and π_i is not reestimated since $\pi_1 = 1, \pi_i = 0, i \neq 1$.

The proper scaling of Eqs. (6.110)–(6.111) is now straightforward since each observation sequence has its own scaling factor. The key idea is to remove the scaling factor from each term before summing. This can be accomplished by writing the reestimation equations in terms of the scaled variables, i.e.,

$$\bar{a}_{ij} = \frac{\sum_{k=1}^{K} \frac{1}{P_k} \sum_{t=1}^{T_k - 1} \hat{\alpha}_t^k(i) a_{ij} b_j(\mathbf{o}_{t+1}^k) \hat{\beta}_{t+1}^k(j)}{\sum_{k=1}^{K} \frac{1}{P_k} \sum_{t=1}^{T_k - 1} \hat{\alpha}_t^k(i) \hat{\beta}_t^k(i)}. \tag{6.112}$$

In this manner, for each sequence $\mathbf{O}^{(k)}$, the same scale factors will appear in each term of the sum over t as appears in the P_k term, and hence will cancel exactly. Thus using the scaled values of the αs and βs results in an unscaled $\bar{a}_{ij}$. A similar result is obtained for the $\bar{b}_j(\ell)$ term.

6.12.3 Initial Estimates of HMM Parameters

In theory, the reestimation equations should give values of the HMM parameters that correspond to a local maximum of the likelihood function. A key question is, therefore, How do we choose initial estimates of the HMM parameters so that the local maximum is equal to or as close as possible to the global maximum of the likelihood function?

Basically there is no simple or straightforward answer. Instead, experience has shown that either random (subject to the stochastic and the nonzero value constraints) or uniform initial estimates of the π and A parameters are adequate for giving useful reestimates of these parameters in almost all cases. However, for the B parameters, experience has shown that good initial estimates are helpful in the discrete symbol case and are essential (when dealing with multiple mixtures) in the continuous-distribution case. Such initial estimates can be obtained in a number of ways; these include (a) manual segmentation of the observation sequence(s) into states and averaging of observations within states, (b) maximum likelihood segmentation of observations and averaging, and (c) segmental k-means segmentation with clustering, etc. We discuss such segmentation techniques later in this chapter.

6.12.4 Effects of Insufficient Training Data

Another problem associated with training HMM parameters via reestimation methods is that the observation sequence used for training is, of necessity, finite ([30]). Thus there is always an inadequate number of occurrences of low-probability events (e.g., symbol occurrences within states) to give good estimates of the model parameters. By way of example, consider the case of a discrete observation HMM. Recall that the reestimation transformation of $\bar{b}_j(k)$, Eq. (6.40c), requires a count of the expected number of times in state j and observing symbol $\mathbf{v}_k$ simultaneously. If the training sequence is so small that it does not have any occurrences of this event (i.e., $q_t = j$ and $\mathbf{o}_t = \mathbf{v}_k$), $b_j(k) = 0$ and

will stay 0 after reestimation. The resultant model would produce a zero probability result for any observation sequence that actually includes ($o_t = v_k$ and $q_t = j$). Such a singular outcome is obviously a consequence of the unreliable estimate that $b_j(k) = 0$ due to the insufficiency of the training set.

One solution to this problem is to increase the size of the training observation set. Often this is impractical. A second possible solution is to reduce the size of the model (e.g., number of states, number of symbols per state). Although this is always possible, often there are physical reasons why a given model is used, and therefore the model size cannot be changed. A third possible solution is to seek unconventional statistical estimation algorithms that can somehow enhance the reliability of the parameter estimates even based on limited training data. Deleted interpolation and parameter thresholding are two such alternatives. Since deleted interpolation is considered more an enhanced parameter estimation method, we discuss that subject in the next section.

The simplest way to handle the effects of insufficient training data is to add extra threshold constraints to the model parameters to ensure that no model parameter estimate falls below a specified level [18]. Thus, for example, we might specify the numeric floor, for a discrete symbol model, that

$$b_j(k) = \begin{cases} b_j(k), & \text{if } b_j(k) \geq \delta_b \\ \delta_b, & \text{otherwise} \end{cases} \tag{6.113a}$$

or, for a continuous distribution model, that

$$U_{jk}(r, r) = \begin{cases} U_{jk}(r, r), & \text{if } U_{jk}(r, r) \geq \delta_u \\ \delta_u, & \text{otherwise} \end{cases} . \tag{6.113b}$$

When the numeric floor is invoked in the reestimation equations, all remaining parameters need to be rescaled so that the densities obey the required stochastic constraints. Such postprocessor techniques are thus considered implementational measures to combat the insufficient data problem and have been applied with good success to several problems in speech processing. The method of parameter thresholding has a justification from a Bayes statistics point of view. It can be shown that Eq. (6.113b) is, in fact, a maximum a posteriori (MAP) estimate of the variance under the assumption that the parameter prior $P(U_{jk}(r, r))$ is an informative one with uniform distribution and $(U_{jk}(r, r))_{\min} = \delta_u$ [31]. (See Section 6.13.)

6.12.5 Choice of Model

The remaining issues in implementing HMMs are the choice of type of model (ergodic or left-right or some other form), choice of model size (number of states), and choice of observation symbols (discrete or continuous, single or multimixture, choice of observation parameters). Unfortunately, there is no simple, theoretically correct way of making such choices. These choices must be made depending on the signal being modeled. With these comments, we end our discussion of the theoretical aspects of hidden Markov models and proceed to a discussion of how such models have been applied to the isolated word-recognition problem.

6.13 IMPROVING THE EFFECTIVENESS OF MODEL ESTIMATES

We discuss three methods that have been shown to be able to enhance the effectiveness of HMM model estimates for speech recognition. These are (1) deleted interpolation, (2) Bayesian adaptation, and (3) corrective training. The first two methods are motivated by the problem of insufficient data, while the last method has the unique objective of trying to reduce recognition errors directly.

