Algorithmic Aspects in Speech Recognition: An Introduction

Adam L. Buchsbaum
AT&T Labs, Florham Park NJ, U.S.A.
and
Raffaele Giancarlo
Universitá di Palermo, Palermo, Italy

Association for Computing Machinery, Inc., 1515 Broadway, New York, NY 10036, USA, Tel: (212) 869-7440

Speech recognition is an area with a considerable literature, but there is little discussion of the topic within the computer science algorithms literature. Many computer scientists, however, are interested in the computational problems of speech recognition. This paper presents the field of speech recognition and describes some of its major open problems from an algorithmic viewpoint. Our goal is to stimulate the interest of algorithm designers and experimenters to investigate the algorithmic problems of effective automatic speech recognition.

Categories and Subject Descriptors: I.2.7 [**Natural Language Processing**]: Speech Recognition and Synthesis—*Algorithms*

General Terms: Algorithms, Experimentation, Theory

Additional Key Words and Phrases: automata theory, graph searching

1. INTRODUCTION

Automatic recognition of human speech by computers has been a topic of research for more than forty years (paraphrasing Rabiner and Juang [1993]). At its core, speech recognition seems to require searching extremely large, weighted spaces, and so naturally leads to algorithmic problems. Furthermore, speech recognition tasks algorithm designers to devise solutions that are not only asymptotically efficient—to

The work of Giancarlo was partially done while he was a Member of Technical Staff at AT&T Bell Laboratories and was supported thereafter by AT&T Labs.
Authors' addresses: Adam L. Buchsbaum, AT&T Labs, 180 Park Ave., Florham Park NJ 07932, U.S.A., alb@research.att.com; Raffaele Giancarlo, Dipartimento di Matematica ed Applicazioni, Universitá di Palermo, Via Archirafi 34, 90123 Palermo, Italy, raffaele@altair.math.unipa.it.

handle very large instances—but also practically efficient—to run in real time. We find little if any coverage of speech recognition in the algorithms literature, however. A sizable speech recognition literature does exist, but it has developed in a separate community with its own terminology. As a result, algorithm designers interested in the problems associated with speech recognition may feel uncomfortable. Such potential researchers, however, benefit from a lack of preconceptions as to how effective (that is, accurate, robust, fast, etc.) speech recognition should be realized.

We aim in this paper to summarize speech recognition, distill some of the current major problems facing the speech recognition community, and present them in terms familiar to algorithm designers. We describe our own understanding of speech recognition and its associated algorithmic problems. We do not try to solve the problems that we present in this paper; rather we concentrate on describing them in such a way that interested computer scientists might consider them.

We believe that speech recognition is well suited to exploration and experimentation by algorithm theorists and designers. The general problem areas that are involved—in particular, graph searching and automata manipulation—are well known to and have been extensively studied by algorithms experts. While some very tight theoretical bounds and even very good practical implementations for some of the specific problems (e.g., shortest path finding and finite state automata minimization) are already well known, the manifestations of these problems as they arise in speech recognition are so large as to defy straightforward solutions. The result is that most of the progress in speech recognition to date is due to clever heuristic methods that solve special cases of the general problems. Good characterizations of these special cases, as well as theoretical studies of their solutions, are still lacking, however. There is much room for both experiments in characterizing various special cases of general problems and also for theoretical analysis to provide more than empirical evidence that deployed algorithms will perform in guaranteed manners. Furthermore, practical implementations of any algorithms are critical to the deployment of speech recognition technology. The interested algorithm expert, therefore, has a wide range of stimulating problems from which to choose, the solutions of which are not only of theoretical but also of practical importance.

Although this paper is not a formal survey, we do introduce the dominant speech recognition formalisms to help algorithm designers understand that literature. While we want to consider speech recognition from as general a perspective as possible, for sake of clarity as well as space we have chosen to present the topic from the dominant viewpoint found in the literature over the last decade or so—that of maximum likelihood as the paradigm for speech recognition. We cannot stress enough that while reading this paper, one should not assume that this is in fact the correct way to address speech recognition. While the maximum-likelihood paradigm has found recent success in some recognition tasks, it is not clear that this model will be the best one over the long term.

In Section 2, we informally introduce some of the notions behind speech recognition. In Section 3, we formalize these ideas and state mathematically the goal of speech recognition. We continue in Section 4 by introducing hidden Markov models and Markov sources for modeling the various components of a speech recognition system. In Section 5, we outline the Viterbi algorithm, which solves the main equation presented in Section 3 using hidden Markov models. In Section 6, we present

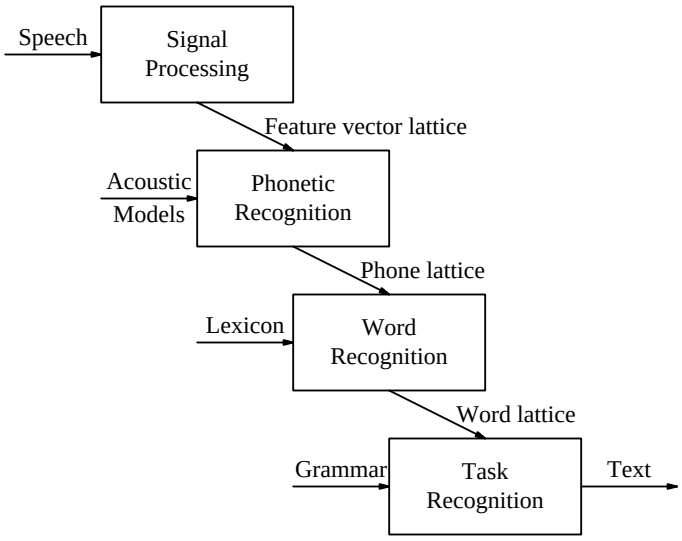

Fig. 1. Block diagram of a speech recognizer. Input speech is digitized into a sequence of feature vectors. An acoustic-phonetic recognizer transforms the feature vectors into a time-sequenced lattice of phones. A word recognition module transforms the phone lattice into a word lattice, with the help of a lexicon. Finally, in the case of continuous or connected word recognition, a grammar is applied to pick the most likely sequence of words from the word lattice.

the A^* algorithm, originally developed by the artificial intelligence community, and a related general optimization paradigm for searching large, weighted graphs and discuss how these can be used in speech recognition. In Section 7, we describe another approach to speech recognition, based on finite-state transducers. In Section 8, we discuss determinization and minimization of weighted lattices and automata, computational problems that are common to the two approaches (hidden Markov models and finite-state transducers). Finally, we conclude with some discussion in Section 9. Throughout the paper, we introduce relevant research areas that we think will be interesting to algorithm experts as well as critical to the advancement of automatic speech recognition. We summarize these in Appendix A. Appendix B gives pointers to some relevant sources of code, data, etc. Appendix C provides a glossary of abbreviations used throughout the paper.

2. AN INFORMAL VIEW OF SPEECH RECOGNITION

Speech recognition is the process of reconstructing the text of a spoken sentence from the continuous acoustic signal induced by the associated utterance. A speech recognizer usually operates in phases, as shown in Figure 1; Pereira and Riley [1997] refer to this pipeline as the recognition *cascade*. By means of signal processing, the acoustic waveform is first transformed into a sequence of discrete observations over some unbounded alphabet $\mathcal{F}$. We call the sequence of discrete observations the *observation* or *input sequence*. Its symbols, referred to as *feature vectors*, are designed to preserve relevant acoustic information from the original signal; in the most

general setting, the feature vectors will also have a probability distribution associated with them. (It is also possible to consider continuous, rather than discrete, observations, and we discuss this issue briefly in Section 4.1.) We have chosen to focus on the later computational aspects of processing this discrete sequence. While the signal processing and acoustics issues are equally important, they are beyond the scope of this paper. Rabiner and Juang [1993] give an extensive treatment of how the transformation from the acoustic signal to a sequence of feature vectors is obtained.

Because different users, or the same user at different times, may utter the same sentence in different ways, the recognition process is stochastic. At a very high level, we can divide the speech recognition area into two branches: *isolated word recognition* (IWR) and *continuous speech recognition* (CSR).

In IWR, the recognizer takes as input the observation sequence of one word at a time (spoken in isolation and belonging to a fixed dictionary) and for each input word outputs, with high probability, the word that has been spoken. The two main algorithmic components are the *lexicon* and the *search algorithm*. For now, we discuss these components informally. The *lexicon* contains the typical pronunciations of each word in the dictionary. An example of a lexicon for English is the set of phonetic transcriptions of the words in an English dictionary. From the phonetic transcriptions, one can obtain *canonical acoustic models* for the words in the dictionary. These acoustic models can be considered to be Markov sources over the alphabet $\mathcal{F}$. The search algorithm compares the input sequence to the canonical acoustic model for each word in the lexicon. It outputs the word that maximizes a given objective function. In theory, the objective function is the *likelihood of a word, given the observation sequence.* In practice, however, the computation of the objective function is usually approximated using heuristics, the effectiveness of which are established experimentally; i.e., no theoretical quantification is available on the disparity between the heuristic solution and the optimal solution. Such an approximation is justified by the need for fast responses in the presence of large dictionaries and lexicons. As we will see, several aspects related to the representation of the lexicon influence the search heuristics.

In CSR, the recognizer takes as input the observation sequence corresponding to a spoken sentence and outputs, with high probability, that sentence. The three algorithmic components are the *lexicon*, the *language model* or *grammar*, and the *search algorithm*. Again, for now, we discuss these components informally. The lexicon is exactly as in IWR, whereas the language model gives a stochastic description of the language. That is, the language model gives a syntactic description of the language, and, in addition, it also provides a (possibly probabilistic) description of which specific words can follow another word or group of words; e.g., which specific nouns can follow a specific verb. The lexicon is obtained as in the case of IWR, whereas the language model is built using linguistic as well as task-specific knowledge. The search algorithm uses the language model and, for each word, the acoustic models derived from the lexicon, to "match" the input sequence, trying to find a grammatically correct sentence that maximizes a given objective function. As an objective function, here again one would like to use the *likelihood of a sentence given the observation sequence.* Even for small languages, however, this is not possible or computationally feasible. (The reasons will be sketched in the

technical discussion.) Therefore, the search algorithms use some reasonable approximations to the likelihood function, and, even within such approximate search schemes, heuristics are used to speed the process.

At first, IWR seems to be a special case of CSR. Therefore, from the algorithmic design point of view, one could think of devising effective search techniques for the special case, hoping to extend them to the more general case. Unfortunately, this approach does not seem viable, because the nature of the search procedures for IWR differs from the nature of the corresponding procedures for CSR. We now briefly address the disparities in high-level terms, along with an example.

In IWR, all the needed acoustic information modeling the words in the dictionary is available to the search procedure. That is, for each word in the dictionary there is a canonical acoustic model of that word in the lexicon. Thus, the search problem becomes one of pattern recognition, in the sense that the search procedure tries to find the canonical model that best matches the input observations.

In CSR, the acoustic information modeling the sentences in the language is given only partially and implicitly in terms of rules. That is, there is no canonical acoustic model for each sentence in the language. The only canonical acoustic models that are available are those in the lexicon that correspond to the words in the language. The search procedure must therefore assemble a sequence of canonical acoustic models that best match the observation sequence, guided by the rules of the language. Such an assembly is complicated by the phenomenon of inter-word dependencies: when we utter a sentence, the sounds associated with one word influence the sounds associated with the next word, via coarticulation effects of each phone on successive phones. (For example, consider the utterances, "How to recognize speech," and, "How to wreck a nice beach.") Since these inter-word dependencies are not completely modeled and described by the lexicon and the language model (otherwise, we would have a canonical acoustic model for each sentence in the language), the search procedure for CSR faces the additional difficult task of determining, using incomplete information, where a word begins and ends. In fact, for a given observation sequence, the search procedure usually postulates many different word boundaries, which may in turn lead to exponentially many ways of decoding the input sequence into a sequence of words.

Figures 2 and 3 demonstrate the differences between the IWR and CSR tasks. Each figure displays, top-to-bottom, an acoustic waveform, a spectrogram, and labelings for the sentence, "Show me a flight to Boston." In Figure 2, the words are spoken in isolation; in Figure 3, the sentence is spoken fluently. The acoustic waveform displays signal amplitude as a function of time. The spectrogram displays energy as a function of time and frequency: darker bands represent more energy at a given frequency and time.[1] The acoustic waveform and spectrogram were hand-segmented into phones. The top set of labels shows the ending time of each phone, and the bottom set of labels shows the ending time of each word. (The phones are transcribed in ARPABET [Shoup 1980].) Notice that the isolated-word case contains distinct, easy-to-detect boundaries, without coarticulation effects on boundary phones. In the continuous-speech case, however, the word boundaries

[1] A black-and-white spectrogram is not, in fact, very useful, except to show where various acoustic features begin and end, which is our purpose.

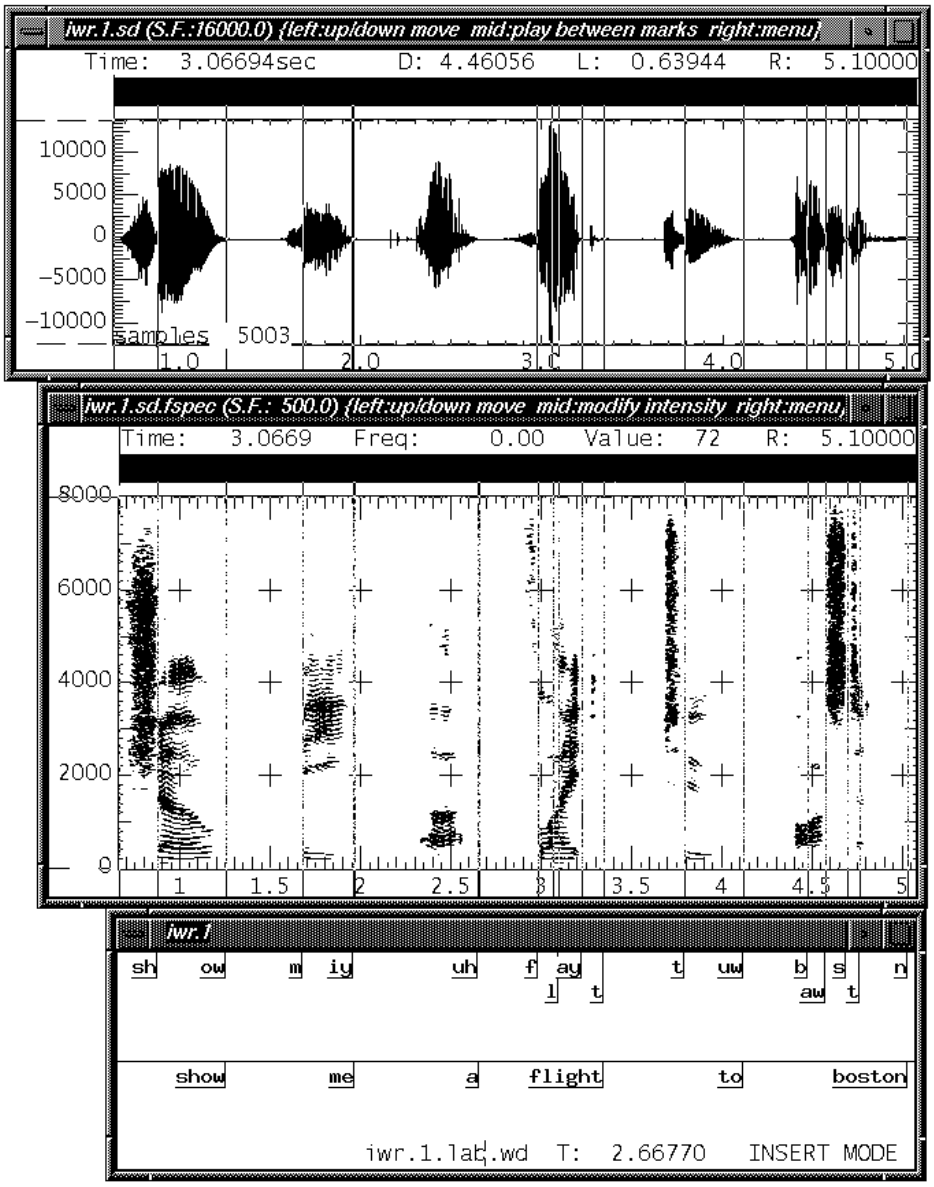

Fig. 2. Acoustic waveform, spectrogram, and labelings for the sentence, "Show me a flight to Boston," with each word spoken in isolation.

