ABSTRACT

Context-dependent models for language units are essential in high-accuracy speech recognition. However, standard speech recognition frameworks are based on the substitution of lower-level models for higher-level units. Since substitution cannot express context-dependency constraints, actual recognizers use restrictive model-structure assumptions and specialized code for context-dependent models, leading to decreased flexibility and lost opportunities for automatic model optimization. Instead, we propose a recognition framework that builds in the possibility of context dependency from the start by using weighted finite-state transduction rather than substitution. The framework is implemented with a general demand-driven transducer composition algorithm that allows great flexibility in model structure, form of context dependency and network expansion method, while achieving competitive recognition performance.

1. INTRODUCTION

1.1. The Substitution Architecture

In the standard architecture for a speech recognizer, the various levels of linguistic information — language model, word pronunciations, phone models — are represented by a cascade of networks linked by a single operation of substitution. Each network at a given level accepts (emits) sequences of symbols. Each of those symbols is modeled by a network at the level immediately below [1]. For instance, the network in Figure 1a is a (very small) 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 represents a hidden Markov model (HMM) for one phone, with the labels along a complete path specifying legal sequences of acoustic distributions for that phone. The legal sequences of acoustic distributions for an entire word sequence can be found by substituting each word in the language model by its pronunciation network, and then each phone in the resulting phone network by its HMM, and multiplying probabilities appropriately; the weights of the network substituted for a particular unit are of course scaled by the weight given to the unit in the higher-level model.

This architecture allows a wide range of implementations. In principle, a cascade can be expanded in advance into a single network accepting sequences of the lowest-level units (VQ or continuous density labels) by applying recursively the substitution of a symbol by a network. However, if the networks involved are large, full expansion may not be desirable, since the result-
contexts or networks outside the assumed restrictions cannot be used. A less obvious disadvantage is that the specialized algorithms actually hide the underlying simplicity of the more general problem and its algorithmic solution, as we shall see below.

**Restricted expansion:** By expanding the network as a tree rather than as the more usual graph, each partial phonetic hypothesis can be developed as a string, making context-dependent substitution straightforward [15]. A potential disadvantage is that tree expansions contain in general repeated subtrees and may thus be less efficient in time and space than a standard network representation in which a single subnetwork corresponds to the repeated subtrees.

In contrast to the previous algorithms, our approach allows any finite-state model of context to be used in a general class of decoding cascades, without requiring specialized decoders, restricted contexts or tree expansion. Furthermore, it can be used to build a network either statically outside the decoder or dynamically inside the decoder. The approach is based on two fundamental ingredients we describe below: a simple generalization, weighted finite-state transducers, of existing network models, and on-demand composition, a novel technique for network combination.

### 2. WEIGHTED FINITE-STATE TRANSDUCERS

Network models are typically given as HMMs, or equivalently as weighted finite-state automata (WFSA)s. Each transition in a WFSA is labeled with the symbol accepted (emitted) by the transition and with the weight of the transition, which in speech recognition applications represents a probability. Each path from an initial state to a final state assigns to the sequence of path transition labels the accumulated path weight (product of transition probabilities). In essence, a WFSA represents a mapping from symbol sequences to weights [3, 16]. In WFSA terms, substitution corresponds to replacing each arc labeled with unit $u$ by a copy of the WFSA that models $u$ in terms of lower-level units. The correspondence between WFSA s at level $I$ and labels at level $I+1$ is only implicit with this scheme. It can be made explicit by using finite-state transducers. Weighted finite-state transducers (WFSTs) generalize WFSA s by replacing the single arc label by a pair $i:o$ of an input label $i$ and an output label $o$. Each path from an initial state to a final state in a transducer associates the sequence formed by the input labels along the path’s arcs to the sequence formed by their output labels, and assigns to this correspondence the accumulated weight (product of transition probabilities) of the path’s arcs. In essence, a WFST represents a binary relation between symbol sequences and their associated weights [2, 8, 16]. In both WFSA s and WFSTs, the transition label $\epsilon$ represents a null input or output.

The correspondence between two modeling levels $I$ and $I+1$ is readily represented by a weighted transducer that reads sequences of level $I$ units and outputs sequences of level $I+1$ units. For instance, the transducer in Figure 2b represents a (very small) pronunciation lexicon. It gives a mapping from phone sequences to words in the lexicon, in this example “data” and “dew,” with probabilities representing the likelihoods of alternative pronunciations. This transducer encodes the (inverse of the) substitution of phone sequences for words. Since a word pronunciation may be a sequence of several phones, the path corresponding to each pronunciation has $\epsilon$ output labels on all but the word-initial arc.