6.13.1 Deleted Interpolation

When training data are insufficient, reliable and robust determination of HMM parameters cannot be accomplished. The HMM obtained by the Baum-Welch reestimation method, based on the maximum likelihood criterion, may be adequate in characterizing the training data, but for new data the match may be quite poor. One parameter estimation method that aims to improve the model reliability is the method of "deleted interpolation."

The concept involves combining two (or more) separately trained models, one of which is more reliably trained than the other. A scenario in which this can happen is the case when we use tied states which forces "different" states to share an identical statistical characterization, effectively reducing the number of parameters in the model. A model with tied states is often more robust than a model without tied states when trained on the same amount of data. But a model with tied states is also less precise than a model without tied states if the training data are sufficient. Therefore, the idea of combining the two models is to allow us to fall back to the more reliable model when the supposedly more precise model is, in fact, unreliable. A similar scenario occurs when context-independent (more robust) and context-dependent (more precise) phone models are used in large vocabulary recognition (see Chapter 8).

Let the two models be defined by the parameter sets $\lambda = (A, B, \pi)$ and $\lambda' = (A', B', \pi')$, respectively. The interpolated model, $\tilde{\lambda} = (\tilde{A}, \tilde{B}, \tilde{\pi})$, is obtained as

$$\tilde{\lambda} = \epsilon\lambda + (1 - \epsilon)\lambda' \tag{6.114}$$

where ϵ represents the weighting of the parameters of the "full" model (with more detailed characterization of the observations) and $(1 - \epsilon)$ represents the weighting of the parameters of the reduced, but more reliable, model. A key issue is the determination of the optimal value of ϵ, which is a strong function of the amount of training data. This is easy to see because as the amount of training data gets large, λ becomes more reliable and we expect ϵ to tend to 1.0. Similarly, for small amounts of training data, λ is unreliable and we expect ϵ to tend to 0.0 so as to fall back to the more reliable model λ'.

The solution to the determination of an optimal value for ϵ was provided by Jelinek and Mercer [30], who showed how the optimal ϵ could be estimated using the forward-backward algorithm by interpreting Eq. (6.114) as an expanded HMM of the type shown in Figure 6.12. Figure 6.12a shows the part of the state-transition structure related to the state S. Using Figure 6.12b, we can interpret the interpolated model of Eq. (6.114) as an

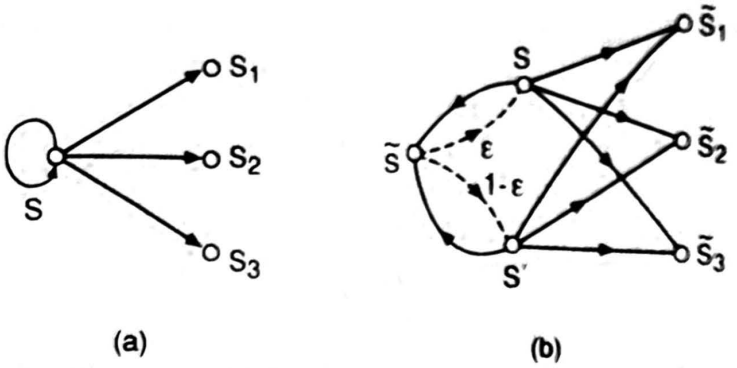


Figure 6.12 Example of how the process of deleted interpolation can be represented using a state diagram.

expanded HMM in which each state is replaced by three states. The null transitions from the expanded state $\tilde{S}$ to S and S' have transition probabilities ϵ and $1 - \epsilon$, respectively. The transitions out of S are characterized by those of λ while those out of S' are associated with those of λ'.

The expanded HMM interpretation suggests that the parameter ϵ can be optimally determined by the usual forward-backward algorithm. However, since the interpolation is designed to better predict unseen (future) data, rather than to account for the training data, determination of ϵ must be based on data that was not used in obtaining either of the two models, λ and λ'. A key idea of deleted interpolation is thus to partition the training data $\mathcal{T}$ into two disjoint sets; that is, $\mathcal{T} = \mathcal{T}_1 \cup \mathcal{T}_2$. For example, one might consider a partition of the training set such that $\mathcal{T}_1$ is 90 percent of $\mathcal{T}$ and $\mathcal{T}_2$ is the remaining 10 percent of $\mathcal{T}$. Training set $\mathcal{T}_1$ is first used to train λ and λ'. Training set $\mathcal{T}_2$ is then used to give an estimate of ϵ, assuming λ and λ' are fixed. There are obviously a large number of ways in which such a partitioning can be accomplished, but one particularly simple one is to cycle $\mathcal{T}_2$ through the data. That is, the first partition uses the last 10 percent of the data as $\mathcal{T}_2$, the second partition uses the next-to-last 10 percent of the data as $\mathcal{T}_2$, etc. An interpretation of deleted interpolation is straightforward. If the unseen data fits the more elaborate model λ well (thus validating the reliability of λ), the forward-backward algorithm would give a value of ϵ which is close to 1. Otherwise, the forward-backward algorithm gives a small value of ϵ, indicating that the more reliable model λ' is a better characterization of the new data than λ.

The technique of deleted interpolation has been successfully applied to a number of problems in speech recognition, including the estimation of trigram word probabilities for language models [13], and the estimation of HMM output probabilities for trigram phone models (to be discussed in Chapter 8 of this book).

6.13.2 Bayesian Adaptation

Another insufficient data situation occurs when we attempt to estimate a speaker-dependent model based on a limited amount of speaker-specific training data. An approach to this

problem is through speaker adaptation, in which a speaker-independent model, obtained by reliable training, is adapted to the particular talker using speaker-specific training data [31].

