WEIGHTED FINITE-STATE TRANSDUCERS IN SPEECH RECOGNITION

Mehryar Mohri[†], *Fernando Pereira*[‡], *Michael Riley*[†]
[†]AT&T Labs – Research
180 Park AV, Florham Park, NJ 07932 USA
[‡]WhizBang! Labs
4616 Henry Street, Pittsburgh, PA 15213 USA
{mohri,riley}@research.att.com
fpereira@whizbang.com

ABSTRACT

We survey the weighted finite-state transducer (WFST) approach to speech recognition developed at AT&T over the last several years. We show that WFSTs provide a common and natural representation for HMM models, context-dependency, pronunciation dictionaries, grammars, and alternative recognition outputs. Furthermore, general finite-state operations combine these representations flexibly and efficiently. Weighted determinization and minimization algorithms optimize their time and space requirements, and a weight pushing algorithm distributes the weights along the paths of a weighted transducer optimally for speech recognition.

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

1. INTRODUCTION

The application of weighted finite-state transducers to speech recognition [15, 21, 12, 14, 16, 18, 17] and speech synthesis [27] has been the subject of considerable study at AT&T in recent years. A transducer is a finite-state device that encodes a mapping between input and output symbol sequences; a *weighted* transducer associates weights such as probabilities, durations, penalties, or any other quantity that accumulates linearly along paths, to each pair of input and output symbol sequences.

Since many information sources in speech processing involve stochastic finite-state mappings between symbol sequences, weighted transducers are a natural choice to represent them. Further, many of the methods used to combine and optimize these information sources in speech processing can be efficiently implemented in terms of mathematically well-defined primitive operations on weighted transducers.

To explain our approach, we will begin by defining weighted finite-state acceptors and transducers more formally. We will then show some simple speech-related examples and describe several general weighted finite-state operations that are useful in speech applications. Finally, we will give some examples where the transducer representation and finite-state operations are applied to significant speech recognition problems with results that meet certain optimality criteria.

2. WEIGHTED FINITE-STATE TRANSDUCER METHODS

The definitions of the objects presented in this section are given in the most general case and depend on the algebraic structure of a *semiring*, $(\mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1})$ [9]. A semiring is a ring that may lack negation. It has two associative operations $\oplus$ and $\otimes$ that are closed over the set $\mathbb{K}$, they have identities $\overline{0}$ and $\overline{1}$, respectively, and $\otimes$ distributes over $\oplus$. For example, $(\mathbb{N}, +, \cdot, 0, 1)$ is a semiring.

The weights used in speech recognition often represent probabilities. The appropriate semiring to use is then the *probability semiring* $(\mathbb{R}, +, \cdot, 0, 1)$.

For numerical stability, implementations often replace probabilities with log probabilities. The appropriate semiring to use is then the image by $\log$ of the semiring $(\mathbb{R}, +, \cdot, 0, 1)$ and is called the *log semiring*. When log probabilities are used and a Viterbi approximation is assumed, the appropriate semiring is the *tropical semiring* $(\mathbb{R}_+ \cup \{\infty\}, \min, +, \infty, 0)$. The semiring abstraction permits the definition of finite-state representations and algorithms that treat these particular choices as special cases.

In the following definitions, we assume that an arbitrary semiring $(\mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1})$ is given. We will give examples with different semirings to illustrate the use of the semirings just mentioned.

Weighted Acceptors

Network models such as HMMs used in speech recognition are special cases of *weighted finite-state acceptors* (WFSAs). A WFSA $A = (\Sigma, Q, E, i, F, \lambda, \rho)$ over the semiring

$\mathbb{K}$ is given by an alphabet or label set Σ, a finite set of states Q, a finite set of *transitions* $E \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times \mathbb{K} \times Q$, an *initial state* $i \in Q$, a set of *final states* $F \subseteq Q$, an *initial weight* λ and a *final weight function* ρ.

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

A path in A is a sequence of consecutive transitions $t_1 \cdots t_n$ with $t_i{}^+ = t_{i+1}{}^-$, $i = 1, \ldots, n-1$. Transitions labeled with the *empty symbol* ϵ consume no input. A *successful path* $\pi = t_1 \cdots t_n$ is a path from the initial state i to a final state $f \in F$. The label of the path π is the result of the concatenation of the labels of its constituent transitions: $\ell(\pi) = \ell(t_1) \cdots \ell(t_n)$. The weight associated to π is the $\otimes$-product of the initial weight, the weights of its constituent transitions and the final weight $\rho(t_n{}^+)$ of the state reached by π: $w(\pi) = \lambda \otimes w(t_1) \otimes \cdots \otimes w(t_n) \otimes \rho(t_n{}^+)$. A symbol sequence x is accepted by A if there exists a path π labeled with x: $\ell(\pi) = x$. The weight associated by A to the sequence x is then the $\oplus$-sum of the weights of the successful paths π labeled with x. Thus, a WFSA provides a mapping from symbol sequences to weights [25, 4, 9].