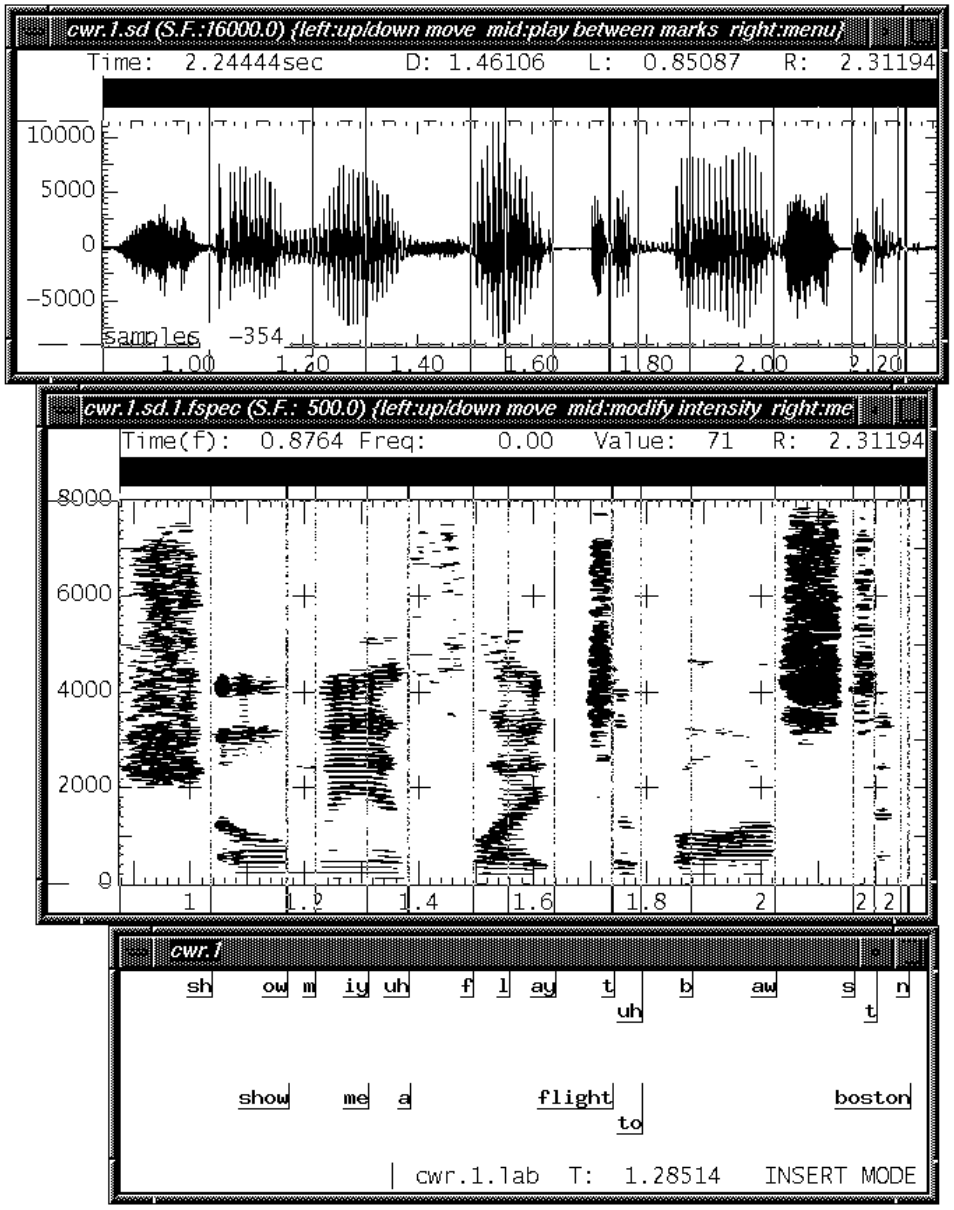

Fig. 3. Acoustic waveform, spectrogram, and labelings for the sentence, "Show me a flight to Boston," spoken fluently.

are not clear, and phones at word-boundaries display coarticulation effects. In an extreme case, the /t/ of "flight" and the /t/ of "to" have elided. Thus, in CSR, not only does one face the problem of finding word boundaries, but also, due to coarticulation effects, the acoustic models for each word in the lexicon do not necessarily reflect the actual utterance of the word in the sentence.

3. FUNDAMENTAL EQUATIONS FOR SPEECH RECOGNITION

In this section, we discuss two major paradigms for speech recognition: the *stochastic approach*—in particular, *maximum likelihood*—and the *template-based approach*. While the remainder of this paper concentrates on the former, due to its dominance in current technology, we briefly discuss the latter to demonstrate alternatives.

3.1 The Stochastic Approach

Let $\mathcal{L}$ denote the language composed of the set of sentences that the system has to recognize, and let D denote the associated dictionary. The task of the speech recognizer is the following. Given an observation sequence X corresponding to some unknown sentence W, output the sentence $\hat{W}$ that, according to some criterion, best accounts for the observation sequence. When the dictionary and/or the language are large, the criterion that has become dominant in making this choice is *maximum likelihood* [Bahl et al. 1983; Jelinek et al. 1992], as follows.

Assume that for each sentence $W = w_1 \cdots w_g \in \mathcal{L}$, we know the probability $\Pr(W)$ of uttering W. We ignore for now how to compute this quantity. Let $\Pr(W|X)$ be the probability that the sentence W was spoken, given that the observation sequence X has been observed. Then, the recognizer should pick the sentence $\hat{W}$ such that

$$\Pr(\hat{W}) = \max_{W} \{\Pr(W|X)\}. \tag{1}$$

Using Bayes' formula, the right hand side of Equation 1 can be rewritten using

$$\Pr(W|X) = \frac{\Pr(X|W)\Pr(W)}{\Pr(X)}. \tag{2}$$

Since the maximization in Equation 1 is over a fixed X, we have from Equations 1–2 a reduction of the problem to determining a sentence $\hat{W}$ such that[2]

$$\hat{W} = \operatorname*{argmax}_{W} \{\Pr(W)\Pr(X|W)\}. \tag{3}$$

Given a generic probability distribution Pr, let $Cs = -\log \Pr$. For instance, $Cs(W) = -\log \Pr(W)$, and $Cs(X|W) = -\log \Pr(X|W)$. The first term is the *cost* of generating W and the second is the *cost* of "matching" the observation sequence X with the sentence W. The term "cost" does not refer to computational complexity but rather to an alternative to probabilities as a measure of the likelihood of an event. Probabilities are referred to as *scores* in the speech recognition literature.

[2] $\operatorname{argmax}_x \{f(x)\} = \hat{x}$ such that $f(\hat{x}) = \max_x \{f(x)\}$. Similarly define argmin_x.

We use costs in order to describe later search algorithms in terms of shortest paths. With this convention, Equation 3 can be rewritten as

$$\hat{W} = \underset{W}{\operatorname{argmin}} \{Cs(W) + Cs(X|W)\} . \tag{4}$$

Since we have decided to base the design of speech recognition systems on the computation of Equations 3–4, we need to develop tools to determine, for a given language, $\Pr(W)$ and $\Pr(X|W)$. Such tools belong to the realm of *language modeling* and *acoustic modeling*, respectively, and we present them in the next section.

3.2 Template-Based Approaches

As we have said, the maximum-likelihood criterion has become dominant for the design of speech recognition systems in which the dictionary and/or language model are large. For small dictionaries and mainly for IWR, the *template-based approach* has been successful. While it is beyond the scope of this paper to provide details on this approach, we briefly outline it here to demonstrate that alternatives to the maximum likelihood paradigm exist. Rabiner and Juang [1993, Ch. 4] give a thorough tutorial on template-based methods, and Waibel and Lee [1990, Ch. 4] give examples of practical applications of this approach.

As discussed in Section 2, consider the output from the signal processing module of a speech recognizer to be a sequence of feature vectors. In the template-based approach to speech recognition, one first builds a collection of *reference templates*, each itself a sequence of feature vectors that represents a unit (usually a whole word) of speech to be recognized. Then, the feature vector corresponding to the current utterance is compared with each reference vector in turn, via some distance measure. Various distance measures (e.g., log spectral distance, cepstral distance, weighted cepstral distance, and likelihood distortions) have been the subject of research and application. Additionally, methods for resolving the difference between the number of feature vectors in the input and those of the individual reference templates have been studied.

The template-based approach has produced favorable results for small-dictionary applications, again mainly for IWR. In particular, the modeling of large utterances (words instead of phones) avoids the errors induced by segmenting inputs into smaller acoustic units. On the other hand, as the units to be modeled grow in size, the number of such units explodes. Comparing an input against all reference templates then becomes too time-consuming; even collecting enough reference templates to build a complete system becomes impractical once the vocabulary exceeds a few hundred units.

Therefore, the template-based approach does not seem extensible to IWR and CSR when the dictionary and language model are large. In these cases, the stochastic approach based on maximum likelihood is applied. A very challenging long-term research goal is to establish whether stochastic approaches other than the one summarized by Equation 3 can underly effective speech recognition systems.

4. MODELING TOOLS FOR SPEECH RECOGNITION

In this section we introduce the main tools that are used for acoustic and language modeling in speech recognition systems. They are based on *hidden Markov models*

(*HMM*s) and *Markov sources* (*MS*s). Intuitively, these are devices for modeling doubly stochastic processes. *States* tend to represent some physical phenomenon (e.g., moment in time, position in space); *actions* or *outputs* occur at states and model the outcome of being in a particular state. As we discuss the formal definitions of *HMM*s and *MS*s, it will be useful to have an example in mind.

Example. Consider a magician who has three hats (red, blue, and yellow) and "randomly" chooses an object (a hare, a guinea pig, or a parrot) out of one hat during a show. From show to show, he chooses first among the hats (to vary the performance for repeat observers), reaches into the hat to pull out an animal, and then replaces the animal. We can use a *HMM* to model the hat trick, as we shall see below.

4.1 Hidden Markov Models

Here we formally define hidden Markov models and the problems related to them whose solutions are essential for speech recognition. Rabiner [1990] provides a thorough tutorial.

Let Σ be an alphabet of M symbols. A *hidden Markov model* is a quintuple $\lambda = (N, M, A, B, \pi)$, where

—N is the number of states, denoted by the integers $1, \ldots, N$. In the magic example, $N = 3$, and the states correspond to which hat (red, blue, or yellow) the magician is about to use.

—M is the number of symbols that each state can output or recognize. $M = 3$ in the magic example, as each symbol corresponds to an animal (hare, guinea pig, or parrot) that can be pulled out of a hat.

—A is an $N \times N$ state transition matrix such that a_{ij} is the probability of moving from state i to state j, $1 \leq i, j \leq N$. We must have that the sum $\sum_j a_{ij} = 1, \forall i$. For our example, the transition matrix represents the probability of using a particular hat in the next performance, given the hat that was used in the current one; e.g., a_{11} (rsp., a_{12}, a_{13}) is the probability of using hat 1 (rsp., 2, 3) next time, given that hat 1 was used currently.

—B is an observation probability distribution such that $b_j(\sigma)$ is the probability of recognizing or generating the symbol σ when in state j. It must be that $\sum_{\sigma \in \Sigma} b_j(\sigma) = 1, \forall j$. In our example, b_j represents the probability of pulling a particular animal out of hat j.

—π is the initial state probability distribution such that π_i is the probability of being in state i at time 1. It must be that $\sum_i \pi_i = 1$. In our example, π reflects the probability of using a particular hat in the first show.

The state transition matrix induces a directed graph, with nodes representing states, and arcs between states labeled with the corresponding transition probabilities. Figure 4 shows the graph for the magic example. For the purposes of this example, we label the states R, B, and Y (for the colors of the hats), and the outputs H, G, and P (for the animals). Table 1 gives the transition and output probabilities. (Assume that π gives an equal probability of starting with any hat.)

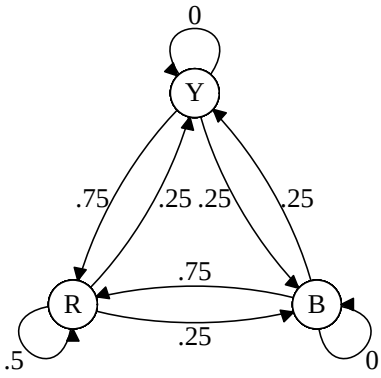

Fig. 4. Graph induced by the state transition matrix for the magic example.

Transition Probabilities

	R	B	Y
R	.5	.25	.25
B	.75	0	.25
Y	.75	.25	0

Output Probabilities

	H	G	P
R	.25	.5	.25
B	.2	.6	.2
Y	.1	.8	.1

Table 1. Transition and output probabilities for the magic example. In the left table, each row gives the probabilities of choosing the next hat based on the given current hat. In the right table, each row gives the probabilities of choosing a certain animal out of a given hat.

The transition probabilities suggest that the magician favors the red hat, and the output probabilities show that he prefers the guinea pig.

The term "hidden" comes from the fact that the states of the Markov model are not observable. In fact, the number of states, output symbols, as well as the remaining parameters of the hidden Markov model are estimated by observing the phenomenon that the unknown Markov chain describes. Rabiner and Juang [1993] overview such estimation procedures. In the magic example, it is as if the hats were not colored (i.e., not distinguishable to the observer) and the magician picks one before the show. In this situation, over time the observer of many shows sees only a sequence of animals produced by the magician; he has no idea which hat is used during which show, and in fact he has no idea how many hats exist at all.

It is also possible to model continuous rather than discrete observations, without somehow quantizing the input. In this case, B is a collection of continuous probability density functions such that for any j, $\int b_j(\sigma)\, d\sigma = 1$. The b_j's must be restricted to allow consistent reestimation, and typically they are expressed as finite mixtures of, e.g., Gaussian distributions. For the purpose of illustrating the search problems in later sections, we will concentrate on discrete observations.

4.1.1 *Hidden Markov Models as Generators.* The *HMM* just defined can be used as *generator* of sequences of Σ^*. Let $X = x_1 \cdots x_T \in \Sigma^*$. It can be generated by a sequence of states $Q = q_1 \cdots q_T$ as follows.

(1) Set $i \leftarrow 1$, and choose the initial state q_i according to the initial state probability distribution π.

(2) If in state q_i (having generated $x_1 \cdots x_{i-1}$), output x_i according to the probability function b_{q_i}.
(3) If $i < T$, then set $i \leftarrow i + 1$, enter state q_i according to probabilities $A[q_{i-1}, 1 : N]$, and then repeat at step (2); otherwise, stop.

In the magic example, the magician starts with hat R, B, or Y with probability $1/3$ each. If he has chosen the red hat, then he picks the hare with probability .25, the guinea pig with probability .5, and the parrot with probability .25; then he next uses the red hat with probability .5 and the blue and yellow hats each with probability .25. If instead he starts with the blue hat, then he picks the hare (rsp., guinea pig, parrot) with probability .2 (rsp., .6, .2) and next uses the red (rsp., blue, yellow) hat with probability .75 (rsp., 0, .25). And if he starts with the yellow hat, then he picks the hare (rsp., guinea pig, parrot) with probability .1 (rsp., .8, .1) and next uses the red (rsp., blue, yellow) hat with probability .75 (rsp., .25, 0). He continues picking animals and choosing hats in this way, and over time, the observer sees a succession of animals being picked out of hats. Correspondingly, the Markov model generates a sequence of animals.

4.1.2 *Hidden Markov Models as Matchers.* A *HMM* can also be used as a probabilistic *matcher* of sequences of Σ^*, in the sense that it gives a measure, in terms of probability mass, of how well the *HMM* λ matches or observes X:

$$\Pr(X|\lambda) = \prod_{t=1}^{T} \sum_{i=1}^{N} \Pr(q_t = i) b_i(x_t) \tag{5}$$

where

$$\Pr(q_t = j) = \begin{cases} \pi_j & t = 1 \\ \sum_{i=1}^{N} \Pr(q_{t-1} = i) a_{ij} & t > 1 \end{cases}.$$

HMM λ induces an unbounded, multipartite directed graph as follows. There are N rows, corresponding to the N states of λ, and for all $t \geq 1$, columns t and $t + 1$ form a complete, directed bipartite graph, with arcs directed from vertices in column t to vertices in column $t + 1$. (This graph is commonly referred to as a *trellis*; see, e.g., Soong and Huang [1991].) In this way the match consists of superimposing X along all paths, starting at vertices in column 1, of length T in the trellis. For a given vertex i in column t on a given path, the measure of how well it matches symbol x_t is composed of two parts: the probability of being in that state ($Pr(q_t = i)$) and the probability that the state outputs x_t ($b_i(x_t)$).