Speaker adaptation can be accomplished based on a Bayesian framework. Consider the HMM probability measure $P(\mathbf{O}|\lambda)$. If the HMM parameter λ is assumed to be fixed but unknown, the maximum likelihood (ML) estimate for λ, given the training sequence $\mathbf{O}$, is obtained by solving the likelihood equation, i.e.

$$\frac{\partial}{\partial \lambda} P(\mathbf{O}|\lambda) = 0. \tag{6.115}$$

(Normally, the Baum-Welch reestimation algorithm is used to obtain certain stationary point solutions instead of directly solving for Eq. (6.115).) If λ is assumed random with a priori distribution $P_0(\lambda)$, then the maximum a posteriori (MAP) estimate for λ is obtained by solving

$$\frac{\partial}{\partial \lambda} P(\lambda|\mathbf{O}) = 0 \tag{6.116}$$

for the given training sequence $\mathbf{O}$. Using the Bayes theorem, we rewrite $P(\lambda|\mathbf{O})$ as

$$P(\lambda|\mathbf{O}) = \frac{P(\mathbf{O}|\lambda)P_0(\lambda)}{P(\mathbf{O})}. \tag{6.117}$$

The influence of the parameter prior $P_0(\lambda)$ in the solution process thus becomes explicit. Note that if the distribution is correctly chosen, the MAP solution attains minimum Bayes risk.

The parameter prior distribution characterizes the statistics of the parameters of interest before any measurement is made. If the prior distribution indicates no preference as to what the parameter values are likely to be, then the prior is called a noninformative prior (which is essentially constant for the entire parameter space). In this case, the MAP estimate obtained by solving Eq. (6.116) is identical to the ML estimate of Eq. (6.115). If we do have prior knowledge about the parameter values of the model, the incorporation of such prior knowledge in the form of a prior distribution would become important in the MAP estimate for minimum Bayes risk. This type of prior is often called an informative prior. Intuitively, if we know what the parameter values are likely to be before observations are made, we may be able to make good use of the data, which may be limited, to obtain a good model estimate. If this is true, the questions remaining are how to derive the informative prior and how to use it in obtaining the MAP estimate.

For mathematical tractability, conjugate priors are often used in Bayesian adaptation. A conjugate prior for a random vector is defined as the prior distribution for the parameters of the probability density function of the random vector, such that the posterior distribution $P(\lambda|\mathbf{O})$ and the prior distribution $P(\lambda)$ belong to the same distribution family for any sample observations $\mathbf{O}$. For example, it is well known that the conjugate prior for the mean of a Gaussian density is also a Gaussian density. In the following, we therefore discuss only the use of conjugate priors. We also discuss only the case of Bayesian adaptation of the Gaussian mean as it is sufficient to demonstrate the idea of Bayesian adaptation in dealing with small training set problems.

Ex.1022 / Page 225 of 333
AMAZON.COM, INC.

Let us focus on a Gaussian mixture component $\mathcal{N}(\mu, \sigma^2)$ in a mixture density HMM. We use one-dimensional observations for simplicity. Assume the mean μ is random with prior distribution $P_0(\mu)$ and the variance σ^2 is known and fixed. It can be shown that the conjugate prior for μ is also Gaussian; that is, if we assume $P_0(\mu)$ to be the conjugate prior of μ, then $P_0(\mu)$ is Gaussian. Thus, let us denote the mean and variance of the prior for μ by ρ and τ^2, respectively. The MAP estimate for the mean parameter μ in Bayesian adaptation, from a set of n training observations, is given by

$$\hat{\mu}_{\text{MAP}} = \frac{n\tau^2}{\sigma^2 + n\tau^2}\bar{\mathbf{o}} + \frac{\sigma^2}{\sigma^2 + n\tau^2}\rho \tag{6.118}$$

where $\bar{\mathbf{o}}$ is the sample mean of the n training data. The interpretation of Eq. (6.118) is as follows. If there are no training data presented, $n = 0$ and the best estimate of μ is simply the mean ρ of the prior distribution of the μ parameter. When training data are collected and used, the MAP estimate becomes a weighted average of the prior mean ρ and the sample mean of the presented observations, $\bar{\mathbf{o}}$. Ultimately, $n \to \infty$ and the best estimate of μ is, as expected, the sample mean $\bar{\mathbf{o}}$. It should also be noted that if the prior variance τ^2 is much larger than σ^2/n, the MAP estimate in Eq. (6.118) is essentially the ML estimate, $\bar{\mathbf{o}}$, which corresponds to the case of using noninformative priors.

A key question is, How do we determine ρ and τ^2? In practice, these prior parameters have to be estimated from a collection of speaker-dependent (or multispeaker) models, or from a speaker-independent model with mixture distributions in each state. For example, ρ and τ^2 can be estimated by

$$\rho = \sum_{m=1}^{M} c_m \rho_m \tag{6.119a}$$

and

$$\tau^2 = \sum_{m=1}^{M} c_m (\rho_m - \rho)^2 \tag{6.119b}$$

where c_m is the weight assigned to the m^{th} model (or the m^{th} mixture component in the corresponding state of a mixture density speaker-independent HMM) and ρ_m is the mean of the corresponding m^{th} model (or mixture component). When using a speaker-independent Gaussian mixture HMM, the weight c_m is basically the mixture gain for the m^{th} mixture component, and the estimates of Eq. (6.119) are the ML estimates of the mean and variance parameters of μ before any speaker-specific training data are observed.

The concept of Bayesian adaptation based on conjugate priors can be applied to other parameters as well. The adaptation method can be shown to provide good parameter estimates even when the number of speaker-specific training tokens is extremely limited. Experiments have shown that large improvements in recognition accuracy are obtained with the Bayesian adaptation method, compared to direct training, particularly when only a small number of training tokens are available [30].