Figure 1 gives some simple, familiar examples of WFSAs as used in speech recognition. The automaton in Figure 1a is a toy finite-state *language model*, with the words along each complete path specifying a legal word sequence and the product of the path arc probabilities giving the likelihood of that word sequence. The network in Figure 1b gives the possible pronunciations of one of the words in the language model, `data`, with the phones along each complete path specifying a legal pronunciation and the product of path arc probabilities giving the likelihood of that pronunciation. Finally, the network in Figure 1c encodes a typical left-to-right, three distribution-HMM structure for one phone, with the labels along a complete path specifying legal sequences of acoustic distributions for that phone.

Weighted Transducers

Weighted finite-state transducers (WFSTs) generalize WFSAs by replacing the single transition label by a pair (i, o) of an input label i and an output label o. While a weighted transducer associates symbol sequences and weights, a WFST associates pairs of symbol sequences and weights, that is, it represents a weighted binary relation between symbol sequences [25, 4, 9].[1]

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

A transition $t = (t^-, \ell_i(t), \ell_o(t), w(t), t^+)$ can be represented by an arc from the *source state* t^- to the *destination state* t^+, with the *input label* $\ell_i(t)$, the *output label* $\ell_o(t)$ and the *weight* $w(t)$. The definitions of path, path input label and path weight are those given earlier for acceptors. A path's output label is the concatenation of output labels of its transitions.

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

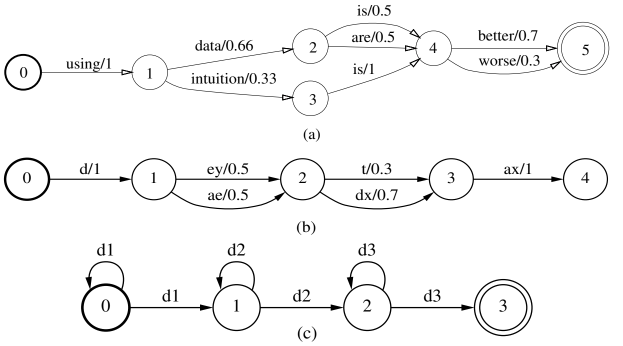

Figure 1: Weighted finite-state acceptor examples. By convention, the states are represented by circles and marked with their unique number. An initial *state* is represented by a bold circle, final states by double circles. The label and weight of a transition t are marked on the corresponding directed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

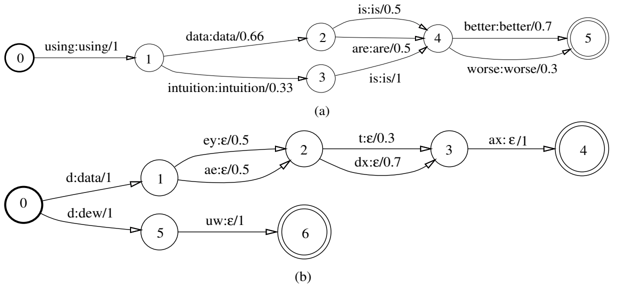

Figure 2: Weighted finite-state transducer examples.

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

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

Weighted Transducer Operations

Speech recognition architectures commonly give the run-time decoder the task of combining and optimizing networks such as those in Figure 1. The decoder finds word pronunciations in its lexicon and substitutes them into the grammar. Phonetic tree representations may be used to improve search efficiency at this point [20]. 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 network. The code that performs these operations is often highly specialized to particular model topologies. For example, the context-dependent models might have to be triphonic, the grammar might be restricted to trigrams, and the alternative pronunciations might have to be enumerated in the lexicon. Further, these network combinations and optimizations are applied in a hardwired sequence to a prespecified number of levels.

Our approach, in contrast, uses a uniform transducer representation for n-gram grammars, pronunciation dictionaries, context-dependency specifications, HMM topology, word, phone or HMM segmentations, lattices and n-best output lists. We then rely on a common set of finite-state operations to combine, optimize, search and prune these automata [15]. Each operation implements a single, well-defined function that has its foundations in the mathematical theory of rational power series [25, 4, 9]. 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 [19].