In the magic example, we can calculate how likely it is that the magician first picks the parrot, then the guinea pig, then the hare. The probability of picking the parrot first is about .183 (1/3 chance of using the red hat times .25 chance of picking the parrot from the red hat, and so on). From π and the transition probability matrix, the probability of using the red (rsp., blue, yellow) hat second is about .667 (rsp., .167, .167); thus the probability of picking the guinea pig second is $.667 \times .5 + .167 \times .6 + .167 \times .8 \approx .567$. The probability of using the red (rsp., blue, yellow) hat third (and last) is about .581 (rsp., .209, .209); thus the probability of picking the hare last is $.581 \times .25 + .209 \times .2 + .209 \times .1 \approx .208$. Therefore, the probability that the magician picks first the parrot, then the guinea pig, and then the hare is approximately .022.

4.1.3 *Problems for Hidden Markov Models.* Problems 4.1 and 4.2 are two interesting problems for *HMM*s that are strictly related to the search phase of speech recognition.

PROBLEM 4.1. *Given an observation sequence $X = x_1 \cdots x_T$, compute the probability* $\Pr(X|\lambda)$ *of the model λ generating (or matching) the sequence X.*

This problem can be solved in $O(NT \times \delta_{max})$ time, where δ_{max} is the maximum in-degree of any state in the *HMM*, using the *forward procedure* [Baum and Eagon 1967; Baum and Sell 1968], which solves Equation 5. The forward procedure generalizes the computation that calculated the probability of the magician pulling first a parrot, then a guinea pig, then a hare out of hats. It is a dynamic programming algorithm that maintains a variable $\alpha_t(i)$, defined as

$$\alpha_t(i) = \Pr(x_1 \cdots x_t, q_t = i|\lambda);$$

i.e., the probability that at time t, we have observed the partial sequence $x_1 \cdots x_t$ and ended in state i. The procedure has three phases.

(1) Initialization.

$$\alpha_1(i) = \pi_i b_i(x_1), \quad 1 \le i \le N.$$

(2) Induction.

$$\alpha_{t+1}(j) = \left(\sum_{i=1}^{N} \alpha_t(i) a_{ij} \right) b_j(x_{t+1}), \qquad \begin{matrix} 1 \le t \le T-1 \\ 1 \le j \le N \end{matrix}.$$

(3) Termination.

$$\Pr(X|\lambda) = \sum_{i=1}^{N} \alpha_T(i).$$

RESEARCH AREA 4.1. *The forward procedure has the following direct application: Given an utterance and a set of* HMM*s that model the words in a lexicon, find the word that best matches the utterance. This forms a rudimentary isolated word recognizer. The speed of the forward procedure (or any algorithm computing the same result) bounds the size of the lexicon that can be employed. Faster algorithms to compute the matching probability of a* HMM *therefore will find immediate applications in isolated word recognizers. A particular direction for experimentation is to determine how the topology of the graph underlying a* HMM *affects the performance of the forward procedure (or subsequent similar algorithms).*

PROBLEM 4.2. *Compute the* optimal *state sequence $Q = (q_1, \ldots, q_T)$ through λ that matches X.*

The meaning of *optimal* is situation dependent. The most widely used criterion is to find the single best state sequence Q that generates X, i.e., to maximize $\Pr(Q|X, \lambda)$ or, equivalently, $\Pr(Q, X|\lambda)$. This computation is usually performed using the *Viterbi* recurrence relation [Viterbi 1967], which we discuss in Section 5. Briefly, though, the Viterbi algorithm computes

(1) the probability along the highest probability path that accounts for the first t observations and ends in state i,

$$\beta_t(i) = \max_{q_1,\ldots,q_{t-1}} \Pr(q_1,\ldots,q_{t-1},q_t = i, x_1 \cdots x_t|\lambda),$$

and

(2) the state at time $t-1$ that led to state i at time t along that path, denoted by $\gamma_t(i)$.

The computation is as follows.

(1) Initialization.

$$\begin{array}{l} \beta_1(i) = \pi_i b_i(x_1) \\ \gamma_1(i) = 0 \end{array}, \quad 1 \le i \le N$$

(2) Induction.

$$\begin{array}{ll} \beta_t(j) = \max_{1\le i\le N}\{\beta_{t-1}(i)a_{ij}\}b_j(x_t) & 2 \le t \le T \\ \gamma_t(j) = \mathrm{argmax}_{1\le i\le N}\{\beta_{t-1}(i)a_{ij}\} & 1 \le j \le N \end{array},$$

(3) Termination.

$$\begin{aligned} P &= \max_{1\le i\le N}\{\beta_T(i)\} \\ q_T &= \operatorname*{argmax}_{1\le i\le N}\{\beta_T(i)\} \end{aligned}$$

(4) Backtracking.

$$q_t = \gamma_{t+1}(q_{t+1}), \quad t = T-1,\ldots,1$$

The Viterbi algorithm, however, may become computationally intensive for models in which the underlying graph is large. For such models, there is a great number of algorithms that use heuristic approaches to approximate $\Pr(Q, X|\lambda)$. The main part of this paper is devoted to the presentation of some of the ideas underlying such algorithms.

4.1.4 *Application to Speech Recognition.* *HMM*s have a natural application to speech recognition at most stages in the pipeline of Figure 1. Each of the post signal-processing modules in that pipeline takes output from the previous module as well as precomputed data as input and produces output for the next module (or the final answer). The precomputed data can easily be viewed as a *HMM*.

For example, consider the lexicon. This model is used to transform the phone lattice into a word lattice by representing possible pronunciations of words (along with stochastic measures of the likelihoods of individual pronunciations). In a *HMM* corresponding to the lexicon, the states naturally represent discrete instances during an utterance, and the outputs naturally represent phones uttered at the respective instances. The *HMM* can then be used to generate (or match) words in terms of phones.

In the rest of this section and Sections 5 and 6, we give more details on the application of *HMM*s to speech recognition.

4.2 Markov Sources

We now define *Markov sources* (MS), following the notation of Bahl, Jelinek, and Mercer [1983]. Let V be a set of states, E be a set of transitions between states, and $\hat{\Sigma} = \Sigma \cup \{\phi\}$ be an alphabet, where ϕ denotes the null symbol. We assume that two elements of V, s_I and s_F, are distinguished as the initial and final state, respectively. The structure of a *MS* is a one-to-one mapping M from E to $V \times \hat{\Sigma} \times V$. If $M(t) = (\ell, a, r)$, then we refer to ℓ as the predecessor state of t, a as the output symbol of t, and r as the successor state of t. Each transition t has a probability distribution z associated with it such that (1) $z_s(t) = 0$ if and only if s is not a predecessor state of t, and (2) $\sum_t z_s(t) = 1$, for all $s \in V$. A *MS* thus corresponds to a directed, labeled graph with some arcs labeled ϕ. The latter are *null* transitions and produce no output. With these conventions, a *MS* is yet another recognition and/or generation device for strings in Σ^*.

As can be easily seen, *HMM*s and *MS*s are very similar: *HMM*s generate output at the states, whereas *MS*s generate outputs during transitions between states. Furthermore, a *MS* can represent any process that can be modeled by a *HMM*: there is a corresponding state for each state of the *HMM*, and a transition (i, σ, j) with $z_i(i, \sigma, j) = a_{ij} b_i(\sigma)$, for all $1 \leq i, j \leq N$ and $\sigma \in \Sigma$. It is not necessarily the case, however, that there is an equivalent *HMM* for a given *MS*. The reason is that *MS*s allow for the output symbol and transition probability distributions of a given state to be interdependent, whereas the output symbol probability distribution at any state in a *HMM* is independent of the transition probability for that state. For example, a three-state *MS* might allow symbols 4 and 5 to be output on transitions from state 1 to state 2 but only symbol 4 to be output on transition from state 1 to state 3; no *HMM* can model the same phenomenon. We could extend the definition of *HMM*s to include null symbols and then allow such interdependencies by the introduction of intermediate states. This approach, however, affects the time-synchronous behavior of *HMM*s as sequence generators/matchers, and whether or not the two machines (the original *MS* and the induced *HMM*) are equivalent becomes application dependent.

We introduce both *HMM*s and *MS*s, because in most speech recognition systems, they are used to model different levels of abstraction, as we will see in the next section. The only notable exceptions are the systems built at IBM [Bahl et al. 1983].

4.3 Acoustic Word Models via Acoustic Phone Models

In this section we describe a general framework in which one can obtain acoustic models for words for use in a speech recognition system.

From the phonetic point of view, *phonemes* are the smallest units of speech that distinguish the sound of one word from that of another. For instance, in English, the /b/ in "big" and the /p/ in "pig" represent two different phonemes. Whether to refer to the phonetic units here as "phonemes" or simply "phones" is a matter of debate that is beyond the scope of this paper. We shall use the term "phone" from now on. American English uses about 50 basic phones. (Shoup [1980] provides a list.) The exact number of phones that one uses depends on linguistic considerations.

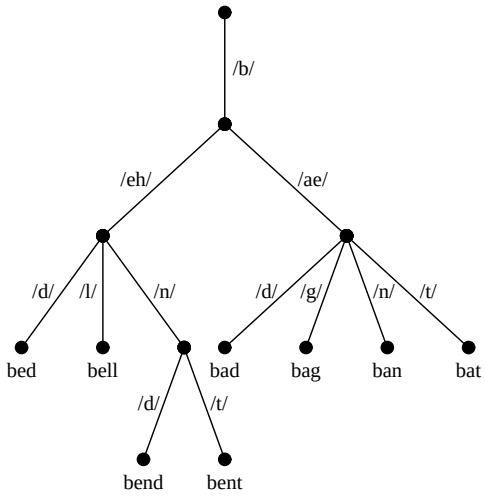

Fig. 5. A trie representing common pronunciations of the words "bed," "bell," "bend," "bent," "bad," "bag," "ban," and "bat."

Let $\mathcal{P}$ denote the alphabet of phones (fixed a priori). With each word $w \in D$ we associate a finite set of strings in $\mathcal{P}^*$ (each describing a different pronunciation of w). This set (often unitary) can be represented, in a straightforward way, using a directed graph G_w, in which each arc is labeled with a phone. The set $\{G_w | w \in D\}$ forms the *lexicon*. Usually the lexicon is represented in a compact form by a trie over $\mathcal{P}$. Figure 5 gives an example.

As defined, the lexicon is a static data structure, not readily usable for speech recognition. It gives a written representation of the pronunciations of the words in D, but it does not contain any acoustic information about the pronunciations, whereas the input string is over the alphabet $\mathcal{F}$ of feature vectors, which encode acoustic information. Moreover, for $w \in D$, G_w has no probabilistic structure, although, as intuition suggests, not all phones are equally likely to appear in a given position of the phonetic representation of a word. The latter problem is solved by using estimation procedures to transform G_w into a Markov source MS_w (necessitating estimating the transition probabilities on the arcs).

Let us consider a solution to the former problem. First, the phones are expressed in terms of feature vectors. For each phone $f \in \mathcal{P}$, one builds (through estimation procedures) a *HMM*, denoted HMM_f, over the alphabet $\Sigma = \mathcal{F}$. Typically, each phone *HMM* is a directed graph having a minimum of four and a maximum of seven states with exactly one source, one sink, self-loops, and no back arcs, i.e., arcs directed from one vertex towards another closer to the source. (See Figure 6(a).) HMM_f gives an *acoustic model* describing the different ways in which one can pronounce the given phone. Intuitively, each path from a source to a sink in HMM_f gives an acoustic representation of a given pronunciation of the phone. In technical terms, HMM_f is a device for computing how likely it is that a given observation sequence $X \in \mathcal{F}^*$ acoustically matches the given phone: this measure is given by $\Pr(X|HMM_f)$ (which can be computed as described in Sections 4.1–4.2). Notice that this model captures the intuition that not all observation sequences

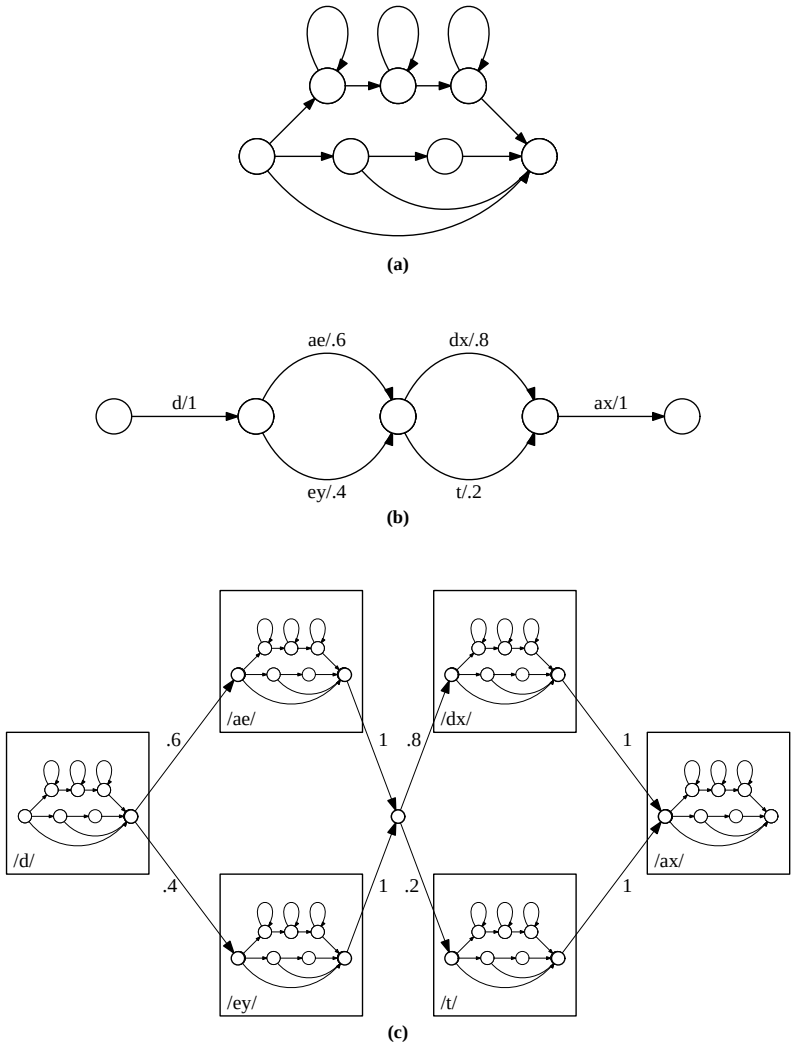

Fig. 6. (a) Topology of a typical seven-state phone *HMM* [Bahl et al. 1993; Lee 1990]. Circles represent states, and arcs reflect non-zero transition probabilities between connected states. Probabilities as well as output symbols (from the alphabet of feature vectors) depend on the specific phone and are not shown. (b) A Markov source for the word "data," taken from Pereira, Riley, and Sproat [1994]. Circles represent states, and arcs represent transitions. Arcs are labeled f/ρ, denoting that the associated transition outputs/recognizes phone $f \in \mathcal{P}$ and occurs with probability ρ. The phones are transcribed in ARPABET [Shoup 1980]. (c) A hidden Markov model for the word "data," built using the Markov source in (b), with the individual phone *HMM*s of (a) replacing the *MS* arcs. Each *HMM* is surrounded by a box marked with the phone it outputs/recognizes. Transition probabilities, taken from (b), are given on arcs that represent state transitions between the individual phone *HMM*s. Other probabilities as well as the individual (feature vector) outputs from each state are not shown.

are equally likely to match a given phone acoustically. We also remark that the observation probability distribution associated with each state is the probability distribution associated with $\mathcal{F}$.

Once the *HMM* for each phone has been built, we can obtain the acoustic model for a word w by replacing each arc labeled $f \in \mathcal{P}$ of MS_w with HMM_f. The result is a *HMM* providing an acoustic model for w (which we denote HMM_w). We will not describe this process explicitly. We point out, however, that it requires the introduction of additional arcs and vertices to connect properly the various phone *HMM*s. (See Figure 6(b)–(c).)