6.13.3 Corrective Training

In statistical pattern recognition, the minimum Bayes' risk is the theoretical recognizer performance bound, conditioned on the exact knowledge of both the class prior $P(v)$ and the conditional distributions $P(\mathbf{O}|v)$. When both distributions are not known exactly, and the classifier needs to be designed based on a finite training set, there are several ways to try to reduce the error rate. One method is based on the theoretical link between discriminant analysis and distribution estimation [32]. The idea is to design a classifier (discriminant function) such that the minimum classification error rate is attained on the training set. In particular, we wish to design a classifier that uses estimates of $P(v)$ and $P(\mathbf{O}|v)$, and that achieves a minimum error rate for the training set in the same way a discriminant function is designed. The reason for using the HMM, $P(\mathbf{O}|\lambda_v)$, for modeling $P(\mathbf{O}|v)$, as opposed to other discriminant functions, is to exploit the strengths of the HMMs—consistency, flexibility and computational ease.

Bahl et al. [33] were the first to propose an error-correction strategy, which they named corrective training, to specifically deal with the misclassification problem. Their training algorithm was motivated by analogy with an error-correction training procedure for linear classifiers. In their proposed method, the observation distribution is of a discrete type, $B = [b_i(k)]$, where $b_i(k)$ is the probability of observing a vector quantization code index (acoustic label) k when the HMM source is in state i. Each $b_i(k)$ is obtained via the forward-backward algorithm as the weighted frequency of occurrence of the code index. The corrective training algorithm of Bahl et al. works as follows. First, use a labeled training set to estimate the parameters of the HMMs $\Lambda = \{\lambda_v\}$ with the forward-backward algorithm. For each utterance $\mathbf{O}$, labeled as v, for example, evaluate $P(\mathbf{O}|\lambda_v)$ for the correct class v and $P(\mathbf{O}|\lambda_w)$ for each incorrect class w. (The evaluation of likelihood for the incorrect classes need not be exhaustive.) For every utterance where $\log P(\mathbf{O}|\lambda_w) > \log P(\mathbf{O}|\lambda_v) - \Delta$, where Δ is a prescribed threshold, modify λ_v and λ_w according to the following mechanism: (1) Apply the forward-backward algorithm to obtain estimates $b_i'(k)$ and $b_i''(k)$, using the labeled utterance $\mathbf{O}$ only, for the correct class v and incorrect class w, respectively; (2) Modify the original $b_i(k)$ in λ_v to $b_i(k) + \gamma b_i'(k)$ and the $b_i(k)$ in λ_w to $b_i(k) - \gamma b_i''(k)$. When the state labels are tied for certain models, the above procedure is equivalent to replacing the original $b_i(k)$ by $b_i(k) + \gamma(b_i'(k) - b_i''(k))$. The prescribed adaptation parameter, γ, controls the "rate of convergence" and the threshold, Δ, defines the "near-miss" cases. This corrective training algorithm therefore focuses on those parts of the model that are most important for word discrimination, a clear difference from the maximum likelihood principle.

Bahl et al. reported that the corrective training procedure worked better (in isolated word-recognition tasks) than models obtained using the maximum mutual information or the conditional maximum likelihood criterion. The method, however, is primarily experimental.

Several other forms of discriminative training were also proposed by Katagiri et al. [34] together with a framework for the analysis of related training/learning ideas for minimizing recognition errors. The discriminative training method described in Sec. 5.6.3

can be applied to HMM training without difficulty. The corrective training algorithm of Bahl et al. can be shown to be just one possible choice for the minimization of a prescribed risk function.

6.14 MODEL CLUSTERING AND SPLITTING

One of the basic assumptions in statistical modeling is that the variability in the observations from an information source can be modeled by the assumed statistical distributions. For speech recognition, the source could be a single word, a subword unit like a phoneme, or a word sequence. Because of variability in the production (e.g., accents, speed of talking), or the processing (e.g., transmission distortion, noise), it is often expedient to consider using more than a single HMM to characterize the source. There are two motivations behind this multiple HMM approach. First, lumping together all the variability from inhomogeneous data sources leads to unnecessarily complex models, often yielding lower modeling accuracy. Second, some of the variability, or rather the inhomogeneity in the source data, may be known *a priori*, thus warranting separate modeling of the source data sets.

Several generalized clustering algorithms exist, such as the k-means clustering algorithm, the generalized Lloyd algorithm as is widely used in vector quantizer designs [35], and the greedy growing algorithm found in set partition or decision tree designs [36], all of which are suitable for the purpose of separating inconsistent training data so that each divided subgroup becomes more homogeneous and therefore is better modeled by a single HMM. The nearest-neighbor rule required in these clustering algorithms is simply to assign an observation sequence $\mathbf{O}$ to cluster i if

$$P(\mathbf{O}|\lambda_i) = \max_j P(\mathbf{O}|\lambda_j) \tag{6.120}$$

where λ_js denote the models of the clusters. Successful application of the model clustering algorithms to the speech-recognition problem, using the straightforward maximum likelihood criterion, has been reported.