Basic *union*, *concatenation*, and *Kleene closure* operations combine networks in parallel, in series, and with arbitrary repetition, respectively. Other operations convert transducers to acceptors by projecting onto the input or output label set, find the best or the n best paths in a weighted transducer, remove unreachable states and transitions, and sort acyclic automata topologically. We refer the interested reader to the library documentation and an overview paper [15] for further details on those operations. Here, we will focus on a few key operations that support the ASR applications described in following sections.

Composition/Intersection As previously noted, a transducer represents a binary relation between a pair of symbol sequences. The composition of two transducers represents their relational composition. In particular, the composition $T = R \circ S$ of two transducers R and S has exactly one path mapping sequence u to sequence w for each pair of paths, the first in R mapping u to some sequence v and the second in S mapping v to w. The weight of a path in T is the $\otimes$-product of the weights of the corresponding paths in R and S [25, 4, 9].

Composition is useful for combining different levels of representation. For instance, it can be used to apply a pronunciation lexicon to a word-level grammar to produce a phone-to-word transducer whose word sequences are restricted to the grammar. Many kinds of ASR network combinations, both context-independent and context-dependent, are conveniently and efficiently represented as compositions.

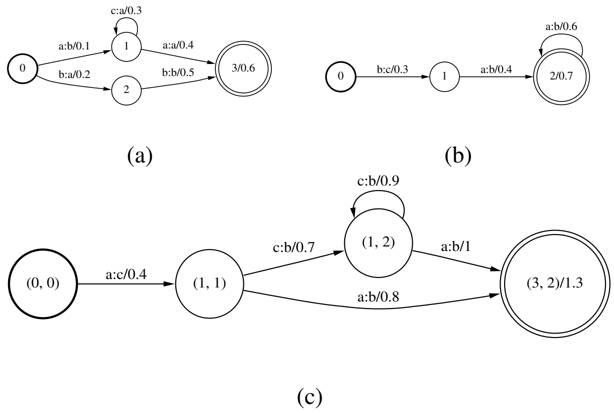

Figure 3: Example of transducer composition.

Our composition algorithm generalizes the classical state-pair construction for finite automata intersection [8] to weighted acceptors and transducers. The composition $R \circ S$ of transducers R and S has pairs of an R state and an S state as states, and satisfies the following conditions: (1) its initial state is the pair of the initial states of R and S; (2) its final states are pairs of a final state of R and a final state of S, and (3) there is a transition t from (r, s) to (r', s') for each pair of transitions t_R from r to r' and t_S from s to s' such that the output label of t matches the input label of t'. The transition t takes its input label from t_R, its output label from t_S, and its weight is the $\otimes$-product of the weights of t_R and t_S when the weights correspond to probabilities. Since this computation is *local* — it involves only the transitions leaving two states being paired — it can be given a *lazy* implementation in which the composition is generated only as needed by other operations on the composed machine. Transitions with ϵ labels in R or S must be treated specially as discussed elsewhere [14, 15].

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

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

Determinization A weighted transducer is *deterministic* or *sequential* if and only if each of its states has at most one transition with any given input label. Figure 4 gives an example of a non-deterministic weighted acceptor: at state 0, for instance, there are several transitions with the same label a.

Weighted determinization, which generalizes the classical subset construction for determinizing finite automata [3],

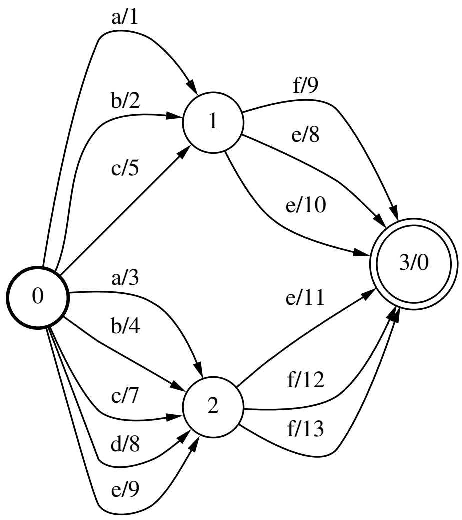

Figure 4: Non-deterministic weighted acceptor A_1.

applies to a weighted automaton and outputs an equivalent deterministic weighted automaton. Two weighted acceptors are *equivalent* if they associate the same weight to each input string; weights may be distributed differently along the paths of two equivalent acceptors. Two weighted transducers are equivalent if they associate the same output sequence and weights to each input sequence; the distribution of the weight or output labels along paths needn't be the same in the two transducers.