Although the approach we have presented for obtaining acoustic word models may seem quite specialized, it is quite modular. In one direction, we can specialize it even further by eliminating phones as building blocks for words and by computing directly from training data the *HMM*s HMM_w, for $w \in D$. This approach is preferable when the dictionary D is small. In the other direction, we can introduce several different layers of abstraction between the phones and the words. For instance, we can express phones in terms of acoustic data, syllables in terms of phones, and words in terms of syllables. Now, the *HMM*s giving the acoustic models for syllables are obtained using the *HMM*s for phones as building blocks, and, in turn, the *HMM*s giving acoustic models for words are obtained using the *HMM*s for syllables as building blocks. In general, we have the following layered approach. Let $\mathcal{P}_i$ be the alphabet of units of layer i, $i = 0, \cdots, k$, with $\mathcal{P}_k = D$. The lexicon of layer i is a set of directed graphs. Each graph corresponds to a unit of $\mathcal{P}_i$ and represents this unit as a set of strings in $\mathcal{P}_{i-1}^*$, $i \geq 1$. We obtain acoustic models as follows.

(1) Using training procedures, build *HMM* acoustic models for each unit in $\mathcal{P}_0$ using the alphabet of feature vectors $\mathcal{F}$.

(2) Assume that we have the *HMM* acoustic models for the units in layer $\mathcal{P}_{i-1}$, $i \geq 1$. For each graph in the lexicon at level i, compute the corresponding *MS*. Inductively combine these Markov sources with the *HMM*s representing the units at the previous layer $(i-1)$ to obtain the acoustic *HMM* models for the units in $\mathcal{P}_i$.

A few remarks are in order. As discussed earlier, *HMM*s and *MS*s are essentially the same objects. The layered approach introduced here, however, uses an *HMM* for its base layer and then *MS*s for subsequent layers. The reason is convenience. Recall that the alphabet of feature vectors is not bounded. To use a *MS* to model phones, its alphabet should be $\Sigma = \mathcal{F}$, and therefore the out-degree of each vertex in the *MS* would be unbounded, causing technical problems for the use of the *MS* in practical recognizers, in that the graphs to be searched would be unbounded. The problem of unbounded out-degree does not arise with *HMM*s, however: The alphabet is associated to the states, and, even if it is unbounded, no difficulties arise as long as the observation probability function b can be computed quickly for each symbol in $\mathcal{F}$.

Notice also that the acoustic information for layer $\mathcal{P}_i$ is obtained by substituting lexical information into the Markov sources at level i with acoustic information known for the lower level $i-1$ (through hidden Markov models). These substitutions introduce a lot of redundancy into the acoustic model at all levels in this hierarchy of

layers. For instance, the same phone may appear in different places in the phonetic transcription of a word. When building an acoustic model for the word, the different occurrences of the same phone will each be replaced by the same acoustic model. The end result is that the graph representing the final acoustic information will be huge, and the search procedures exploring it will be slow.

RESEARCH AREA 4.2. *One of the recurring problems in speech recognition is to determine how to alleviate this redundancy. A critical open problem, therefore, is to devise methods to reduce the sizes of* HMM*s and lattices, and we discuss this in more detail in Sections 5 and 8.*

Limited to phones and words, this layered acoustic modeling, or variations of it, is used in a few current systems. (Kenney et al. [1993] and Lacouture and Mori [1991] are good examples.) In its simplest form, the lexicon is a trie defined over the alphabet of phones, with no probabilistic structure attached to it [Lacouture and Mori 1991], whereas in other approaches, the trie structure as well as the probabilistic structure is preserved [Kenny et al. 1993]. Even in such specialized layered acoustic modelings, there is the problem of redundancy, outlined above. The approaches that are currently used to address this problem are heuristic in nature, even when they employ minimization techniques from automata theory [Hopcroft and Ullman 1979], ignoring the probability structure attached to *HMM*s.

Finally, the above approach does not account for coarticulatory effects on phones. That is, the pronunciation of a phone f depends on preceding and following phones as well as f itself. For instance, contrast the pronunciations of the phones at word boundaries in Figure 2 with their counterparts in Figure 3. How to model these dependencies is an active area of research. (Lee [1990] gives a good overview.) One solution is to use *context-dependent diphones* and *triphones* [Bahl et al. 1980; Jelinek et al. 1975; Lee 1990; Schwartz et al. 1984]. Rather than build an acoustic model for each phone $f \in \mathcal{P}$, we build models for the diphones αf and $f\beta$ and the triphones $\alpha f \beta$, for $\alpha, \beta \in \mathcal{P}$. The diphones model f in the contexts of a preceding α and a following β, respectively, and the triphones model f in the mutual context of a preceding α and a following β. The diphone and triphone models are then connected appropriately to build word *HMM*s. Two problems arise due to the large number of diphones and triphones: memory and training. Storing all possible diphone and triphone models can consume a large amount of memory, especially considering that many models may be used rarely, if ever. Also due to this sparsity of occurrence, training such models is difficult; usually some sort of interpolation of available data is required.

4.4 The Language Model

Given a language $\mathcal{L}$, the language model provides both a description of the language and a means to compute $\Pr(W)$, for each $W \in \mathcal{L}$. $\Pr(W)$ is required for the computation of Equations 3–4. Let $W = w_1 \cdots w_k$. $\Pr(W)$ can be computed as

$$\Pr(W) = \Pr(w_1 \cdots w_k) = \Pr(w_1) \Pr(w_2|w_1) \cdots \Pr(w_k|w_1 \cdots w_{k-1}).$$

It is infeasible to estimate the conditional word probabilities $\Pr(w_j|w_1 \cdots w_{j-1})$ for all words and sentences in a language. A simple solution is to approximate

$\Pr(w_j|w_1 \cdots w_{j-1})$ by $\Pr(w_j|w_{j-K+1} \cdots w_{j-1})$, for a fixed value of K. A *K-gram language model* is a Markov source in which each state represents a $(K-1)$-tuple of words. There is a transition between state S representing $w_1 \cdots w_{K-1}$ and state S' representing $w_2 \cdots w_K$ if and only if w_K can follow $w_1 \cdots w_{K-1}$. This transition is labeled w_K, and it has probability $\Pr(w_K|w_1 \cdots w_{K-1})$. Usually K is 2 or 3, and the transition probabilities are estimated by analyzing a large corpus of text. (Jelinek, Mercer, and Roukos [1992] give an example.)

A few comments are in order. First, by approximating the language model by a K-gram language model, the search algorithms that use the latter model are inherently limited to computing approximations to Equations 3–4. Moreover, the accuracy of these approximations can only be determined experimentally. (We do not know $\Pr(W)$.) Another problem is that for a typical language dictionary of 20,000 words and for $K = 2$, the number of vertices and arcs of a 2-gram language model would be over four hundred million. The size of the language model may thus be a serious obstacle to the performance of the search algorithm. One way to alleviate this problem is to group the K-tuples of words into equivalence classes [Jelinek et al. 1992] and build a reduced K-gram language model in which each state represents an equivalence class. This division into equivalence classes is performed via heuristics based on linguistic as well as task-specific knowledge. Jelinek, Mercer, and Roukos [1992] provide a detailed description of this technique.

Analogous to coarticulatory effect on phones, we can also consider modeling inter-word dependencies—how the pronunciation of a word changes in context—in the language model. One approach is to insert boundary phones at the beginnings and endings of words and connect adjacent words accordingly. (Bahl et al. [1980] and Jelinek, Bahl, and Mercer [1975] give examples.) This approach makes the language model graph even larger, affecting future search algorithms, and also contributes to the redundancy problem outlined in the previous section.

4.5 Use of Models

Here we briefly discuss how the modeling tools are actually used in speech recognition. Recall from Section 3 that, given an observation sequence X, we have to compute the sentence $\hat{W} \in \mathcal{L}$ minimizing Equation 4. In principle, this computation can be performed as follows. We can use the "layered approach" described in Section 4.3 to build a *HMM* for the language $\mathcal{L}$. That would give a canonical acoustic model for the entire language. Then, we could use either the forward or Viterbi procedure (cf. Section 4.1) to perform the required computation (or an approximation of it). Unfortunately, the *HMM* for the entire language would be too large to fit in memory, and, in any case, searching through such a large graph is too slow.

The approach that is currently used instead is the one depicted in Figure 1, in which the search phase is divided into pipelined stages. The output of the first stage is a phone lattice, i.e., a directed acyclic graph (DAG) with arcs labeled by phones. Each arc has an associated weight, corresponding to the probability that some substring of the observation sequence actually produces the phone labeling the arc. This graph is given as input to the second stage and is "intersected" with the lexicon. The output is a word lattice, i.e, a DAG in which each arc is labeled with a word and a weight. Figure 7 gives an example. The weight assigned to an

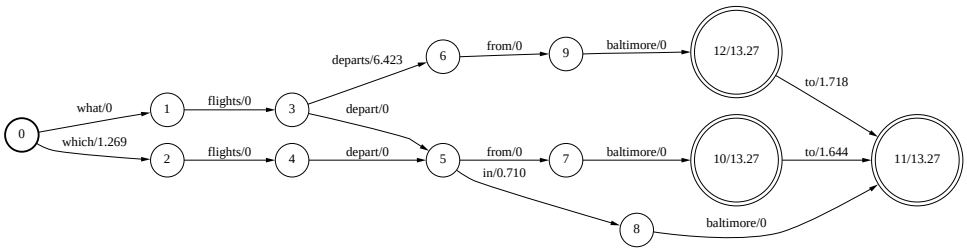

Fig. 7. A word lattice produced for the utterance, "What flights depart from Baltimore?" Arcs are labeled by words and weights; each weight is the negated log of the corresponding transition probability. Final states are in double circles and include negated log probabilities of stopping in the corresponding state.

arc corresponds to the cost that a substring of phones (given by a path in the phone lattice at the end of the previous stage) actually produces the word labeling the arc. Finally, the word lattice is "intersected" with the language model to get the most likely sentence corresponding to the observation sequence.

Each stage of the recognition process depends heavily on the size of the lattice received as input. In order to speed up the stage, each lattice is pruned to reduce its size while (hopefully) retaining the most promising paths. To date, pruning has been based mostly on heuristics. (See for instance, Ljolje and Riley [1992], Riley et al. [1995], and the literature mentioned therein.) As we will see, however, very recent results on the use of weighted automata in speech recognition [Mohri 1997b; Pereira et al. 1994; Pereira and Riley 1997] have provided solid theoretical ground as well as impressive performance for the problem of size reduction of lattices.

In Sections 5–6 we will present the main computational problems that so far have characterized the construction of pipelined recognizers. Then, in Sections 7–8 we will outline a new approach to recognition and its associated computational problems. The novelty of the approach consists of considering the recognition process as a transduction. Preliminary results are quite encouraging [Pereira and Riley 1997].

5. THE VITERBI ALGORITHM

One of the most important tools for speech recognition is the *Viterbi algorithm* [Viterbi 1967]. Here we present a general version of it in the context of IWR, and we state a few related open problems.

Let G_D be the lexicon for D. Assume that G_D is a directed graph, with one designated source node and possibly many designated sink nodes, in which each arc is labeled with a phone; multiple arcs connecting the same pair of nodes are labeled with distinct phones. We say that a path in the graph is *complete* if it starts at the source and ends at a sink. For each word in D, there is a complete path in G_D that induces a phonetic representation of the word. Let MS_D be the Markov source corresponding to G_D; i.e., MS_D has the same topological structure as does G_D and, in addition, a probability structure attached to its arcs. We can transform MS_D into a hidden Markov model H_D by applying to MS_D the same procedure that transforms MS_w, $w \in D$, into HMM_w. (See Section 4.3.) We remark that the

output alphabet of H_D is $\mathcal{F}$. Notice that both MS_D and H_D are directed graphs with one source. For example, in Figure 6(c), the first (leftmost) state of the *HMM* for the phone /d/ is the source, and the last (rightmost) state for the phone /ax/ is the only sink. Assume that H_D has N states.

The problem is the following: Given an input string $X = x_1 \cdots x_T \in \mathcal{F}^*$ (corresponding to the acoustic observation of a word), we want to compute Equation 4, where W is restricted to be one word in the dictionary. $Cs(W)$ is given by the language model. (If not available, $Cs(W)$ is simply ignored.) Thus, the computation of Equation 4 reduces to computing $Cs(X|w)$, for each $w \in D$. Since the only acoustic model available for w is HMM_w, we can consider $Cs(X|w)$ to be the overall cost $Cs(X|HMM_w)$ of the model HMM_w generating X. (Since $Cs(X|HMM_w)$ depends on $\Pr(X|HMM_w)$, we are simply solving Problem 4.1 of Section 4.)

This approach, however, would be too time consuming for large dictionaries. Moreover, it would not exploit the fact that many words in D may have common phonetic information, e.g., a prefix in common. We can exploit these common phonetic structures by estimating $Cs(X|HMM_w)$ through the shortest complete path in H_D that generates X. Since this path ends at a sink, it naturally corresponds to a word $\hat{w} \in D$, which approximates the solution to Equation 4. It is an approximation of the cost of $\hat{w}$, because it neglects other, longer paths that also induce the same word. The validity of the approximation is usually verified empirically.

For example, in Figure 6(c), a complete path generates an acoustic observation for the word "data." If we transform the arc lengths into the corresponding negative log probabilities, then the shortest complete path gives the optimal state sequence that generates such an acoustic observation. That path yields an approximation to the "best" acoustic observation of the word "data," i.e., the most common utterance of the word.

We note that here we see the recurrence of the redundancy problem mentioned in Research Area 4.2. In this case, we want to remove as much redundancy from the lexicon as possible while trying to preserve the accuracy of the recognition procedures. Indeed, we have compressed the lexicon $\{G_w | w \in D\}$ and the corresponding set $\{HMM_w | w \in D\}$ by representing the lexicon by a directed labeled graph. On the other hand, we have ceded accuracy in the computation of $Cs(X|HMM_w)$ by approximating it. Assuming that experiments demonstrate the validity of this approximation, the computation of Equation 4 has been reduced to the following restatement of Problem 4.2.

Problem 4.2 *Given an input string $X = x_1 \cdots x_T \in \mathcal{F}^*$, compute a complete path $Q = q_1 \cdots q_T$ that minimizes* $\mathrm{Cs}(Q|X, H_D)$ *or, equivalently,* $\mathrm{Cs}(Q, X|H_D)$.

We will compute a path Q that minimizes the latter cost. For the remainder of this section we will work with H_D. Let $in(s)$ be the set of states that have arcs going into s (states s' such that $a_{s',s} > 0$), and let $V_t(s)$ be the lowest cost of a single path that accounts for the first t symbols of X and ends in state s.

$$V_t(s) = \min_{q_1, \cdots, q_{t-1}} \{Cs((q_1, \cdots, q_{t-1}, q_t = s), (x_1, \cdots, x_t)|H_D)\}.$$

Letting $c_{tr}(s_1, s_2)$ be the cost of the transition between states s_1 and s_2, $c_o(s, x)$

the cost for state s to output x, and $c_i(s)$ the cost for state s to be the initial state,[3] one can express $V_t(s)$ recursively.

$$V_1(s_0) = c_i(s_0) + c_o(s_0, x_1), \text{ for source state } s_0; \quad (6)$$

$$V_t(s) = \min_{s' \in in(s)} \{V_{t-1}(s') + c_{tr}(s', s)\} + c_o(s, x_t), \ s \in [1..N], \ t > 1. \quad (7)$$

It is easy to see that, using these equations, we can determine the best state path in $O(|E|T)$ operations, where E is the set of arcs in the graph underlying H_D. (Refer back to the discussion of Problem 4.2 in Section 4.1.3.)

Research Area 5.1. *The major open problem regarding the computation of equation Equation 7 is to derive faster algorithms that either compute Equation 7 exactly or yield a provably good approximation to it. As with Research Area 4.1, a direction for experimental work is to investigate how to characterize and exploit the topologies of the relevant graphs to achieve faster search algorithms; i.e., tailor search algorithms to handle these particular special cases of graphs.*

In what follows, we describe two current major lines of research directed at Research Area 5.1.