An alternative to model clustering is to arbitrarily subdivide a given speech source into a large number of subclasses with specialized characteristics and then consider a generalized procedure for model merging based on source likelihood considerations. By way of example, for large vocabulary speech recognition we often try to build specialized units (context sensitive) for recognition. For example, we could consider building units that are a function of the sound immediately preceding the unit (left-context) and the sound immediately following the unit (right-context). There are on the order of 10,000 such units in English. Many of the units are functionally almost identical. The problem is how to determine which pairs of units should be merged (so that the number of model units is made more manageable and the variance of the parameter estimate is reduced). To get ideas, consider two distinct models, λ_a and λ_b, corresponding to training observation sets $\mathbf{O}_a$ and $\mathbf{O}_b$, and the merged model λ_{a+b}, corresponding to the merged observation sets $\{\mathbf{O}_a, \mathbf{O}_b\}$. We can then compute the change in entropy (i.e., loss of information) resulting from the

merged model as

$$\begin{aligned}\Delta H_{ab} &= H_a + H_b - H_{a+b} \\ &= -P(\mathbf{O}_a|\lambda_a)\log P(\mathbf{O}_a|\lambda_a) - P(\mathbf{O}_b|\lambda_b)\log P(\mathbf{O}_b|\lambda_b) \\ &\quad + P(\{\mathbf{O}_a, \mathbf{O}_b\}|\lambda_{a+b})\log P(\{\mathbf{O}_a, \mathbf{O}_b\}|\lambda_{a+b}).\end{aligned} \tag{6.121}$$

Whenever ΔH_{ab} is small enough, it means that the change in entropy resulting from merging the models will not affect system performance (at least on the training set) and the models can be merged. The question of how small is acceptable is dependent on specific applications. This model merging technique has been used successfully by Lee [37] to create a generalized set of triphone models for large vocabulary speech recognition.

6.15 HMM SYSTEM FOR ISOLATED WORD RECOGNITION

To illustrate the techniques discussed in this chapter, consider using HMMs to build an isolated word recognizer ([38]). Assume we have a vocabulary of V words to be recognized and that each word is to be modeled by a distinct HMM. Further assume, for simplicity of notation, that for each word in the vocabulary we have a training set of K utterances of the word (spoken by one or more talkers) where each utterance constitutes an observation sequence, of some appropriate representation of the (spectral and/or temporal) characteristics of the word. To do isolated word speech recognition, we must perform the following:

1. For each word v in the vocabulary, we must build an HMM λ_v—that is, we must estimate the model parameters (A, B, π) that optimize the likelihood of the training set observation vectors for the vth word.
2. For each unknown word to be recognized, the processing shown in Figure 6.13 must be carried out, namely, measurement of the observation sequence $\mathbf{O} = \{\mathbf{o}_1\, \mathbf{o}_2 \ldots \mathbf{o}_T\}$, via a feature analysis of the speech corresponding to the word; followed by calculation of model likelihoods for all possible models, $P(\mathbf{O}|\lambda_v)$, $1 \leq v \leq V$; followed by selection of the word whose model likelihood is highest—that is,

$$v^* = \arg \max_{1 \leq v \leq V} [P(\mathbf{O}|\lambda_v)]. \tag{6.122}$$

The probability computation step is generally performed using the Viterbi algorithm (i.e., the maximum likelihood path is used) and requires on the order of $V \cdot N^2 \cdot T$ computations. For modest vocabulary sizes, e.g., $V = 100$ words, with an $N = 5$ state model, and $T = 40$ observations for the unknown word, a total of 10^5 computations is required for recognition (where each computation is a multiply, and add, and a calculation of observation density, $b(\mathbf{o})$). Clearly this amount of computation is modest compared to the capabilities of most modern signal processor chips.