In contrast to the unweighted case, not all weighted automata can be determinized, as explained rigorously elsewhere [12]. Fortunately, most weighted automata used in speech processing can be either determinized directly or easily made determinizable by simple transformations, as we shall discuss later. In particular, any acyclic weighted automaton can be determinized.

Our discussion and examples of determinization and, later, minimization will be illustrated with weighted acceptors. The more general weighted transducer case can be shown to be equivalent to this case by interpreting weight-output label pairs as new 'weights' combined by the appropriate semiring [12].

The fundamental advantage of a deterministic automaton over its nondeterministic counterparts is that it is irredundant, that is, it contains at most one path matching any input sequence, thus reducing the time and space required to process an input sequence.

To eliminate redundancy, weighted determinization needs to calculate the combined weight of all the paths for a given input sequence. For instance, in the case, common in speech recognition, where weights are interpreted as (negative) logarithms of probabilities, the weight of a path is obtained by adding the weights of its transitions, and the combined weight for an input string is the minimum of the weights of all paths accepting that string. This is the case for the examples that follow. In the case where weights are probabilities where the overall probability mass of all paths accepting an input is sought, weights are multiplied along a path and summed across paths. Both cases can be handled by the

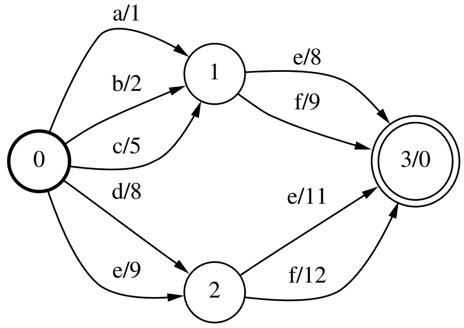

Figure 5: Equivalent weighted automaton A_2 obtained by weighted determinization of A_1.

same algorithm, parameterized with appropriate definitions of the two weight combination operations.

Figure 5 shows the weighted determinization of automaton A_1 from Figure 4. In general, the determinization of a weighted automaton is equivalent to the original, that is, it associates the same weight to each input string. For example, there are two paths corresponding to the input string ae in A_1, with weights $\{1+8 = 9, 3+11 = 14\}$. The minimum 9 is also the weight associated by A_2 to the string ae.

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

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

For example, state 0 in A_2 corresponds to the initial subset $\{(0, 0)\}$ constructed by the algorithm. The A_2 transition leaving 0 and labeled with a is obtained from the two transitions labeled with a leaving the state 0 in A_1: its weight is the minimum of the weight of those two transitions, and its destination state is the subset $S' = \{(1, 1-1 = 0), (2, 3-1 = 2)\}$, numbered 1 in A_2.

Note that the order of expansion of the result automaton does not affect the result so this algorithm also admits a lazy implementation.

Minimization Any deterministic automaton can be minimized using classical algorithms [2, 22]. In the same way, any deterministic weighted automaton A can be minimized using our minimization algorithm [12].

The resulting weighted automaton B is equivalent to the

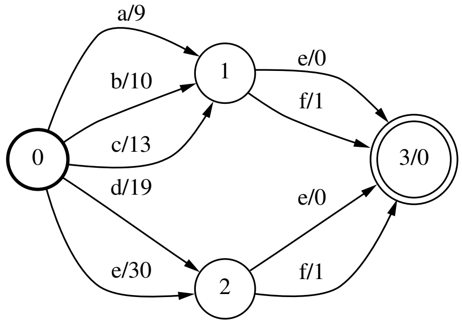

Figure 6: Equivalent weighted automaton B_2 obtained by *pushing* from A_2.

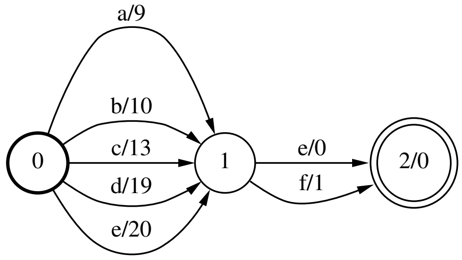

Figure 7: Equivalent weighted automaton A_3 obtained by weighted minimization from A_2.

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

Weighted minimization is quite efficient. Its time complexity is the same as that of classical minimization: linear in the acyclic case ($O(m+n)$), and $O(m \log n)$ in the general case, where n is the number of states and m the number of transitions.