It is convenient to represent the top level of the cascade also as a transducer. In Figure 2a, a language model is represented by a transducer by simply having the input and output labels of each transition be the same.

The examples in Figure 2 encode (a superset of) the information in the WFSA s of Figure 1a,b as WFSTs. A transducer corresponding to the WFSA in Figure 1c can be constructed in the same way as the transducer in Figure 2b.

The advantages of the transducer representation may not be apparent in these simple examples. Figure 3 demonstrates one advantage — the opportunity for the minimization of networks, which can save both space and time in use. The transducer in Figure 3 is equivalent to the transducer in Figure 2b, but has fewer states and arcs. We call two WFSTs equivalent if they relate the same complete input sequences to the same output sequences with the same accumulated weights.

### 2.1. Context-dependency transducers

A major advantage of transducers for speech recognition is that they generalize naturally the notion of context-independent substitution to the context-dependent case. The transducer of Figure 4a does not correspond to a simple substitution, since it describes the mapping from context-independent phones to context-dependent triphone models, denoted by $\text{phone} / \text{left context} \_ \text{right context}$. Just two hypothetical phones $x$ and $y$ are considered for simplicity. Each state encodes the knowledge of the last and of the next phone. State labels in the figure are pairs $(a, b)$ of the past $a$ and the future $b$, with $\epsilon$ representing the start or end of a phone sequence and $s$ an unspecified future. For instance, it is easy to see that the phone sequence $xyer$ is mapped by the transducer to $x\epsilon y devastating x / y \epsilon$ via the unique state sequence $(\epsilon, s, (x, y), (y, x), (x, \epsilon))$. More generally, when there are $n$ context-independent phones, this construction gives a transducer with $O(n^2)$ states and $O(n^2)$ transitions.
Of course, a practical speech recognition system may not use all and only triphones. For each context specification, an appropriate context-dependency transducer can still be constructed. The main idea, which is embodied in the example above, is to use transducer states to represent appropriate classes of contexts. For instance, if training data is insufficient to create accurate full triphonic models, contexts may be collapsed by ignoring left or right context, or by clustering contexts. Furthermore, automatic training techniques may be used to create suitable context classes [19]. Context classifiers may then depend not only on phones but on additional information such as word boundaries or vowel stress. For such complex context classes, direct construction of the context-dependency transducer is quite delicate. Instead, it is easier to specify the possible contexts as context-dependent regular expressions or rewrite rules and to compile these specifications into finite-state transducers using one of several existing compilation methods [7, 14, 18]. This approach has the advantage that contexts are specified in a perspicuous notation appropriate to the problem, with the conversion into a compact transducer representation being left to a general-purpose compilation algorithm. This is analogous to the compiling a set of pattern-matching regular expressions into an efficient finite-state automaton that implements that match in Unix tools such as grep and lex.

Any context-dependency transducer, whether directly constructed or compiled from patterns or rules, may benefit from determinization. Each state of a deterministic transducer, by definition, has distinct input labels for each transition leaving it. In Figure 4, many states have multiple transitions with the same input label leaving them (for instance, the initial state has three $x$ labeled transitions). Since nondeterminism may increase the work of algorithms such as composition, it is often better to construct a deterministic transducer or apply a determinization algorithm to an existing non-deterministic transducer [10, 13]. The construction in Figure 4 can be altered to produce a deterministic transducer by using the right context phone rather than the center context phone as the input label of a transition.  

3. COMPOSITION

We just showed how the units at different levels of speech modeling can be represented by transducers. It remains to put the units together into a complete modeling network. This is done in two conceptual steps. First, we allow arbitrary sequences of units at each level by adding an $\epsilon$-transition from each final state in the lexicon (Figure 2b) to the initial state (Kleene closure [16]). Second, we compose the various level transducers in the correct order — context-dependency transducer, lexicon transducer and language model — yielding a transducer from distributions to word sequences that is guaranteed to represent all the required cross-word context dependencies.

As previously noted, a transducer represents a binary relation between strings. The composition of two transducers represents their relational composition. In particular, the composition $T = R \circ S$ of two transducers $R$ and $S$ has exactly one path mapping sequence $u$ to sequence $v$ for each pair of paths, the first in $R$ mapping $u$ to some sequence $w$ and the second in $S$ mapping $v$ to $w$. The weight of an output path is the product of the weights of the corresponding pair of input paths [16].