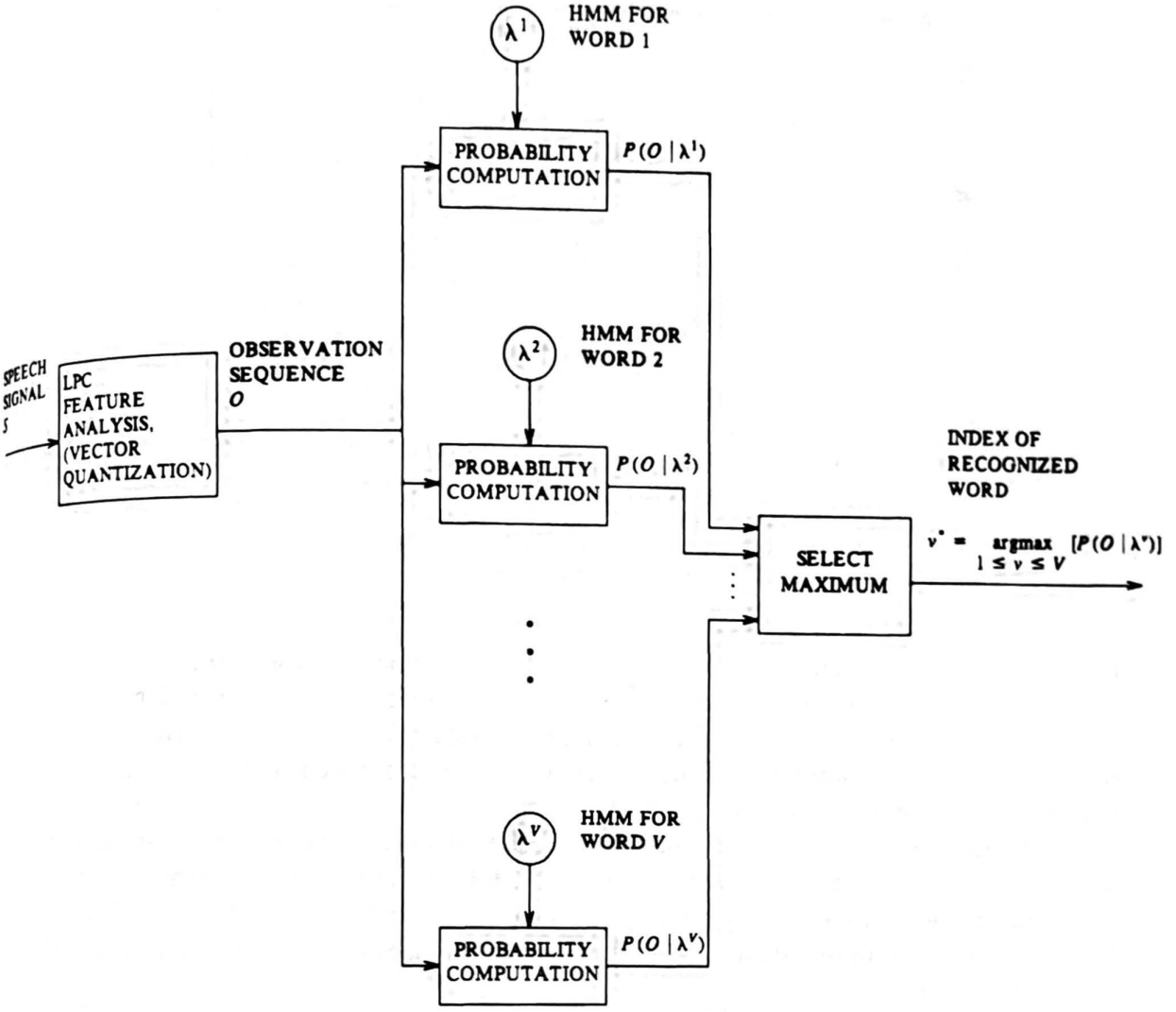


Figure 6.13 Block diagram of an isolated word HMM recognizer (after Rabiner [38]).

6.15.1 Choice of Model Parameters

We now return to the issue that we have raised several times in this chapter—namely, How do we select the type of model, and how do we choose the parameters of the selected model? For isolated word recognition with a distinct HMM designed for each word in the vocabulary, it should be clear that a left-right model is more appropriate than an ergodic model, since we can then associate time with model states in a fairly straightforward manner. Furthermore, we can envision the physical meaning of the model states as distinct sounds (e.g., phonemes, syllables) of the word being modeled.

The issue of the number of states to use in each word model leads to two schools of thought. One idea is to let the number of states correspond roughly to the number of sounds (phonemes) within the word—hence, models with from 2 to 10 states would be appropriate [18]. The other idea is to let the number of states correspond roughly to the average number of observations in a spoken version of the word, the so-called Bakis model

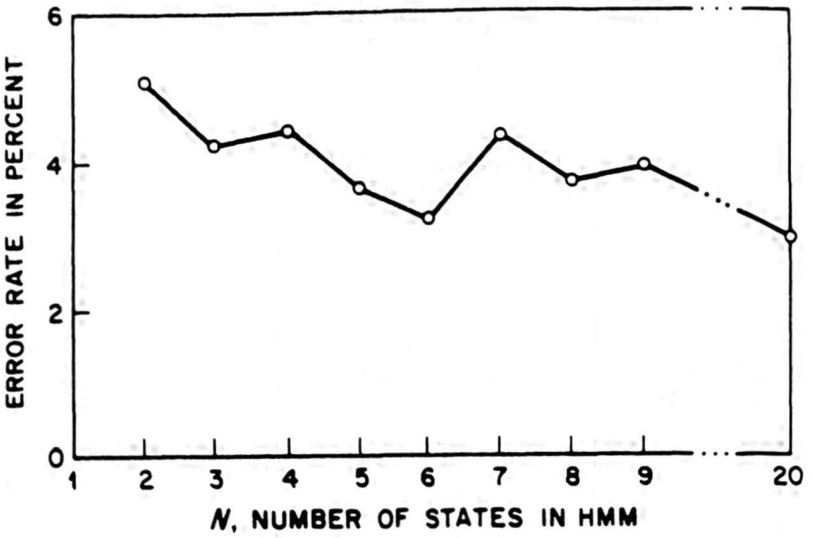


Figure 6.14 Average word error rate (for a digits vocabulary) versus the number of states N in the HMM (after Rabiner et al. [18]).

[11]. In this manner each state corresponds to an observation interval—i.e., about 10–15 ms for the standard methods of analysis. In the results to be described later in this section, we use the former approach. Furthermore, we restrict each word model to have the same number of states; this implies that the models will work best when they represent words with the same number of sounds.

To illustrate the effect of varying the number of states in a word model, Figure 6.14 shows a plot of average word error rate versus N, for the case of recognition of isolated digits (i.e., a 10-word vocabulary). It can be seen that the error is somewhat insensitive to N, achieving a local minimum at $N = 6$; however, differences in error rate for values of N close to 6 are small.

The next issue is the choice of observation vector and the way it is represented. As discussed in Chapter 3, possibilities include LPC-derived weighted cepstral coefficients and weighted cepstral derivatives or (for autoregressive HMMs) the autocorrelation coefficients as the observation vectors for continuous models; for discrete symbol models we use a codebook to generate the discrete symbols. For the continuous models we use as many as $M = 64 \sim 256$ mixtures per state; for the discrete symbol models we use codebooks with as many as $M = 512 \sim 1024$ code words. Also, for the continuous models, it has been found that it is more convenient and sometimes preferable to use diagonal covariance matrices with several mixtures, rather than fewer mixtures with full covariance matrices. The reason for this is simple—namely, the difficulty in performing reliable reestimation of the off diagonal components of the covariance matrix from the necessarily limited training data. Figure 6.15 illustrates the need for using mixture densities for modeling observation vectors (eighth-order cepstral vectors derived from LPC with log energy appended as the ninth vector component). Figure 6.15 shows a comparison of marginal distributions $b_j(\mathbf{o})$ against a histogram of the actual observations within a state (as determined by a maximum likelihood segmentation of all the training observations into states). The observation vectors