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

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

Pushing is a special case of *reweighting*. We describe reweighting in the case of the tropical semiring, a similar definition can be given in the case of other semirings. A (non-trivial) weighted automaton can be reweighted in an infinite number of ways that produce equivalent automata. To see how, for convenience, assume the automaton A has a single final state f_A.[2] Let $V : Q \to \mathbb{R}$ be an arbitrary *potential* function on states. Update each transition weight as follows:

$$w(t) \leftarrow w(t) + (V(t^+) - V(t^-))$$

and each final weight as follows:

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

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

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

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

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

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

3. WEIGHTED FINITE-STATE TRANSDUCER APPLICATIONS

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

Network Combination

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

$$L \circ G$$

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

We have used composition here to implement a context-independent substitution. However, a major advantage of transducers for speech recognition is that they generalize naturally the notion of context-independent substitution of label by a network to the context-dependent case. The transducer of Figure 8a does not correspond to a simple substitution, since it describes the mapping from context-independent phones to context-dependent triphonic models, denoted by *phone/left context_right context*. Just two hypothetical phones x and y are considered for simplicity.

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

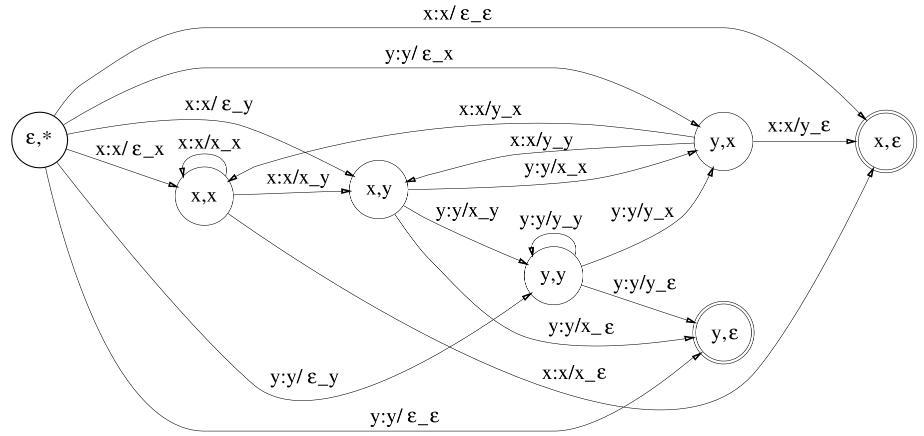

Figure 8: Context-dependent triphone transducer.

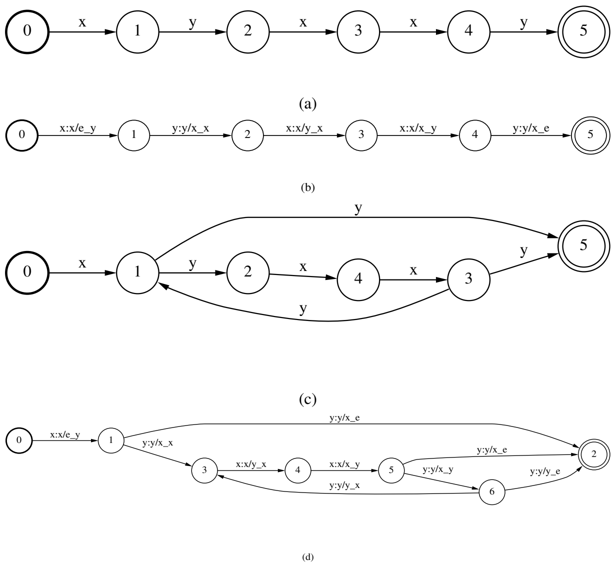

Figure 9: Context-dependent composition examples.

Each state encodes the knowledge of the previous and next phones. State labels in the figure are pairs (a, b) of the past a and the future b, with ϵ representing the start or end of a phone sequence and $*$ an unspecified future. For instance, it is easy to see that the phone sequence xyx is mapped by the transducer to $x/\epsilon\_y\ y/x\_x\ x/y\_\epsilon$ via the unique state sequence $(\epsilon, *)(x, y)(y, x)(x, \epsilon)$. More generally, when there are n context-independent phones, this triphonic construction gives a transducer with $O(n^2)$ states and $O(n^3)$ transitions. A tetraphonic construction would give a transducer with $O(n^3)$ states and $O(n^4)$. In real applications, context-dependency transducers will benefit significantly from determinization and minimization since the n-phone is modeled by an HMM that is likely to be shared among many n-phones due to context clustering required by data sparsity.

The following simple example shows the use of this context-dependency transducer. A context-independent string can represented by the obvious automaton having a single path as in the example in Figure 9a. This can then be composed with the context-dependency transducer in Figure 8.[3] The result is the transducer in Figure 9b, which has a single path labeled with the context-independent labels on the input side and the corresponding context-dependent labels on the output side.