5.1 Graph Theoretic Approach

We first introduce some notation. We denote the class of classical shortest-path problems on weighted graphs as *CSP*. (Cormen, Leiserson, and Rivest [1991] summarize such problems.) Recall that the length of a path is the number of arcs in it. We refer to the shortest-path problem solved by the Viterbi algorithm as *VSP* and, for each vertex v, to the shortest path to v computed by the Viterbi algorithm at iteration t as $vpath(v, t)$. We would like to establish a relationship between *CSP* and *VSP*.

In *CSP*, the contribution that each arc can give to the paths using it is fixed once and for all when the graph G is specified. Exploiting this cost structure and/or the structure of graph G, one can obtain fast algorithms for problems in *CSP* [Cormen et al. 1991].

At a very high level, the paradigm underlying efficient algorithms for solving special cases of the single-source shortest path problem is the following. At each iteration, maintain a partition of the vertices into two sets: *DONE* and *ACTIVE*. For each vertex x in *DONE*, the algorithm has computed the shortest path to x and is sure that it will not change. For the vertices in *ACTIVE*, only estimates of the shortest paths are available. Examples of this scheme are Dijkstra's algorithm [Dijkstra 1959] (which exploits the fact that the graph has nonnegative arc weights) and the algorithm for directed acyclic graphs (which exploits the topological structure of the graph). Unfortunately, the partition cannot be efficiently maintained for arbitrary graphs with negative weights.

In *VSP*, the contribution that each arc (u, v) of H_D can give to the *vpath*s using it has two main parts: the cost of the arc and the cost of how well a given input

[3] I.e., using the definition of *HMM*s in Section 4.1, $c_{tr}(s_1, s_2) = -\log a_{s_1,s_2}$; $c_o(s, x) = -\log b_s(x)$; and $c_i(s) = -\log \pi_s$.

symbol matches the symbols that state u can output. Whereas the first cost is fixed once and for all, the second depends on the **time** t that arc (u, v) is traversed, since t determines which input symbol we are matching. Thus, the cost of traversing an arc (u, v) when solving *VSP* is dynamic. In general, with this dynamic cost structure, it does not seem possible to maintain efficiently a partition of the vertices of H_D into two classes, as a Dijkstra-type algorithm requires. Indeed, even if the costs on the arcs and vertices of H_D are nonnegative, the fact that they change depending on **when** they are traversed implies that the cost of $vpath(v, t)$ may change from time t to $t+1$. That is, we can be sure that we have computed the best-cost path to v only at time T. Informally, the way in which this dynamicity of costs in *VSP* affects the computation of *vpath*s is similar to the way in which the introduction of negative arc weights affects the computation of shortest paths in *CSP*. Indeed, there is a striking similarity between the Viterbi and Bellman-Ford algorithms for shortest paths [Bellman 1958; Ford and Fulkerson 1962]: the structure of the computation is essentially the same, except that the length of the *vpath* is bounded by T in Viterbi whereas the length of the shortest path is bounded by $|V|$ in Bellman-Ford.

Another technique that has proven successful for *CSP* is *scaling* [Edmonds and Karp 1972; Gabow 1985; Gabow and Tarjan 1989]. There is no analog to this technique in the speech recognition literature. Intuitively, scaling transforms a general *CSP* problem into an equivalent and computationally simpler problem on a graph with nonnegative arc weights.

Research Area 5.2. *Devise a technique analogous to scaling that would transform* VSP *into an equivalent and computationally simpler problem.*

As mentioned previously, an interesting avenue to explore is how the computation of Equation 7 depends on the topological structure of H_D. For instance, the vertices are processed in an arbitrary order for any given step of the computation of Equation 7. An analysis of the structure of the *HMM* may suggest more effective processing orders for the vertices.

5.2 Language Theoretic Approach

Let Q be a deterministic finite-state automaton that accepts strings from a language L. It is well known that, starting from Q, we can compute a minimal automaton Q^*, i.e., the one with the smallest number of states, that accepts strings from L [Hopcroft and Ullman 1979]. One can think of Q^* as the "most efficient" deterministic automaton that supports the membership operation for the language L. Q^* is obtained by defining an equivalence relation R on the strings of Σ^* and the states of Q: xRy if and only if $\delta(q_0, x) = \delta(q_0, y)$, where q_0 and δ are the initial state and the transition function of Q, respectively. The states of Q^* are the equivalence classes obtained by applying R to the states of Q. The states of Q^* can be suitably connected so that Q^* still recognizes L, because one can show that R is *right invariant*; i.e., xRy implies that $xzRyz$.

Continuing, H_D can be seen as some kind of finite automaton that supports the operation: Given input string x of length t, find the best $vpath(v, t)$, $v \in V$. Analogous to the membership operation defined for languages, we would like to build a "minimal" H'_D that supports the operation just defined for H_D. To be useful, H'_D should have substantially fewer states than H_D, and the Viterbi

computation on H'_D should run substantially faster than on H_D. That is, we would like to eliminate some of the redundancy in H_D to avoid the repetition of the Viterbi computation on similar parts of the graph underlying H_D. This is a restatement of Research Area 4.2. The above problem requires investigation at both the definitional and computational levels. Indeed, variations of the stated minimization problem may turn out to be more relevant to speech recognition. That H_D is not deterministic must also be considered. (Here again we see the theme of elimination of redundancy versus accuracy of recognition.)

We now explore some of the difficulties that one may face in trying to define such H'_D. We would like to obtain an H'_D that *preserves the Viterbi computation* of H_D (i.e., yields the same solution, or a good approximation, to Equations 6–7) but that has fewer states than H_D. For a string x of length t, let

$$vstate\,(x) = \{s \in [1..N] \mid vpath(s,t) \leq vpath(s',t),\ s' \in [1..N]\}\,;$$

i.e., $vstate\,(x)$ is the set of states $\{s\}$ of H_D such that $vpath(s,t)$ is minimum when computed over input string x. Let R be the following equivalence relation, defined over the strings of Σ^* of length t: xRy if and only if $vstate\,(x) = vstate\,(y)$. R induces a partition of the states of H_D into equivalence classes. One can easily build an example showing that R is not right invariant, however. Therefore, we cannot obtain an automaton "equivalent to" H_D based on such an equivalence relation, because we cannot "connect" the equivalence classes to obtain a *HMM* H'_D.

The natural question here is to identify right-invariant equivalence relations over the states of H_D that try to achieve the goal of eliminating the redundancy from H_D while trying to preserve the Viterbi computation on H_D. That is, it would be interesting to obtain a *HMM* H'_D that is not necessarily minimal but that approximates well and fast the behavior of H_D with respect to the computation of Equation 7. Some research related to this question has already been performed. (Kenny et al. [1993] provide an example.) We will revisit this issue in Section 8.

Another approach that has been used to speed the Viterbi computation is to introduce a certain amount of nondeterminism into H_D. That is, in some cases [Bahl et al. 1983; Kenny et al. 1993], H_D is augmented with ϵ-transitions. The effect of these transitions is to reduce the size of H_D and therefore to speed the computation of *VSP*.

6. A STRUCTURED APPROACH TO IWR

Another fundamental tool for speech recognition tasks is the A^* *algorithm*, a computational paradigm for solving optimization problems that involve searching large, weighted graphs. Hart, Nilsson, and Raphael [1968] originally introduce and give an extensive treatment of this paradigm. Here we first reformulate Problem 4.1 as an optimization problem over the lexicon G_D. (We refer to the new problem as Problem 6.1.) Then we outline how the A^* algorithm can be used to find a feasible solution to Problem 6.1, and we also provide a general framework for studying a wide class of optimization problems that are related to Problem 6.1. Finally, we outline some general principles used in IWR to design good algorithms for such optimization problems; many of these principles extend to CSR as well.

6.1 An Optimization Problem on $\mathbf{G_D}$

Let $\mathbf{f}$ be a sequence of phones corresponding to a path in MS_D, starting at the source. We call $\mathbf{f}$ a *transcript* and say that it is *complete* when the corresponding path is complete.

Recall from Section 5 that a solution to Problem 4.1 consists of finding a complete path Q in H_D that minimizes $Cs(Q, X|H_D)$. In that section, we also outline how to obtain H_D from MS_D by substituting phones with the associated *HMM*s. Since the complete path Q in H_D minimizing $Cs(Q, X|H_D)$ starts at the source and ends at a sink, it naturally corresponds to a complete path in MS_D that induces a sequence of phones $\mathbf{f} = f_1 \cdots f_k$. This sequence of phones is the one that "best accounts" for the input sequence X. Therefore, Problem 4.1 can be restated as

PROBLEM 6.1. *Given a string $X = x_1 \cdots x_T \in \mathcal{F}^*$, compute*

$$\underset{\mathbf{f}\ \textit{complete transcript}}{\operatorname{argmin}} \quad \mathrm{Cs}(\mathbf{f}, X|\mathrm{MS}_D). \tag{8}$$

6.2 The A* Algorithm

We outline an algorithm that will find a *feasible* solution to Problem 6.1, i.e., a complete transcript $\mathbf{f}$ that accounts for the input string X. If additional conditions are verified, $\mathbf{f}$ will be an *optimal* transcript, i.e., a real solution to Equation 8. We first introduce some notation. For each transcript $\mathbf{f}$ and string $Y \in \mathcal{F}^*$, let $PCs(\mathbf{f}, Y)$ be as $Cs(\mathbf{f}, Y|MS_D)$, except that $\mathbf{f}$ need not be a complete transcript. PCs is the *cost* of "matching" Y along the path given by $\mathbf{f}$. Let $EECs(\mathbf{f}, Z)$, $Z \in \mathcal{F}^*$, be the *estimated cost* of extending transcript $\mathbf{f}$ into a complete transcript $\mathbf{fg}$ such that $\mathbf{g}$ "matches" Z. This heuristic estimate is performed over all possible extensions of $\mathbf{f}$. We assume the heuristic is known but leave it unspecified. Finally, let $ECs(\mathbf{f}, X) = PCs(\mathbf{f}, X_1) + EECs(\mathbf{f}, X_2)$, where $X = X_1X_2$ and $\mathbf{f}$ "matches" X_1, be the *estimated cost* of transcript $\mathbf{f}$ being a prefix of the optimal solution to Equation 8. At each step, the algorithm keeps transcripts in a priority queue[4] $QUEUE$ sorted in increasing order according to the value of the ECs function. Let $DEQUEUE$ be the operation that removes and returns the item of highest priority (lowest estimated cost) from the queue, and let $ENQUEUE(x, p)$ be the operation that inserts a new item x into the queue according to its priority p.

Algorithm A*

1. $QUEUE = \phi$; $ENQUEUE(\mathbf{f}, ECs(\mathbf{f}, X))$, where $\mathbf{f}$ is the empty transcript.
2. **while** $\mathbf{f} = DEQUEUE$ is not a complete transcript **do**
   2.1 Using the lexicon, compute all legal one-phone extensions $\mathbf{f}g$.
   For each such $\mathbf{f}g$, $ENQUEUE(\mathbf{f}g, ECs(\mathbf{f}g, X))$.
   2.2 **done**.
3. **done**.

The above algorithm is guaranteed to find the complete transcript $\mathbf{f}$ that minimizes $Cs(\mathbf{f}, X|MS_D)$, provided that the estimated cost ECs is *admissible*: that is,

[4] In the speech recognition literature, priority queues are often simply called *stacks*.

$ECs(\mathbf{f}, X) \leq CS(\mathbf{fg}, X|MS_D)$ holds, for all possible extensions $\mathbf{g}$ of $\mathbf{f}$ such that $\mathbf{fg}$ is complete [Hart et al. 1968].

6.3 A General Optimization Problem

We now cast Problem 6.1 into a general framework. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a directed, labeled graph with one source and possibly many sinks. The labels on the arcs of $\mathcal{G}$ come from a finite alphabet Γ. There is a match function $\mathcal{M}$ that measures how well the label on a given arc e of $\mathcal{G}$ matches a string $y \in \Sigma^*$, where Σ is another alphabet. Formally, $\mathcal{M} : \Sigma^* \times \mathcal{E} \rightarrow \mathcal{R}$. In general, $\mathcal{M}$ is not computable in constant time. We define a cost function C, which, for each vertex v of $\mathcal{G}$ and each string $y_1 \cdots y_t \in \Sigma^*$, gives the cost of matching the string with a path in $\mathcal{G}$ that ends in v.

$$C(y_1 \cdots y_t, v) = \min_{\substack{k \geq 1 \\ u_j,\ 0<j<k}} \sum_{i=0}^{k-1} \mathcal{M}(y_{t_i+1} \cdots y_{t_{i+1}}, (u_i, u_{i+1})) \tag{9}$$

subject to the conditions (1) $0 = t_0 < t_1 < \cdots < t_{k-1} < t_k = t$; (2) $(u_i, u_{i+1}) \in \mathcal{E}$, $0 \leq i < k$; and (3) u_0 is the source, and $u_k = v$. Moreover, we assume that, for each $v \in \mathcal{V}$ and $t' < t$, $C(y_1 \cdots y_{t'}, v) \leq C(y_1 \cdots y_{t'} y_{t'+1}, v)$. We derive the following.

PROBLEM 6.2. *Given a string $X = x_1 \cdots x_T \in \Sigma^*$, compute*

$$\min_{v \in sink(\mathcal{G})} C(X, v). \tag{10}$$

The following algorithm finds a *feasible* solution to Problem 6.2, and, if additional conditions are verified, the solution will be *optimal*; *feasible* and *optimal* are defined as in Section 6.2, with respect to Equation 10. Let $EC(v)$ be an estimate of the cost of a feasible solution to Equation 10 that passes through v. This estimate will continually be updated by the algorithm. Moreover, let $EEC(Y, w)$, $Y \in \Sigma^*$ and $w \in \mathcal{V}$, be a *predictor* of how well the paths of $\mathcal{G}$ starting at w and ending at sinks will match Y. This predictor is a heuristic function and is used to estimate and update $EC(w)$. *THR* is a threshold that is continually updated, as we will discuss below. A vertex whose estimated cost rises above *THR* is eliminated from future computation. Initially, *THR* gets an arbitrarily high value. *QUEUE* is a priority queue containing vertices sorted according to the values of their estimated costs.

Algorithm SEARCH

1. $EC(v) = THR - 1$, for each $v \in \mathcal{V}$. $ENQUEUE(v_0, EC(v_0))$ for source v_0.
2. **While** $v := DEQUEUE$ is not a sink **do**
   2.1. For each vertex w such that $e = (v, w) \in \mathcal{E}$ and $EC(v) < THR$, compute $\hat{H} = \min_{1 \leq t < t' \leq T} C(x_1 \cdots x_t, v) + \mathcal{M}(x_{t+1} \cdots x_{t'}, e) + EEC(x_{t'+1} \cdots x_T, w)$.
   2.2. If $\hat{H} < \min\{THR, EC(w)\}$, set $EC(w) = \hat{H}$, and $ENQUEUE(w, \hat{H})$. Update the threshold *THR*.
   2.3. **done**.
3. **done**.

The threshold *THR* constrains the search by eliminating paths that have high costs. It is a dynamic value that is set according to the cost of the "best comparable

path." The notion of thresholds and "best comparable paths" derives from *beam searching* as applied to the Viterbi algorithm [Bahl et al. 1983; Lowerre and Reddy 1980]. In Viterbi beam searching, a common definition of *best comparable path* involves considering all paths with the same number of arcs as a class and defining *THR* for each path in a particular class relative to the cost of the best path in that class. The relative difference between *THR* and the cost of the best comparable path is called the *beam width.* How to set the beam width is another active area of research. A basic problem is that the correct path can easily be locally bad at certain points in the computation; if the beam width is too narrow, the correct path will thus be eliminated from further exploration and never found as the solution.

The condition that guarantees the optimality of **SEARCH** is the *admissibility* of EC—$EC(w)$ must be a lower bound to the cost of any of the paths containing w that are feasible solutions to Equation 10—and THR—the correct path must not fall outside the beam at any point in the computation. What is important to notice is that the above optimization problem requires a dynamic evaluation of the cost of traversing an arc of $\mathcal{G}$ during the search. In general, one cannot assume that this evaluation can be performed in constant time. Moreover, Step 2.1 of **SEARCH** requires the computation of C and EEC. Again, one cannot assume that these computations can be done in constant time, as we will see in the next section.