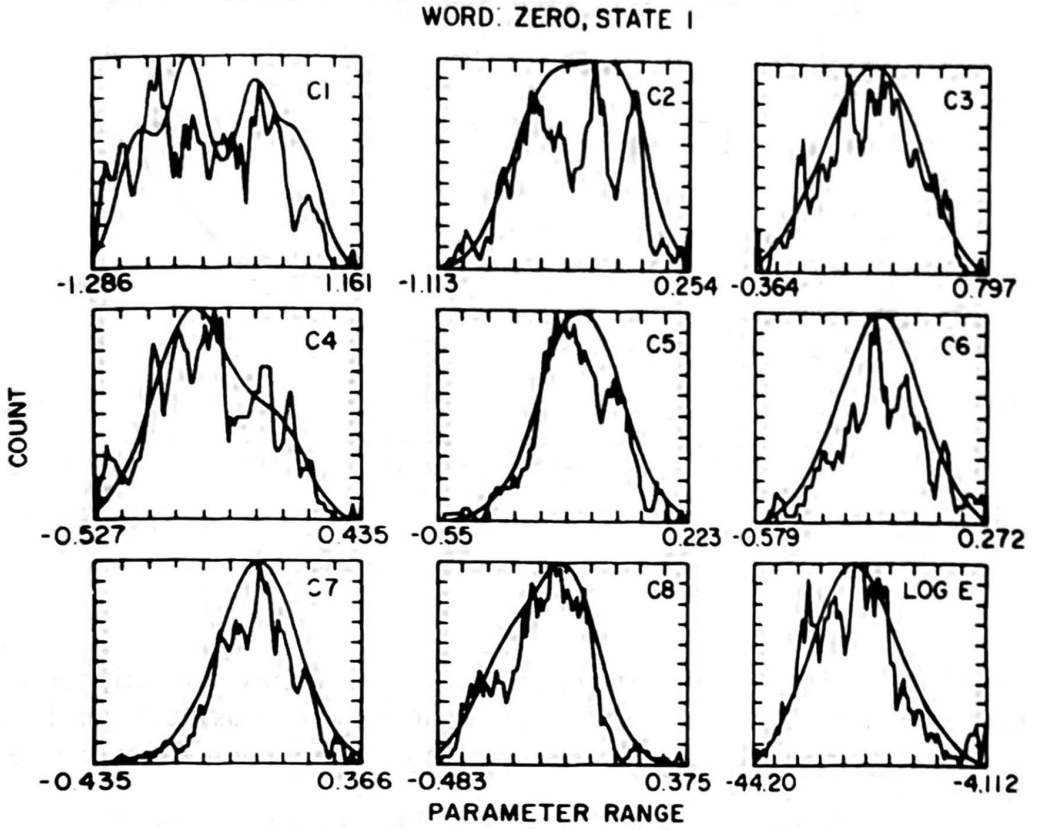


Figure 6.15 Comparison of estimated density (jagged contour) and model density (smooth contour) for each of the nine components of the observation vector (eight cepstral components, one log energy component) for state 1 of the digit zero (after Rabiner et al. [38]).

are ninth order, and the model density uses $M = 5$ mixtures. The covariance matrices are constrained to be diagonal for each individual mixture. The results of Figure 6.15 are for the first model state of the word "zero." The need for values of $M > 1$ is clearly seen in the histogram of the first parameter (the first cepstral component) which is inherently multimodal; similarly, the second, fourth, and eighth cepstral parameters show the need for more than a single Gaussian component to provide good fits to the empirical data. Many of the other parameters appear to be well fitted by a single Gaussian; in some cases, however, even $M = 5$ mixtures do not provide a sufficiently good fit.

Another experimentally verified fact about the HMM is that it is important to limit some of the parameter estimates to prevent them from becoming too small. For example, for the discrete symbol models, the constraint that $b_j(k)$ be greater than or equal to some minimum value ϵ is necessary to ensure that even when the kth symbol never occurred in some state j in the training observation set, there is always a finite probability of its occurrence when scoring an unknown observation set. To illustrate this point, Figure 6.16 shows a curve of average word error rate versus the parameter ϵ (on a log scale) for a standard word-recognition experiment. It can be seen that over a very broad range ($10^{-10} \leq \epsilon \leq 10^{-3}$) the average error rate remains at about a constant value; however,

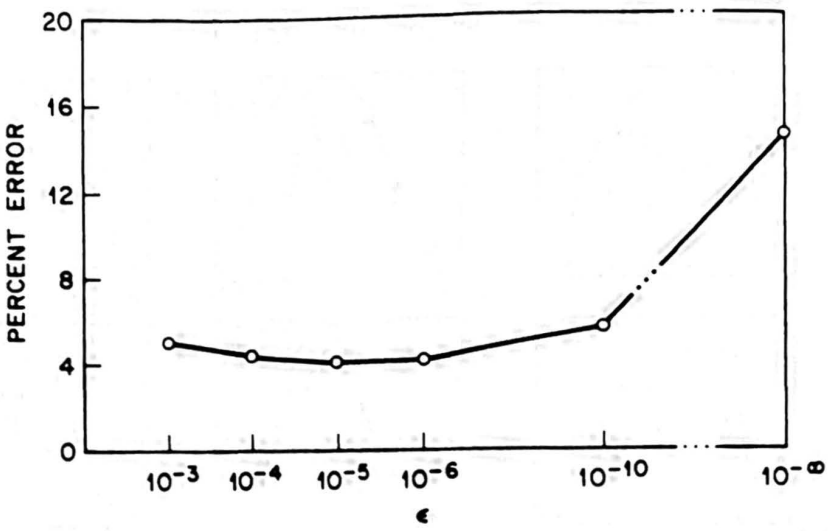


Figure 6.16 Average word error rate as a function of the minimum discrete density value ϵ (after Rabiner et al. [18]).

when ϵ is set to 0 (i.e., $10^{-\infty}$), then the error rate increases sharply. Similarly, for continuous densities it is important to constrain the mixture gains c_{jm} as well as the diagonal covariance coefficients $U_{jm}(r, r)$ to be greater than or equal to some minimum values (we use 10^{-4} in all cases) [18].