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

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

$$C \circ L \circ G$$

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

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

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

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

We thus can use composition to combine all levels of our ASR network into an integrated network in a convenient, efficient and general manner. When these machines are statically provided, we can apply the optimizations discussed

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

next to reduce decoding time and space requirements. If the network 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 network and uses lazy composition to combine them with the dynamic portions during recognition.

Network Standardization

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

Determinization We use weighted transducer determinization at each step of the composition of each pair of networks. The main purpose of determinization is to eliminate redundant paths in the composed network, 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 network size.

It can be shown that, in general, the transducer $L \circ G$ from phone sequences to words is not determinizable. This is clear in presence of homophones. But, even in the absence of homophones, $L \circ G$ might not be determinizable because, in some cases, the first word of the output sequence cannot be determined before the entire input phone sequence is known. Such unbounded output delays make $L \circ G$ non-determinizable.

To make it possible to determinize $L \circ G$, we introduce an auxiliary phone symbol denoted $\#_0$ marking the end of the phonemic 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, where P is the maximum degree of homophony, are introduced. The pronunciation dictionary transducer augmented with these auxiliary symbols is denoted by $\tilde{L}$.

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

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

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

First, $\tilde{L}$ is composed with G and determinized, yielding $det(\tilde{L} \circ G)$.[4] 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 network 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 $10^5 - 10^6$ words, the advantages of the optimization are clear.

The context-dependency transducer might not be deterministic with respect the context-independent phone labels. For example, the transducer shown in figure 8 is not deterministic since the initial state admits several outgoing transitions with the same input label x or y. To build a small and efficient integrated network, it is important to first determinize the inverse of $\tilde{C}$.[5]

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

As a final step, the auxiliary distribution symbols of the resulting network are simply replaced by ϵ's. The corresponding operation is denoted by π_ϵ. The sequence of operations just described is summarized by the following construction formula:

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

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

Minimization Once we have determinized the integrated network, it is worth minimizing it. To do so, the auxiliary symbols are left in place, the minimization algorithm is applied, and then the auxiliary symbols are removed:

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

In addition to reducing the number of state and transitions, minimization has another another useful effect on recognition performance. As described earlier, a key step of minimization is to push weights toward the initial state. This has an very large effect on pruning for our decoder, which uses a conventional Viterbi beam search. In fact, if pushing were used exactly as described earlier, it slows down decoding many fold. However, with a conceptually simple modification, it has a beneficial effect on the decoding speed.

[4] An n-gram language model G is often constructed as a deterministic weighted automaton with back-off states – in this context, the symbol ϵ is treated as a regular symbol for the definition of determinism. If this does not hold, G is first determinized [12].

[5] Triphonic or more generally n-phonic context-dependency models can be built directly with a deterministic inverse [24].

The modification is that instead of using the lowest weight from each to state to the (super-)final state as the reweighting potential function, the negative log of the sum of all probability mass from each state to the (super-)final state is used. When the integrated network is reweighted with this potential function, the total sum of probabilities over all transitions leaving any state is 1. Thus, 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. In other words, it is pushed with respect the log semiring rather than the tropical semiring. Interestingly, it can be proved that using either pushing in the minimization step results in equivalent machines. However, by using log probability pushing (pushing in the log semiring), a property that holds for the language model now also holds for the integrated network, namely the weights of the transitions leaving each state are normalized as in a probabilistic automaton [5]. We have observed that probability pushing makes pruning more effective, and conjecture that this is because during pruning the acoustic likelihoods and the network probabilities are now *synchronized* to obtain the optimal likelihood ratio test for sequential decisions. We further conjecture that this reweighting is the best possible for pruning. A rigorous proof of these conjectures will, however, require careful mathematical analysis of pruning.

One step that has not been described yet is how to compute the reweighting potential function. If the lowest cost path potential function is used, then classical single-source shortest path algorithms can be employed [6]. However, adopting the sum of probability mass potential function required a significant generalization, of independent interest, to the classical algorithms [13].

We have thus *standardized* the integrated network in our construction — it is the *unique* deterministic, minimal network for which the weights for all transitions leaving any state sum to 1 in probability, up to state relabeling. Therefore, if one accepts that these are desired properties of the result network, then our methods obtain the *optimal* solution among all integrated networks.