Research Area 6.1. *Most if not all of the heuristics in the literature for speeding computation of the A^* and* **SEARCH** *algorithms either fail to guarantee actual speedups or fail to guarantee accuracy of approximations. That is not to say that the algorithms perform poorly, simply that the results are only derived empirically. A natural open problem, therefore, is to (1) devise admissible heuristics that will significantly speed computation of the A^* and* **SEARCH** *algorithms and (2) provide theoretical analysis proving both admissibility and computational speedup. A related problem is to determine how to measure theoretically the error rates of fast but inadmissible heuristics.*

6.4 Putting the Concepts Together

Let us apply **SEARCH** to solve Problem 6.1. In the IWR case, $\mathcal{G} = G_D$, $\Gamma = \mathcal{P}$, and $\Sigma = \mathcal{F}$. For an arc e labeled with phone $f \in \mathcal{P}$ and a string $y_1 \cdots y_t \in \mathcal{F}^*$, $\mathcal{M}(y_1 \cdots y_t, e)$ is the cost of the best path in HMM_f matching $y_1 \cdots y_t$ plus the cost of the transition corresponding to e in MS_D. For a vertex v and a string $y_1 \cdots y_t \in \mathcal{F}^*$, $C(y_1 \cdots y_t, v)$ is defined as in Equation 9.

The predictor function EEC depends on the heuristic that is implemented. Usually, the heuristic is chosen so that it can be computed efficiently and in such a way that the search converges rapidly to the solution.

With the above choices of C and $\mathcal{M}$, the computation of Step 2.1 of Algorithm **SEARCH** is time consuming. Indeed, $\mathcal{M}(x_{t+1} \cdots x_{t'}, e)$ is computed via the Viterbi algorithm, and $C(x_1 \cdots x_t, v)$ must be computed over all paths that start at the source of G_D and end in v. A few general ideas are used to speed the computation of this inner loop of **SEARCH**.

6.4.1 *Heuristic Match Functions.* Define a new match function $\mathcal{M}'$ that can be computed quickly. This function approximates $\mathcal{M}$. Define a new cost function $\hat{C}$, analogous to C, but that uses $\mathcal{M}'$ and is possibly restricted to some privileged

paths. **SEARCH** is then used in two possible ways.

(1) With the new functions $\mathcal{M}'$ and $\hat{C}$, run **SEARCH** to get a set of promising words D'. Use $\mathcal{M}$ and C to run **SEARCH** on graph $G_{D'}$ to get the best word from D' matching X.

(2) Restrict the set D' above to be only one word, which is output as a feasible solution.

Usually, the function $\mathcal{M}'$ is obtained by simplifying the hidden Markov models for all the phones. That is, for each phone f, HMM_f is transformed into an "equivalent" HMM'_f that is usually much smaller than HMM_f. Then, $\mathcal{M}'$ is still computed via the Viterbi algorithm but using the new *HMM*s. The reduction from HMM_f to HMM'_f usually eliminates the admissibility of **SEARCH** in the sense that the set D' may not contain the word w corresponding to a complete transcript $\mathbf{f}$ that solves Problem 6.1. We remark that, once again, we find that the study of minimization techniques for hidden Markov models (Research Area 4.2) is central to the development of fast algorithms for speech recognition. The ideas outlined above have been extracted from several papers by Bahl et al. [1988, 1989, 1993].

6.4.2 *Model Compression.* In a preprocessing step, compute a graph G_D^* corresponding to a compressed version of G_D. Perform **SEARCH** on G_D^*, restricting also C and EEC to G_D^*. Usually, the *HMM*s for the phones do not change; i.e., $\mathcal{M}$ is still computed via the Viterbi algorithm on HMM_f, $f \in \mathcal{P}$.

Usually, G_D^* is obtained via standard automata minimization techniques. The equivalence relation R used for the minimization may, however, be weaker than that used to minimize G_D in the automata theoretic sense. The probability structure imposed on G_D by the Markov source MS_D is ignored in computing G_D^*. That is, each arc of G_D^* is considered equally likely to be traversed. The omission of this probability structure is, once again, due to the fact that no minimization techniques exist for hidden Markov models. This omission may compromise the admissibility of **SEARCH**, however. Some of the above ideas have been used by Kenny et al. [1993] and Lacouture and Mori [1991].

6.4.3 *Boundary Detection.* The ideas presented so far are oriented towards a speed-up of **SEARCH** by changing the global structure of G_D and/or the *HMM*s for the phones. Another idea, which is more local to the procedure, is to develop tools to produce good estimates of the values of t and t' that yield the minimum $\hat{H}$ of Step 2.1. Then compute $\hat{H}$ only for such values of t and t'.

The development of such tools reduces to understanding which times t, $1 \leq t \leq T$, are most likely to correspond to phone boundaries in $X = x_1 \cdots x_T$. In general, the problem of identifying word and phone boundaries in an input string X is very difficult and requires knowledge of acoustics and signal processing; see Section 9.

7. SPEECH RECOGNITION AS A TRANSDUCTION

In this section we present a new approach to speech recognition, developed by Pereira et al. [1994, 1997], in which recognition is seen as a composition of several transductions. We recall two definitions from the theory of rational transductions and languages. (Berstel [1979], Berstel and Reutenauer [1988], Eilenberg [1974], and Elgot and Mezei [1965] extensively discuss this theory and its correspondence

to automata.) Given two alphabets Σ and Γ and a semiring $(K, \oplus, \otimes, \overline{0}, \overline{1})$, a *transduction* is a function $T : \Sigma^* \times \Gamma^* \rightarrow K$. Intuitively, a transduction assigns a weight to each pair of strings in $\Sigma^* \times \Gamma^*$. A *weighted language* is a function $L : \Sigma^* \rightarrow K$. Intuitively, a weighted language assigns a weight to each string in Σ^*.

Assume that we take one alphabet as $\mathcal{F}$ (the alphabet of observation "symbols") and the other as the dictionary of words D (from which the language $\mathcal{L}$ is generated). Consider now Equation 4 and for the time being, let us ignore the term $Cs\,(W)$, $W \in D^*$. Recall that $X \in \mathcal{F}^*$. If we interpret $Cs\,(X|W)$ as the weight of the pair (X, W), then the cost function defined for each pair of strings in $\mathcal{F}^* \times D^*$ is a transduction. Since X is fixed when we solve Equation 4, this latter problem reduces to the process of computing $\hat{W} \in D^*$ such that the transduction $(X, \hat{W})$ is the best possible.

The use of transductions allows a novel application of the pipelined approach to speech recognition. (Again, see Figure 1.) Indeed, in Sections 7.1 we show that the solution to Equation 4 can be seen as the "composition" of several transductions. The "composition" operation that we use is associative. Associativity allows the stages of the recognition process depicted in Figure 1 subsequent to signal processing to be run in any order. In turn, this processing-order freedom allows a substantial reduction in the size of the search space.

7.1 Fundamental Equations for Speech via Composition of Transductions

Consider a generic commutative semiring $(K, \oplus, \otimes, \overline{0}, \overline{1})$. Given two transductions $S : \Sigma^* \times \Gamma^* \rightarrow K$ and $T : \Gamma^* \times \Delta^* \rightarrow K$, their *composition* $S \circ T$ is defined:

$$S \circ T(x, w) = \bigoplus_{y \in \Gamma^*} S(x, y) \otimes T(y, w). \tag{11}$$

We say that transduction S is *applied* to weighted language L to yield a weighted language $S[L]$ over Γ, where $S[L](y) = \bigoplus_{x \in \Sigma^*} L(x) \otimes S(x, y)$.

We need to introduce some terminology that relates weighted languages and transductions. That will allow us to consider composition and application as the same operation. Transduction S has two weighted languages associated with it: its first and second *projections*, ϕ_1 and ϕ_2. $\phi_2(S) : \Gamma^* \rightarrow K$ is such that $\phi_2(S)(y) = \bigoplus_{x \in \Sigma^*} S(x, y)$. ϕ_1 is defined similarly. On the other hand, a weighted language L is the *identity* transduction restricted to L. That is, $L(x_1, x_2) = L(x_1)$ if and only if $x_1 = x_2$; otherwise, $L(x_1, x_2) = 0$. Now, it can be easily shown [Pereira and Riley 1997] that the application operation is $\phi_2(L \circ S)$. From now on, $\circ$ will also denote application (implemented via projection and composition). We refer to this operation as *generalized composition*.

We now show how to use those tools to express Equation 4 as a generalized composition of transductions. Our starting point is the term $Cs\,(X|W)$ from Equation 4. Recall that in the layers of abstraction we introduced in Section 4.3, $X \in \mathcal{F}^*$ can be transformed into a string over the alphabet $\mathcal{P}_0$. This latter string can be transformed into a string over the alphabet $\mathcal{P}_1$, and so on until we get strings over the alphabet $\mathcal{P}_k = D$ of words. Without loss of generality, we assume that we have only two layers of abstraction. Now, $\mathcal{P}_0$ is the alphabet of phones, and $\mathcal{P}_1$ is the alphabet of words. Using strings over the alphabet of phones, $Cs\,(X|W)$ can be

rewritten as

$$Cs(X|W) = \min_{Y \in \mathcal{P}_0^*} Cs(X|Y) + Cs(Y|W). \tag{12}$$

A few remarks are in order. If we interpret $Cs(\cdot|\cdot)$ as a weight, then $Cs(X|Y)$ is a transduction for each pair of strings $(X, Y) \in \mathcal{F}^* \times \mathcal{P}_0^*$. Observe that the lattice of feature vectors output by the signal processing module in Figure 1 is a weighted language, which we denote by L_F, that assigns weight 1 to the observation sequence X and zero to any other string in $\mathcal{F}^*$. The precomputed acoustic models that are input to the phonetic recognition module in Figure 1 also comprise a transducer, which we denote by L_A. Then the above transduction $Cs(X|Y)$ is the composition $L_F \circ L_A$, which is computed by the phonetic recognition module in Figure 1.

Similarly $Cs(Y|W)$ is a transduction for each pair of strings $(Y, W) \in \mathcal{P}_0^* \times \mathcal{P}_1^*$. This transduction is computed by the word recognition box in Figure 1. We have already observed that $Cs(X|W)$ is a transduction. Recalling Equation 11 and assuming that we are working with the *min-sum* semiring $(\Re^+, \min, +, \infty, 0)$, we have from Equation 12 that the transduction (X, W) is the composition $L_F \circ L_A \circ L_D$, where L_D is the transducer induced by the lexicon. Now, denote by L_M the precomputed grammar that is input to the task recognition box in Figure 1; i.e., L_M is the language model (a weighted language). We thus reduce solving Equation 4 to computing the sentence $\hat{W}$ of minimum weight in the language

$$\phi_2(L_F \circ L_A \circ L_D \circ L_M). \tag{13}$$

Since $\circ$ is associative, we can compute Equation 13 in several ways. One that corresponds to the pipelined stages outlined in Section 4.5 is $\phi_2(((L_F \circ L_A) \circ L_D) \circ L_M)$. Notice that $L_F \circ L_A$ gives, for each phone sequence, the best cost of generating X. Similarly, $(L_F \circ L_A) \circ L_D$ gives, for each word w, the best cost of generating X. For each sentence in the language, the best cost of generating X is given by $((L_F \circ L_A) \circ L_D) \circ M$.

There may be more profitable ways of computing Equation 13, however, and, as pointed out by Pereira and Riley [1997], the fastest approach is application specific. (The size of intermediate results can depend heavily on the recognition task at hand.)

By using probabilities instead of costs and the *sum-times* semiring $(\Re^+, +, \cdot, 0, 1)$, we could obtain Equation 13 from Equation 3. The elementary operations on which generalized composition is based when working with the *sum-times* semiring are different than those used when working with the *min-sum* semiring, however. This difference is important in practice, since the techniques that effectively reduce the sizes of lattices guarantee reductions only when working with the *min-sum* semiring. (See the discussion at the end of Section 8.1.) Therefore, although Equation 3 and Equation 4 are dual, it seems more profitable to solve the latter.

7.2 Implementation Issues

The reduction of the solution of Equation 4 to the evaluation of Equation 13 raises several issues. First, we need to implement the generalized composition operation between two transductions. As we will briefly outline, this problem is solved by

using the correspondence between transductions and weighted automata. Second, we need a software package for experimentation to establish which evaluation order for Equation 13 works best. (This point depends heavily on the task for which we have built the recognizer.) We briefly discuss those issues here.

As is well known, some classes of transductions and weighted languages can be characterized by weighted automata. (See for instance Kuich and Salomaa [1986].) We define weighted automata in Section 8 when we discuss some computational problems relevant to their use in speech recognition. For the time being, we can think of a weighted automaton as a directed graph with one source and one sink. Each arc of the graph is labeled with a symbol and has a weight. For instance, the word lattice in Figure 7 is a weighted automaton. (The multiple final nodes can be appropriately unified into a single sink.) In general, any lattice can be seen as a weighted automaton. Informally, a string z is accepted by the automaton if there is a path from the source to the sink such that z is obtained by the catenation of the labels on the arcs of the path. The weight (cost) of z is the minimum among the sums of the labels on each path accepting z. (Again, we assume that we are working with the *min-sum* semiring.)

Pereira et al. [1994, 1997] have shown that, given two transductions and the corresponding weighted automata, their generalized composition can be implemented via the "intersection" of the corresponding two automata. This intersection, referred to as *generalized intersection*, is a nontrivial extension of the intersection operation defined on nondeterministic finite automata. (Hopcroft and Ullman [1979] describe this latter operation.) The generalization works roughly as follows. Assume that we have two automata that correspond to two transductions, S and T, to be composed. If we use the ordinary intersection operation for those two automata, we would obtain an intersection automaton that assigns the wrong weights to the pair of strings in the transduction $S \circ T$. In order to obtain the correct weights, one needs to filter out some paths from the intersection automaton. Pereira and Riley [1997] give details.

Now, let us return to the pipelined stages of Section 4.5. We have seen that the output of each stage is given by the "intersection" of a lattice (for instance, the word lattice) with some canonical model (for instance, the language model). Using the approach of Pereira et al. [1994, 1997] those "intersections" are generalized intersections of weighted automata, where one automaton is the lattice and the other is the model. But, by the remarks following Equation 13, we can compute those intersections in any order, rather than in the order given in Section 4.5.

In order to experiment with various alternatives, the operations on weighted automata defined by Pereira et al. [1994, 1997] have been implemented by means of a library of functions, each working on an abstract weighted automaton data type. Moreover, there is also a set of composable shell commands for fast prototyping and experimentation. In conclusion, there is a software package that, for any given (reasonable) recognition task, will generate the code corresponding to the various evaluations of Equation 13. In turn, that code can be used to determine experimentally which evaluation order works best for the recognition task at hand. (Appendix B describes how to obtain this software.)

8. SIZE REDUCTION OF LATTICES

Notice that both the pipelined recognizers described in Section 4.5 and the modular ones described in Section 7 can exploit size reductions in lattices and weighted automata before performing "intersection" operations. The following problem is therefore important to both types of recognizers: Given a weighted automaton, compute the smallest equivalent weighted automaton. Unfortunately, the weighted automata we wish to minimize for speech recognition are nondeterministic. Therefore, we first need to determinize them (when possible) and then minimize them (again, when possible). We first define weighted automata and then discuss those two issues.

A *weighted finite automaton* is a quadruple $\mathcal{A} = (Q, q_0, \Lambda, \delta)$ such that Q is the set of states, q_0 is the initial state, Λ is the set of labels (strings over some finite alphabet Σ) and δ is the set of transitions. A transition $t \in \delta$ is a quadruple (q_1, y, m, q_2) with the following interpretation: Given that the automaton is in state q_1 and it is given in input y, it can move to state q_2 assigning to y the weight m. We assume that the weights are taken from a semiring $(K, \oplus, \otimes, \overline{0}, \overline{1})$. In general, the automaton is nondeterministic. There may be another transition $t' = (q_1, y, m', q_2')$ and, given the input y, the automaton can move to both q_2 and q_2' (when in state q_1). It is convenient to specify a single final state q_f. Obviously, the automaton can be represented as a directed graph, and a path from initial state to final state naturally corresponds to a sequence of transitions. Analogous to the definition in Section 5, a path p that starts at the start state and ends at the final state is a *complete* path. A path p of k arcs *induces* a string $z \in \Sigma^*$ if and only if there is a partition of $z = z_1 \cdots z_k$ such that the i-th arc (from left) in p has label z_i. The *weight* $\mathcal{W}(p)$ of a path p is given by combining the weights of its arcs according to the $\otimes$ operation. A string $z \in \Sigma^*$ is *accepted* by $\mathcal{A}$ if there exists at least one complete path p that induces z. The *weight* of z is given by $\bigoplus \mathcal{W}(p)$, where $\bigoplus$ is taken over all complete paths p that induce z.

