10,000 words, (2) 20,000 words, (3) 40,000 words, and (4) 160,000 words (LG Optimized Only). (b) NAB Eval '95 recognition results for the Seymore & Rosenfeld shrink factors of 5, 10, and 40 (thanks to RWTH; uses RWTH acoustic models).

partly funded by NSF grants EIA 0205456, EIA 0205448, and IIS 0428193.

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.

Cyril Allauzen and Mehryar Mohri. An Optimal Pre-Determinization Algorithm for Weighted Transducers. *Theoretical Computer Science*, 328(1-2): 3–18, November 2004.

Cyril Allauzen and Mehryar Mohri. Efficient Algorithms for Testing the Twins Property. *Journal of Automata, Languages and Combinatorics*, 8(2): 117–144, 2003.

Cyril Allauzen, Mehryar Mohri, and Michael Riley. Statistical Modeling for Unit Selection in Speech Synthesis. In 42*nd Meeting of the Association for Computational Linguistics (ACL 2004), Proceedings of the Conference*, Barcelona, Spain, July 2004a.

Cyril Allauzen, Mehryar Mohri, Brian Roark, and Michael Riley. A Generalized Construction of Integrated Speech Recognition Transducers. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2004)*, Montréal, Canada, May 2004b.

Jean Berstel. *Transductions and Context-Free Languages*. Teubner Studienbucher, Stuttgart, 1979.

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

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

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.

Samuel Eilenberg. *Automata, Languages and Machines*, volume A-B. Academic Press, 1974-1976.

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

Stephan Kanthak, Hermann Ney, Michael Riley, and Mehryar Mohri. A Comparison of Two LVR Search Optimization Techniques. In *Proceedings of the International Conference on Spoken Language Processing 2002 (ICSLP '02)*, Denver, Colorado, September 2002.

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.

Slava M. Katz. Estimation of probabilities from sparse data for the language model component of a speech recogniser. *IEEE Transactions on Acoustic, Speech, and Signal Processing*, 35(3):400–401, 1987.

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

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

Daniel J. Lehmann. Algebraic Structures for Transitive Closures. *Theoretical Computer Science*, 4: 59–76, 1977.

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.

Mehryar Mohri. Semiring Frameworks and Algorithms for Shortest-Distance Problems. *Journal of Automata, Languages and Combinatorics*, 7(3): 321–350, 2002.

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

Mehryar Mohri. Statistical Natural Language Processing. In M. Lothaire, editor, *Applied Combinatorics on Words*. Cambridge University Press, 2005.

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 and Fernando 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 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 2001.

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

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

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

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

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

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.

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

Muret Saraclar, Michael Riley, Enrico Bocchieri, and Vincent Goffin. Towards automatic closed captioning : Low latency real time broadcast news transcription. In *Proceedings of the International Conference on Spoken Language Processing (ICSLP'02)*, 2002.

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.

Index