Factoring Our decoder has a separate representation for variable-length left-to-right HMMs for efficiency reasons, which we will call the *HMM specification*. However, the integrated network of the previous section does not take good advantage of this since, having combined the HMMs into the recognition network proper, the HMM specification consists of trivial one-state HMMs. However, by suitably *factoring* the integrated network, 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 *linear path*. The integrated recognition network just described may contain many linear paths after the composition with $\tilde{H}$, and after determinization. The set of all linear paths of N is denoted by $\mathrm{Lin}(N)$.

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

$$N = H' \circ F$$

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

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

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

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

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

Factoring does not affect recognition time, however it can lead to a substantial reduction in the size of the network. We believe more effective, less constrained factoring methods may be found in the future.

Experimental Results – First-Pass Networks We applied the techniques outlined in the previous sections to build an integrated, optimized recognition network for a $40,000$-word vocabulary North American Business News (NAB) task. The following models were used:

- Acoustic model of 7,208 distinct HMM states, each with an emission mixture distribution of up to twelve Gaussians.
- Triphonic context-dependency network C with 1,525 states and 80,225 transitions.
- $40,000$-word pronunciation dictionary L.
- 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 and shrunk with an epsilon of 40 using the method of Seymore and Rosenfeld [26].

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

Observe that the factored network *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

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

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

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

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

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 networks in a simple, general-purpose, one-pass Viterbi decoder applied to the DARPA NAB Eval '95 test set. Table 3 shows the speed of recognition on an Compaq Alpha 21284 processor for the various optimizations, where the word accuracy has been fixed at 83.0%. We see that the fully-optimized recognition network, $min(F)$, substantially speeds up recognition.

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

- Acoustic model of 5,520 distinct HMM states, each with an emission mixture distribution of up to four Gaussians.
- Triphonic context-dependency network C with 1,525 states and 80,225 transitions.
- $160,000$-word pronunciation dictionary L.
- bigram language model G with 1,238,010 transitions built by Katz's back-off method with frequency cutoffs of 2 for bigrams and shrunk with an epsilon of 160 using the method of Seymore and Rosenfeld [26].

We used an efficient approximate lattice generation method [1] to generate word lattices. These word lattices were then used as the 'grammar' in a second rescoring pass. The following models were used in the second pass:

- Acoustic model of 7,208 distinct HMM states, each with an emission mixture distribution of up to four Gaussians. The model was adapted to each speaker using a single full-matrix MLLR transform [10].

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

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

- Triphonic context-dependency network C with 1,525 states and 80,225 transitions.
- $160,000$-word stochastic, TIMIT-trained, multiple-pronunciation lexicon L [23].
- 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, and shrunk with an epsilon of 5 using the method of Seymore and Rosenfeld.

We applied the network optimization steps described in the previous section but only to the level of $L \circ G$ (where G is each lattice). Table 3 shows the the speed of second-pass recognition on an Compaq Alpha 21284 processor for these optimizations when the word accuracy is fixed at 88.0% on the DARPA Eval '95 test set.[6] We see that the optimized recognition networks again substantially speed up recognition.

Recognizer Combination

It is known that combining the output of different recognizers can improve recognition accuracy. [7]. We describe a simple lattice-based method for accomplishing this. Our method is to add together the negative log probability estimate $-logP_n(s,x)$ for sentence hypothesis s and utterance x from each of the n recognizer lattices and then select the lowest cost path in this combination. This can be implemented by taking the finite-state intersection of the lattices and then finding the lowest cost path using the acyclic single-source shortest path algorithm [6]. (Recall that the finite-state intersection of two acceptors $A_1 \cap A_2$ is identical to the finite-state composition of $T_1 \circ T_2$ where T_1 and T_2 are the corresponding transducers with identical input and output labels).

We used this combination technique in the AT&T submission to the NIST Large Vocabulary Continuous Speech Recognition (LVCSR) 2000 evaluation [11]. For that system, we used six distinct acoustic models to generate six sets of word lattices. These acoustic models differed in their context-dependency level (triphone vs. pentaphone), whether they were gender-dependent and whether they were cepstral variance normalized. All these models were MLLR-adapted. The system used a 40,000 word vocabulary and a 6-gram language model. Table 4 shows the word error rate on the LVCSR Eval '00 test set using each of these models. Also shown are the word error rates after the finite-state intersection of the lattices for the first n acoustic mod-

[6] The recognition speed excludes the offline network construction time.

Word Error Rate (%)						
Model/pass	penta_GD_vn	penta_GD_nvn	penta_GI_vn	penta_GI_n vn	tri_GD_vn	tri_GI_nvn
MLLR	30.3	30.2	30.8	30.7	31.4	32.6
Combined	30.3	29.6	28.9	28.8	28.7	28.6

Table 4: Word error rate on LVCSR-2000 task before and after model combination

els, where $n = 2$ through 6.[7] As we can see, the six-fold model combination gives an absolute 1.6% word error rate reduction over the best single model.