A crucial algorithmic advantage of transducer composition is that it can be computed on demand in a natural way. Our composition algorithm creates on demand just those states and arcs of the composed transducer that are required in a particular decoding run, for instance, those required for paths within the given beam width from the best path. We can thus use the on-demand composition algorithm as a subroutine in a standard Viterbi decoder to combine a language model, a multipronunciation lexicon with corpus-derived pronunciation probabilities, and a context-dependency transducer. Our external interface to composed transducers does not distinguish between on-demand and precomputed compositions, so the decoder algorithm is the same as for an explicit network. In fact, we may also choose to build our networks off-line when appropriate for the given task. In general, expanding the networks on demand may save space, while expanding them off-line may save time (during recognition). In the next section, we look more closely at this issue.

Transducer composition is a generalization of the standard state-pair construction for finite automata intersection [6]. The composition $T$ of transducers $R$ and $S$ is a transducer whose states are pairs of a state from $R$ and a state from $S$, and that satisfies the following conditions: (1) the initial state of the composition $T$ is the pair of the initial states of $R$ and $S$; (2) a state is final in $T$ iff it is a pair of final states from $R$ and $S$, and (3) there is a transition in $T$ from $(q, q')$ to $(r, r')$ iff there are compatible transitions $t$ from $q$ to $r$ and from $q'$ to $r'$, where compatible means the output label of $t$ matches the input label of $t'$. When they match, the resulting $T$ transition takes its input label from $t$, its output label from $t'$, and its weight is the product of the weights of $t$ and $t'$.

When there are $\epsilon$ labels present in the output of $R$ or the input of $S$, the set of transitions for $T$ given above must be extended. The basic extension is to allow a transition with $\epsilon$ output (input) in $R$ ($S$) to match an implicit $\epsilon$ self-loop in $S$ ($R$). However, when there are $\epsilon$ transitions on both transducers, care must be taken...
to avoid multiple redundant matches. The interested reader is referred to [16, 12] for the algorithm for this general case and a proof of its correctness.

Operations (1) and (2) can be implemented trivially. Operation (3) requires finding compatible transitions leaving a pair of states in \( R \) and \( S \); this can be done, for example, by pre-sorting labels and merging as needed.

The above three operations can be computed on demand since they depend only on local properties of the component transducers. In fact, we use precisely these three operations as the interface between the decoder and the modeling network. Thus, the decoder can not distinguish between explicit and on-demand networks. Further, other kinds of on-demand combinations are also easily implemented in this way such as the dynamic union of networks.

There is an important optimization for the on-demand case. As described so far, every time a composition state is expanded, the work done to build the composition is repeated. Since a decoder may often expand the same state many times, it can instead be useful to save and reuse the results of previously expanded states. There is variety of such caching disciplines that can be used: for example, (1) save all previously expanded states (for the given utterance), or (2) save recently expanded states, but free less recently used ones. For the results in the next section, we used caching method (1).

### 4. EXPANSION METHODS

To evaluate the tradeoffs in space and time between expanding networks on demand during recognition and off-line network expansion, we examined several recognition tasks. In particular, we evaluated a recognizer for the DARPA ATIS task using full-cross-word triphonic models, a 1533 word vocabulary, and a class-based trigram grammar [17]. Table 1 shows the sizes of the context-dependency, lexicon and language model transducers for this task.

For the same recognition accuracy as with a static, fully expanded network, on-demand composition expands on-average less than 3% of the total number of arcs while adding no significant runtime overhead (less than 2% of total runtime). We found proportionally similar space reductions on the 991-word DARPA RM task with finite-state and word-pair grammars and on a 20,000 DARPA NAB task with bigram and trigram grammars. However, with very large vocabularies such as in NAB, we have found the runtime cost of the on-demand composition becomes significant (approximately 25% of the total runtime) unless weighted determinization is applied to the composition of the lexicon and the grammar. Once this optimization is applied, the overhead of on-demand composition is significantly reduced and so is the size of the fully-expanded network. This more recent work is described elsewhere [13].

### 5. CONCLUSION

While we focused here primarily on triphonic contexts and finite-state language models, our approach applies much more generally. Any bounded-memory context-dependency can be modeled as a transducer, for instance any context model based on decision trees. Furthermore, any language model-class closed under composition with finite-state transducers, such as (probabilistic) context-free grammars, is compatible with our on-demand composition method.

### Acknowledgments

We thank Emerald Chung and Yi-Min Wang for implementing the first Viterbi decoder using our on-demand composition algorithm, which enabled the experiments described in this paper. We also thank Andrej Ljolje for proving acoustic models, Giuseppe Riccardi for providing the ATIS language model and Don Hindle for the NAB language model.

### 6. REFERENCES