8.1 Determinization

Determinization is the following problem. Given a weighted automaton $\mathcal{A}$, compute an automaton $\mathcal{A}'$ accepting the same set of strings with the same weights as $\mathcal{A}$, such that, given any state q' of $\mathcal{A}'$, there is only one transition out of q' that can be taken with a given input symbol. We refer to this *deterministic* automaton as a *sequential* weighted automaton. Determinization of weighted automata turns out to be challenging from the theoretical point of view and relevant for its applications to speech. From the theoretical point of view, not all weighted automata can be determinized. It is therefore natural to seek conditions under which a given weighted automaton can be determinized. Elaborating on results obtained by Choffrut [1978] and Weber and Klemm [1995] in the realm of string-to-string transduction, Mohri [1997a, 1997b] identifies those conditions and provides an algorithm that checks whether they hold; the algorithm is constructive in the case that the input automaton can be determinized. For an arbitrary automaton A, the algorithm takes time exponential in the number of states in A. As we will see, however, it tends to perform extremely well on lattices arising in speech recognition tasks. We now briefly discuss the algorithm and its performance.

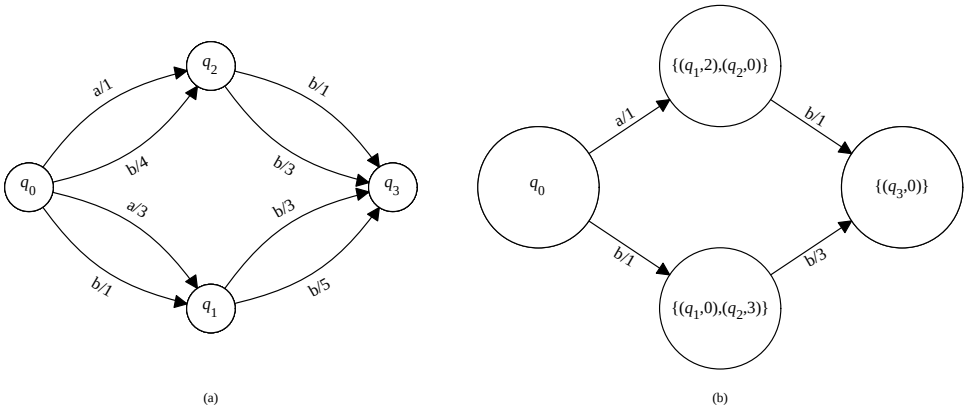

Fig. 8. (a) A nondeterministic weighted automaton. (b) The result of applying Mohri's determinization algorithm to the automaton of (a). This is derived from Figures 11–12 of Mohri [1997a].

Intuitively, the determinization algorithm devised by Mohri [1997a] is a generalization of the determinization procedure for nondeterministic finite automata. We briefly outline the algorithm on an example assuming that we are working with the *min-sum* semiring, although the algorithm will work for any commutative semiring. Consider the weighted automaton in Figure 8(a); its determinization proceeds as follows. From the initial state q_0, we can reach states q_1 and q_2 using the input symbol a. Analogous to the determinization of finite-state automata, we establish a new state $\{q_1, q_2\}$ reachable from q_0 with input symbol a. Since we are interested in minimum weight paths, we assign weight 1 to the new arc. Now, however, we have *remainder* weights of 2 for transition $q_0 \rightarrow q_1$ and 0 for the transition $q_0 \rightarrow q_2$. We save those remainders in the new state by encoding it as $\{(q_1, 2), (q_2, 0)\}$. Similarly, from state q_0 in the original automaton, we can reach states q_1 and q_2 via symbol b. Again the minimum weight among these transitions is 1, so we assign this weight to the new arc, and encode the remainder weights (0 and 3, respectively) in the new state $\{(q_1, 0), (q_2, 3)\}$. Now, consider state $\{(q_1, 2), (q_2, 0)\}$ in the new machine and input symbol b. Notice that, in the automaton of Figure 8(a), we can reach state q_3 from both q_1 and q_2. For each such original arc, we consider the sum of the weight of the arc and the remainder associated with the original source state encoded in state $\{(q_1, 2), (q_2, 0)\}$ in the new machine; taking the minimum among those values gives us the weight of 1 for the new arc. Since there is only one destination state (q_3) in the original machine, there is no new remainder, so we encode the new destination state as $\{(q_3, 0)\}$. Similarly, we construct an arc with weight 3 on symbol b from $\{(q_1, 0), (q_2, 3)\}$ to $\{(q_3, 0)\}$. The end result is shown in Figure 8(b).

The relevance to speech recognition of this algorithm is as follows. Consider the word lattice in Figure 9, and assume that we are working with the *min-sum* semiring. This word lattice comes from the output of the word recognition module in Figure 1 for the utterance, "What flights depart from Baltimore?" (That the lattice is illegible is partly the point; we hope to reduce its size via determinization.) The lattice must be intersected with the language model to obtain the final answer.

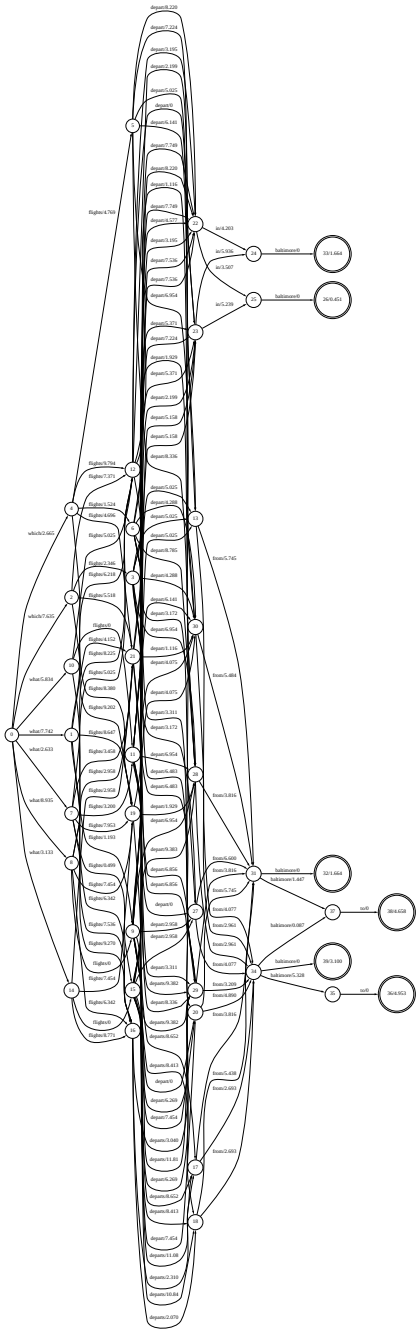

Fig. 9. A word lattice produced for the utterance, "What flights depart from Baltimore?"

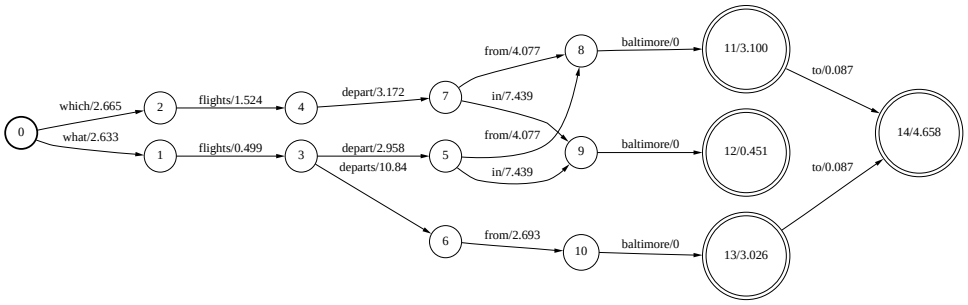

Fig. 10. The lattice resulting from the application of Mohri's determinization algorithm to the word lattice of Figure 9.

Among the many paths in the lattice that induce a given sentence, the path of minimum weight contains the critical information. If we consider the other paths that induce the same sentence in Figure 9 to be redundant, then part of this redundancy can be eliminated by determinization. (We give an intuitive explanation of this phenomenon below.) Figure 10 shows the result of applying Mohri's determinization algorithm to the word lattice of Figure 9. The determinization of the given lattice produces a smaller word lattice that preserves the critical information of the original lattice, as desired. Obviously, if we use the smaller lattice, the intersection with the language model will be faster. Extensive experiments have shown that, when applied to lattices resulting from the various phases of speech recognition, the determinization algorithm of Mohri [1997a] tends to run in linear time and produce smaller lattices than the ones it takes in input. These time and reduction properties derive from the fact that the Mohri's determinization process nicely captures the intuitive meaning of redundancy for those graphs. Indeed, when we ignore the weights on the arcs of the lattice in Figure 9, we obtain a directed acyclic graph that has many isomorphic subgraphs. Moreover, since we are interested only in keeping minimum paths, we can keep, among isomorphic subgraphs, the "lightest" one. Matters are even further simplified because the topological structure of the acyclic graph is very much like that of a tree. The choice of the semiring is also very important. Indeed, for the determinization of the lattice in Figure 9, we are using the *min-sum* semiring. Therefore, for each string z accepted by the lattice, only the path of minimum cost that induces z is relevant, and the other paths can be discarded. An analogous fact does not hold for the *sum-times* semiring.

Research Area 8.1. *Determine when and why determinization of sequential weighted automata can be performed efficiently. That is, characterize the essential properties of the automata that allow efficient determinization. In particular, give formal proof that Mohri's algorithm will always work well for lattices arising in speech recognition (or show where it might produce a blowup in automata size).*

8.2 Minimization

Minimization is the following problem. Given a weighted automaton $\mathcal{A}$, compute the "smallest" automaton equivalent to $\mathcal{A}$. Here matters are somewhat unclear,

and much work is still needed. We first review the state of the art for string-to-string transducers, of which weighted automata comprise a special case. Given a string-to-string transducer T, Reutenauer and Schützenberger [1991] have devised a method to construct a new transducer that performs the same transduction as T and that is minimal with respect to certain constraints. Roche [1995] presents a different approach that leads to a transducer with an exponentially smaller number of states than that of Reutenauer and Schutzenberger. Mohri [1994] also gives a minimization procedure, but his works only for special (although important) classes of transducers. The methods of Roche and Mohri have been used with demonstrated success in natural language processing; Roche [1995], for example, uses his methods to produce a small representation of a French dictionary. Mohri's algorithm, however, is the only one with proven asymptotic results. (Breslauer [1996] gives recent improvements to this algorithm.) As an aside, we note that methods for the representation of dictionaries and phonetic rules is a very active area of experimentation [Roche 1993; Silberztein 1993]. It is also worth mentioning that the minimization algorithm of Roche [1995] is based on the computation of an approximate solution to an NP-Hard problem. Therefore, it is a natural candidate for experimentation. For weighted automata, Mohri [1997a] provides an algorithm that works for sequential weighted automata; it is a specialization of his algorithm for the minimization of string-to-string transducers [Mohri 1994]. We now briefly discuss the algorithm and its performance.

Given a weighted sequential automaton $\mathcal{A}$, the minimization algorithm devised by Mohri [1997a] consists of two stages. One is *extraction*. During this phase a new weighted sequential automaton $\mathcal{A}'$ is built; $\mathcal{A}'$ differs from $\mathcal{A}$ only in the weights on its transitions. Then, $\mathcal{A}'$ is treated as a deterministic automaton with arcs labeled by elements of two alphabets, one of strings and the other of weights. The standard minimization procedure for automata is applied to $\mathcal{A}'$ [Hopcroft and Ullman 1979]. One can show [Mohri 1997a] that the result is the most compact weighted automaton equivalent to $\mathcal{A}$. Figure 7 actually results from applying the minimization algorithm to the word lattice in Figure 10.

We briefly outline the extraction phase. For each state q of $\mathcal{A}$, let $d(q)$ be the minimum among the weights of all strings w that reach the final state from q. The new automaton $\mathcal{A}'$ is defined in exactly the same way as $\mathcal{A}$, except that the weight on the transition (q, a) is given by $\beta+\gamma-d(q)$. β is the weight on the transition (q, a) of $\mathcal{A}$, and $\gamma = d(\delta(q, a))$. One can easily show [Mohri 1997a] that extraction reduces to the computation of a single-source shortest path with nonnegative arc lengths (the source being the final state). Therefore, we can use standard algorithms for this problem [Cormen et al. 1991].

It should be clear that minimization of weighted sequential automata is relevant to speech recognition for the same reasons as is determinization, although the lattice size reductions achieved by minimization do not seem as impressive as those obtained by determinization.

Research Area 8.2. *The minimization algorithm of Mohri [1997a] (and also Breslauer [1996]) produces automata that are the smallest possible but only for special classes of transducers. The algorithm of Roche [1995] is more general but lacks proven asymptotic results. Can these results be unified?*

8.3 Comparison to Viterbi Searching

In the previous sections we have outlined methods for reducing the size of a given lattice while preserving the relevant information—i.e., the minimum cost source-to-sink path. Those methods work because the lattice is given a priori. Moreover, the strings that such a lattice accepts form a finite, albeit large set. We are therefore in a static situation, in which the elimination of redundancy—i.e., irrelevant information—consists of eliminating, for any string x accepted by the lattice, the paths that induce x and that are not guaranteed to be minimal. Technically, this intuition can yield a definition of a *right invariant* equivalence relation that we can use to minimize the lattice [Mohri 1997a].

Now recall from Section 5.2 that we have formulated the problem of minimizing a *HMM* H_D so that the minimized *HMM* H'_D preserves the Viterbi computation. The difference between minimization of lattices and the corresponding problem for *HMM*s is that, while the former are static objects, *HMM*s are used as dynamic objects. That is, the Viterbi computation must provide an optimal path for any string x given as input. That string is usually given on-line, and the number of strings matched by an HMM is infinite.

When we discussed the graph theoretic approach to the Viterbi computation in Section 5.1, we noted that a fundamental difference between Dijkstra-type shortest-path algorithms and Viterbi-type shortest-path algorithms is that the costs of arcs in the former are fixed a priori, whereas the costs of arcs in the latter vary with the input. An analogous situation seems to hold for the language theoretic approach. That is, lattices are used as static objects, while *HMM*s are used as dynamic ones.

9. DISCUSSION

We have formulated some core problems in speech recognition as search problems on very large, weighted graphs. We conclude here with some general comments about this viewpoint; in Appendix A, we review some of the specific open problems mentioned in this paper.

If one uses the layered approach of Section 4.3, one can consider speech recognition as follows. There is a very large, static collection of vertices. Each utterance induces a set of weighted, labeled arcs on the vertex set. The task is to find (or approximate "well") the shortest path through the graph. The problem is that the graph is enormous and also contains redundancies—indeed nested redundancies—that result in repetition of the same shortest-path computations in many different places in the graph. How to reduce the graph or otherwise direct the search so as to avoid most of the graph that does not contribute to the solution, as well as how to exploit the redundancies, are key problems.

If one simply computes directly on the individual *HMM*s rather than "flattening" the graph as above, one faces the problem that arc weights are dynamic with respect to the input sequence. Furthermore, the arcs in each model represent distinct portions of a hypothetical input, whereas the input sequence itself is not explicitly partitioned. Therefore, some mechanism, incurring additional computational complexity per arc traversal, must determine the best ways to match arcs to subsequences of the input. How to speed computation over the *HMM*s, perhaps exploiting their topologies, is a key problem.

Pereira et al. [1994, 1997] consider speech recognition as a composition of weighted finite-state transducers. In their approach, the search problem becomes one of computing on-line multiway "joins" of automata, each of which models various phases in the recognition process. Here again, the transducers become extremely large and contain information that is mostly redundant. Techniques to determinize and minimize these transducers are critical.