4. CONCLUSION

We gave a brief overview of weighted finite-state transducer methods and their application to speech recognition. The algorithms we described are very general. Similar techniques can be used in various other areas of speech processing such as speech synthesis, in text and image processing, and in many other fields.

REFERENCES

[1] A. Ljolje and F. Pereira and M. Riley. Efficient general lattice generation and rescoring. In *Proceedings of the European Conference on Speech Communication and Technology (Eurospeech '99)*, Budapest, Hungary, 1999.

[2] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The design and analysis of computer algorithms*. Addison Wesley: Reading, MA, 1974.

[3] A. V. Aho, R. Sethi, and J. D. Ullman. *Compilers, Principles, Techniques and Tools*. Addison Wesley: Reading, MA, 1986.

[4] J. Berstel and C. Reutenauer. *Rational Series and Their Languages*. Springer-Verlag: Berlin-New York, 1988.

[5] J. W. Carlyle and A. Paz. Realizations by stochastic finite automaton. *Journal of Computer and System Sciences*, 5:26–40, 1971.

[6] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms*. The MIT Press: Cambridge, MA, 1992.

[7] J. Fiscus. Post-processing system to yield reduced word error rates: Recognizer output voting error reduction (rover). In *Proceedings of the 1997 IEEE ASRU Workshop*, pages 347–354, Santa Barbara, CA, 1997.

[8] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison Wesley: Reading, MA, 1979.

[9] W. Kuich and A. Salomaa. *Semirings, Automata, Languages*. Number 5 in EATCS Monographs on Theoretical Computer Science. Springer-Verlag, Berlin, Germany, 1986.

[10] C. Leggetter and P. Woodland. Maximum likelihood linear regession for speaker adaptation of continuous density hmms. *Computer Speech and Language*, 9(2):171–186, 1995.

[11] A. Ljolje, D. Hindle, M. Riley, and R. Sproat. The at&t lvcsr-2000 system. In *Proceedings of the NIST Large Vocabulary Conversational Speech Recognition Workshop*, College Park, Maryland, 2000.

[12] M. Mohri. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23:2, 1997.

[13] M. Mohri. General Algebraic Frameworks and Algorithms for Shortest-Distance Problems. Technical Memorandum 981210-10TM, AT&T Labs - Research, 62 pages, 1998.

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

[14] M. Mohri, F. C. N. Pereira, and M. Riley. Weighted automata in text and speech processing. In *ECAI-96 Workshop, Budapest, Hungary*. ECAI, 1996.

[15] M. Mohri, F. C. N. Pereira, and M. Riley. The design principles of a weighted finite-state transducer library. *Theoretical Computer Science*, 231:17–32, January 2000.

[16] M. Mohri and M. Riley. Network optimizations for large vocabulary speech recognition. *Speech Communication*, 25:3, 1998.

[17] M. Mohri and M. 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.

[18] M. Mohri, M. Riley, D. Hindle, A. Ljolje, and F. C. N. 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.

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

[20] S. Ortmanns, H. 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.

[21] F. C. N. Pereira and M. Riley. *Finite State Language Processing*, chapter Weighted Rational Transductions and their Application to Human Language Processing. The MIT Press, 1997.

[22] D. Revuz. Minimisation of acyclic deterministic automata in linear time. *Theoretical Computer Science*, 92:181–189, 1992.

[23] M. Riley, W. Byrne, M. Finke, S. Khudanpur, A. Ljolje, J. McDonough, H. Nock, M. Saraclar, C. Wooters, and G. Zavaliagkos. Stochastic pronunciation modelling form hand-labelled phonetic corpora. *Speech Communication*, 29:209–224, 1999.

[24] M. Riley, F. C. N. Pereira, and M. Mohri. Transducer composition for context-dependent network expansion. In *Proceedings of Eurospeech'97*. Rhodes, Greece, 1997.

[25] A. Salomaa and M. Soittola. *Automata-Theoretic Aspects of Formal Power Series*. Springer-Verlag: New York, 1978.

[26] K. Seymore and R. Rosenfeld. Scalable backoff language models. In *Proceedings of ICSLP*, Philadelphia, Pennsylvania, 1996.

[27] R. Sproat. Multilingual text analysis for text-to-speech synthesis. *Journal of Natural Language Engineering*, 2(4):369–380, 1997.