Finally, we revisit the decision to concentrate mainly on search issues at the expense of signal processing and acoustics. Consider the approximation

$$\Pr(A|W) \simeq \Pr(A_1|W_1) \cdots \Pr(A_k|W_k) \Pr(W_1, \ldots, W_k), \tag{14}$$

where $A = (A_1, \ldots, A_k)$ and $W = (W_1, \ldots, W_k)$. That is, the probability of observing some sequence A of acoustic feature vectors given that utterance W has been spoken can be approximated by the product of the corresponding probabilities for each *unit* in the utterance and the joint probability of all the units. (For example, the utterance could be a sentence, and we could regard the words as individual units.) Approximation 14 is crucial to the maximum likelihood paradigm, in that even the fastest, most accurate algorithms for solving Equations 3–4 depend on the validity of this approximation to guarantee good results. Boothroyd and Nittrouer [1988], building on the work of Fletcher and Galt [1950] and of Boothroyd [1968], show that Approximation 14 is valid when the units are *allophones*, which are decoded by the human auditory system *before meaning is extracted* (in the words of Allen [1996]). Unfortunately, it is not known yet how to model allophones robustly, so parameterized models (banks-of-filters, linear predictive coding (LPC) coefficients, LPC cepstral coefficients, weighted cepstral coefficients, etc.) have been developed instead; the signal processing module then provides the parameters to plug into the model. Devising such models is a crucial application of signal processing and acoustics to speech recognition. Again we refer the reader to Rabiner and Juang [1993] for an extensive treatment of these issues.

ACKNOWLEDGMENTS

Our understanding of the material in this paper is due in large part to an interdisciplinary seminar that met for several months, in which many papers and problems in speech recognition were discussed. We thank the other participants in those sessions—Hiyan Alshawi, Enrico Bocchieri, Edith Cohen, Emerald Chung, Jim Hieronymus, Andre Ljolje, Yossi Matias, Fernando Pereira, and Mike Riley—for their advice and opinions. We thank Jont Allen, Jon Bentley, David Johnson, Fred Juang, Mehryar Mohri, Ravi Sethi, Bob Tarjan, and Mihalis Yannakakis for their comments and advice. Finally, we thank the anonymous referees for many constructive suggestions that improved the presentation of this paper.

REFERENCES

Allen, J. B. 1996. Harvey Fletcher's role in the creation of communication acoustics. *Journal of the Acoustical Society of America 99*, 4, 1825–39.

Bahl, L. R., Bakis, R., Cohen, P. S., Cole, A. G., Jelinek, F., Lewis, B. L., and Mercer, R. L. 1980. Further results on the recognition of a continuously read natural corpus. In *Proc. IEEE Int'l. Conf. on Acoustics, Speech, and Signal Processing*, Volume 3 (1980), pp. 872–5.

Bahl, L. R., Bakis, R., de Souza, P. V., and Mercer, R. L. 1988. Obtaining candidate words by polling in a large vocabulary speech recognition system. In *Proc. IEEE Int'l. Conf. on Acoustics, Speech, and Signal Processing*, Volume 1 (1988), pp. 489–92.

Bahl, L. R., Gennaro, S. V. D., Gopalakrishnan, P. S., and Mercer, R. L. 1993. A fast approximate acoustic match for large vocabulary speech recognition. *IEEE Transactions on Speech and Audio Processing 1*, 59–67.

Bahl, L. R., Gopalakrishnan, P. S., Kanevski, D., and Nahamoo, D. 1989. Matrix fast match: A fast method for identifying a short list of candidate words for decoding. In *Proc. IEEE Int'l. Conf. on Acoustics, Speech, and Signal Processing*, Volume 1 (1989), pp. 345–8.

Bahl, L. R., Jelinek, F., and Mercer, R. L. 1983. A maximum likelihood approach to continuous speech recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence PAMI-5*, 179–190.

Baum, L. E. and Eagon, J. A. 1967. An inequality with applications to statistical estimation for probabilistic functions of a Markov process and to a model for ecology. *Bulletin of the American Mathematical Society 73*, 360–3.

Baum, L. E. and Sell, G. R. 1968. Growth transformations for functions on manifolds. *Pacific Journal of Mathematics 27*, 211–27.

Bellman, R. 1958. On a routing problem. *Quarterly of Applied Mathematics 16*, 87–90.

Berstel, J. 1979. *Transduction and Context-Free Languages*, Volume 38 of *Leitfaden der angewandten Mathematik und Mechanik LAMM*. Springer-Verlag.

Berstel, J. and Reutenauer, C. 1988. *Rational Series and Their Languages*, Volume 12 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag.

Boothroyd, A. 1968. Statistical theory of the speech discrimination score. *Journal of the Acoustical Society of America 43*, 2, 362–7.

Boothroyd, A. and Nittrouer, S. 1988. Mathematical treatment of context effects in phoneme and word recognition. *Journal of the Acoustical Society of America 84*, 1, 101–14.

Breslauer, D. 1996. The suffix tree of a tree and minimizing sequential transducers. In *Proc. 7th Symposium on Combinatorial Pattern Matching* (1996).

Choffrut, C. 1978. Contributions á l'étude de quelques familles remarquables de function rationnelles. Ph. D. thesis, LITP-Université Paris 7, Paris, France.

Cormen, T. H., Leiserson, C. E., and Rivest, R. L. 1991. *Introduction to Algorithms*. The MIT Electrical Engineering and Computer Science Series. MIT Press, Cambridge, MA.

Dijkstra, E. W. 1959. A note on two problems in connexion with graphs. *Numerische Mathematik 1*, 269–271.

Edmonds, J. and Karp, R. M. 1972. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM 19*, 248–64.

Eilenberg, S. 1974. *Automata, Languages, and Machines*, Volume A. Academic Press, San Diego.

Elgot, C. C. and Mezei, J. E. 1965. On relations defined by generalized finite automata. *IBM Journal of Research and Development 9*, 47–68.

Fletcher, H. and Galt, R. H. 1950. The perception of speech and its relation to telephony. *Journal of the Acoustical Society of America 22*, 2, 89–151.

Ford, L. R. and Fulkerson, D. R. 1962. *Flows in Networks*. Princeton University Press, Princeton, NJ.

Gabow, H. N. 1985. Scaling algorithms for network problems. *Journal of Computer and System Sciences 31*, 148–68.

Gabow, H. N. and Tarjan, R. E. 1989. Faster scaling algorithms for network problems. *SIAM Journal on Computing 18*, 1013–36.

Hart, P. E., Nilsson, N. J., and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems, Science, and Cybernetics 4*, 100–7.

Hopcroft, J. E. and Ullman, J. D. 1979. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley Series in Computer Science. Addison-Wesley, Reading, MA.

Jelinek, F., Bahl, L. R., and Mercer, R. L. 1975. Design of a linguistic statistical decoder for the recognition of continuous speech. *IEEE Transactions on Information Theory IT-21*, 250–6.

Jelinek, F., Mercer, R. L., and Roukos, S. 1992. Principles of lexical language modeling for speech recognition. In S. Furui and M. M. Sondhi Eds., *Advances in Speech Signal Processing*, Chapter 21, pp. 651–99. New York: Marcel Dekker.

Kenny, P., Hollan, R., Gupta, V. N., Lenning, M., Mermelstein, P., and O'Shaughnessy, D. 1993. A*-Admissible heuristics for rapid lexical access. *IEEE Transactions on Speech and Audio Processing 1*, 49–57.

Kuich, W. and Salomaa, A. 1986. *Semirings, Automata, Languages*, Volume 5 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag.

Lacouture, R. and Mori, R. D. 1991. Lexical tree compression. In *Proc. 2nd Euro. Conf. on Speech Communication and Technology*, Volume 2 (1991), pp. 581–4.

Lee, K.-F. 1990. Context-dependent phonetic hidden Markov models for speaker-independent continuous speech recognition. In A. Waibel and K.-F. Lee Eds., *Readings in Speech Recognition*, pp. 347–65. Morgan Kaufman.

Ljolje, A. and Riley, M. D. 1992. Optimal search recognition using phone recognition and lexical access. In *Proc. 2nd Int'l. Conf. on Spoken Language Processing* (1992), pp. 313–316.

Lowerre, B. and Reddy, R. 1980. The Harpy speech understanding system. In *Trends in Speech Recognition*, Chapter 15, pp. 340–60. Englewood Cliffs, NJ: Prentice-Hall.

Mohri, M. 1994. Minimization of sequential transducers. In *Proc. 5th Symposium on Combinatorial Pattern Matching*, Volume 807 of *Lecture Notes in Computer Science* (1994), pp. 151–63.

Mohri, M. 1997a. Finite-state transducers in language and speech processing. To appear in *Computational Linguistics*.

Mohri, M. 1997b. On the use of sequential transducers in natural language processing. In *Finite State Devices in Natural Language Processing*. MIT Press. To appear.

Pereira, F. and Riley, M. 1997. Speech recognition by composition of weighted finite automata. In *Finite State Devices in Natural Language Processing*. MIT Press. To appear.

Pereira, F., Riley, M., and Sproat, R. 1994. Weighted rational transductions and their application to human language processing. In *Proc. ARPA Human Language Technology Conf.* (1994), pp. 249–54.

Pickering, J. B. and Rosner, B. S. 1993. *The Oxford Acoustic Phonetic Database on Compact Disk*. Oxford University Press.

Rabiner, L. and Juang, B.-H. 1993. *Fundamentals of Speech Recognition*. Prentice Hall Signal Processing Series. Prentice Hall, Englewood Cliffs, NJ.

Rabiner, L. R. 1990. A tutorial on hidden Markov models and selected applications in speech recognition. In A. Waibel and K.-F. Lee Eds., *Readings in Speech Recognition*, pp. 367–96. Morgan Kaufman.

Reutenauer, C. and Schützenberger, M.-P. 1991. Minimization of rational word functions. *SIAM Journal on Computing 20*, 4, 669–85.

Riley, M. D., Ljolje, A., Hindle, D., and Pereira, F. C. N. 1995. The AT&T 60,000 word speech-to-text system. In J. M. Pardo, E. Enrìquez, J. Ortega, J. Ferreiros, J. Macìas, and F. J. Valverde Eds., *Proc. 4th Euro. Conf. on Speech Communication and Technology*, Volume 1 (1995), pp. 207–210.

Roche, E. 1993. Analyse syntaxique transformationelle du francais par transducteurs et lexique-grammaire. Ph. D. thesis, LITP-Université Paris 7, Paris, France.

Roche, E. 1995. Smaller representations for finite-state transducers and finite-state automata. In *Proc. 6th Symposium on Combinatorial Pattern Matching*, Volume 937 of *Lecture Notes in Computer Science* (1995), pp. 352–65.

Schwartz, R., Chow, Y., Roucos, S., Krasner, M., and Makhoul, J. 1984. Improved hidden Markov modeling of phonemes for continuous speech recognition. In *Proc. IEEE Int'l. Conf. on Acoustics, Speech, and Signal Processing*, Volume 3 (1984), pp. 35.6.1–4.

Shoup, J. E. 1980. Phonological aspects of speech recognition. In *Trends in Speech Recognition*, Chapter 6, pp. 125–38. Englewood Cliffs, NJ: Prentice-Hall.

Silberztein, M. 1993. Dictionnaires électroniques et analise automatique de textes: le systéme intex. Ph. D. thesis, Masson, Paris, France.

Soong, F. K. and Huang, E.-F. 1991. A tree-trellis based fast search for finding the N best sentence hypotheses in continuous speech recognition. In *Proc. IEEE Int'l. Conf. on Acoustics, Speech, and Signal Processing*, Volume 1 (1991), pp. 705–8.

Viterbi, A. J. 1967. Error bounds for convolutional codes and an asymptotically optimal decoding algorithm. *IEEE Transactions on Information Theory IT-13*, 260–9.

Waibel, A. and Lee, K.-F. Eds. 1990. *Readings in Speech Recognition*. Morgan Kaufmann.

Weber, A. and Klemm, R. 1995. Economy of description for single-valued transducers. *Information and Computation 118*, 327–40.

APPENDIX

A. SUMMARY OF RESEARCH AREAS

Here we summarize the earlier statements of open algorithmic problems. We give page numbers for reference back to the full problem statements.

Research Area 4.1, page 13. Devise faster methods to compute the probability that a *HMM* matches a given observation. In particular, can the topology of the *HMM* be exploited towards this end?

Research Area 4.2, page 19. Devise algorithms to reduce the size of a *HMM*. This is analogous to the determinization and minimization problems on finite-state automata.

Research Area 5.1, page 23. Devise faster search algorithms to solve the Viterbi equation (Equation 7). As with Research Area 4.1, investigate how to characterize and exploit the particular graph topologies that arise in speech recognition.

Research Area 5.2, page 24. Devise an analogue to the *CSP* scaling technique that would apply to *VSP*.

Research Area 6.1, page 28. Investigate the potential for admissible heuristics that will significantly speed computation of the A^* and **SEARCH** algorithms, or determine how to measure theoretically the error rates of fast but inadmissible heuristics.

Research Area 8.1, page 36. Characterize the essential properties of sequential weighted automata that permit efficient determinization.

Research Area 8.2, page 37. Unify the results given by Mohri and Breslauer (provably good minimization for special cases of sequential weighted automata) with those of Roche (minimization for more general cases of sequential weighted automata without proven asymptotic size reductions).

B. SOURCES OF CODE AND DATA

In this section we give some pointers to sources of code, data, etc., of interest to anyone who wants to experiment with speech recognition. Rather than list points of contact for individual data sets, programs, etc., we instead give pointers

to bigger and therefore presumably more durable collections of information. Note that development of speech products is now a business; collection of speech and text corpora is also extremely labor intensive. Therefore, most programs and data are not available without cost.

B.1 Code

Most commercially available speech recognition products are tailored more towards applications developers than researchers. One product, though, called HTK, provides a more low-level toolkit for experimenting with speech recognition algorithms in addition to an application-building interface. It is available from Entropic Research Lab, Inc.

`http://www.entropic.com/htk/`

The finite-state toolkit developed by Pereira et al. (cf. Section 7.2) can be obtained by sending electronic mail to

`fsm@research.att.com.`

B.2 Data

A large variety of speech and text corpora is available from the Linguistic Data Consortium.

`http://www.ldc.upenn.edu/`

Paying a membership fee to join the LDC entitles one to free corpora that were released during that year (and reduced prices on corpora from prior years); nonmembers pay more for corpora. The following are some of the commonly used speech corpora that they have.

TIMIT	Acoustic-Phonetic Continuous Speech Corpora.
RM	Resource Management Corpora.
ATIS	Air Travel Information System.
CSR	Continuous Speech Recognition.
SWITCHBOARD	Switchboard Corpus of Recorded Telephone Conversations.

Among the commonly used text corpora they have are the following.

PENN TREEBANK	The Penn Treebank Project, Release 2.
UN	United Nations Parallel Text Corpus (Complete).
SPANISH NEWS	Spanish News Text Collection.

Another source of speech data is the *Oxford Acoustic Phonetic Database* on CDROM, published by Pickering and Rosner [1993]. It is a set of two CDs that contain digitized recordings of isolated lexical items plus isolated monophthongs from each of the following eight languages/dialects: American English, British English, French, German, Hungarian, Italian, Japanese, and Spanish.

B.3 Commercial Products

As mentioned above, most commercial speech recognition products are tailored more towards applications developers than researchers. Still, those wishing to experiment with the human factors issues of speech recognition (which we did not discuss at all in this paper) might be interested in the following products.

(1) AT&T Watson Advanced Speech Applications Platform.
`http://www.att.com/aspg/blasr.html`

(2) BBN Speech Products.
`http://www.bbn.com/speech_prods/`

(3) DragonDictate from Dragon Systems, Inc.
`http://www.dragonsys.com/`

B.4 General Information

Finally, two free, on-line source of information are of interest. First is the USENET newsgroup

`comp.speech.`

While the signal-to-noise ratio of most USENET newsgroups is pretty low, the `comp.speech` list of frequently asked questions (FAQ), which is posted at least monthly, does provide an extensive, well-maintained list of pointers to other on-line sources of information on speech processing. The current version of the FAQ can be found at

`http://www.speech.cs.cmu.edu/comp.speech/.`

Second is the *Free Speech Journal*, at

`http://www.cse.ogi.edu/CSLU/fsj/home.html,`

an on-line, peer-reviewed journal covering human language technology.

C. GLOSSARY

ARPABET	Standard phonetic alphabet used in ARPA projects.
CSP	Classical shortest-path problems.
CSR	Continuous speech recognition.
DAG	Directed, acyclic graph.
HMM	Hidden Markov model.
IWR	Isolated word recognition.
LPC	Linear predictive coding.
MS	Markov source.
VSP	Shortest-path problem as solved by the Viterbi algorithm.