6.15.2 Segmental K-Means Segmentation into States

In this chapter, we have emphasized that good initial estimates of the parameters of the $b_j(\mathbf{o}_t)$ densities are essential for rapid and proper convergence of the reestimation formulas. Hence a procedure for providing good initial estimates of these parameters was devised and is shown in Figure 6.17. The training procedure is a variant on the well-known K-means iterative procedure for clustering data.

We assume that we have a training set of observations (the same as is required for parameter reestimation), and an initial estimate of all model parameters. However, unlike the one required for reestimation, the initial model estimate can be chosen randomly or on the basis of any available model appropriate to the data.

Following model initialization, the set of training observation sequences is segmented into states, based on the current model λ. This segmentation is achieved by finding the optimum state sequence, via the Viterbi algorithm, and then backtracking along the optimal path. This procedure is illustrated in Figure 6.18, which shows a log-energy plot, an accumulated log-likelihood plot, and a state segmentation for one occurrence of the word "six." The states correspond roughly to the sounds in the spoken word "six."

The result of segmenting each of the training sequences is, for each of the N states, a maximum likelihood estimate of the set of the observations that occur within each state j according to the current model. In the case where we are using discrete symbol densities, each of the observation vectors within a state is coded using the M-code-word codebook,

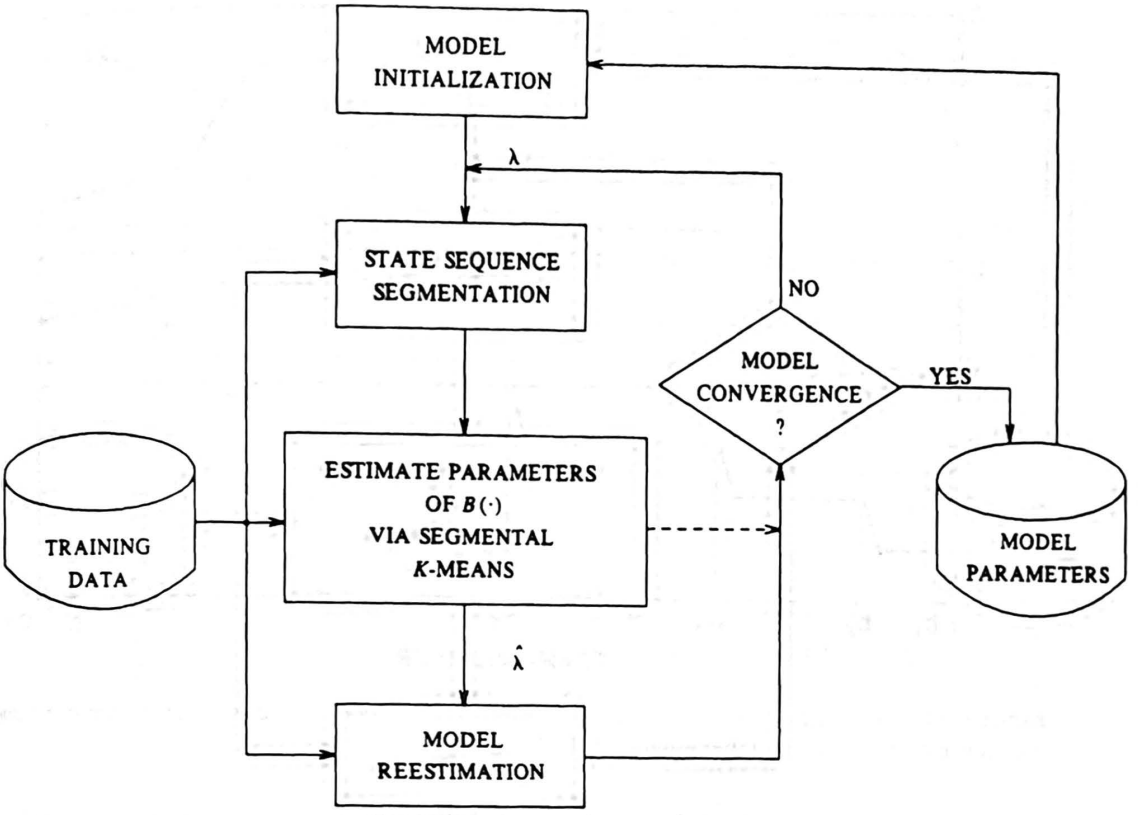


Figure 6.17 The segmental k-means training procedure used to estimate parameter values for the optimal continuous mixture density fit to a finite number of observation sequences (after Rabiner et al. [38]).

and the updated estimate of the $b_j(k)$ parameters is

$$\hat{b}_j(k) = \text{number of vectors with codebook index } k \text{ in state } j \text{ divided by the number of vectors in state } j.$$

When we are using continuous observation densities, a segmental K-means procedure is used to cluster the observation vectors within each state j into a set of M clusters (using a Euclidean distortion measure), where each cluster represents one of the M mixtures of the $b_j(\mathbf{o}_t)$ density. From the clustering, an updated set of model parameters is derived as follows:

$$\begin{aligned}
\hat{c}_{jm} &= \text{number of vectors classified in cluster } m \text{ of state } j \text{ divided by the number of vectors in state } j \\
\hat{\boldsymbol{\mu}}_{jm} &= \text{sample mean of the vectors classified in cluster } m \text{ of state } j \\
\hat{\mathbf{U}}_{jm} &= \text{sample covariance matrix of the vectors classified in cluster } m \text{ of state } j.
\end{aligned}$$

Based on this state segmentation, updated estimates of the a_{ij} coefficients can be obtained by counting the number of transitions from state i to j and dividing it by the number of transitions from state i to any state (including itself).

An updated model $\hat{\lambda}$ is obtained from the new model parameters, and the formal reestimation procedure is used to reestimate all model parameters. The resulting model is then compared to the previous model (by computing a distance score that reflects the

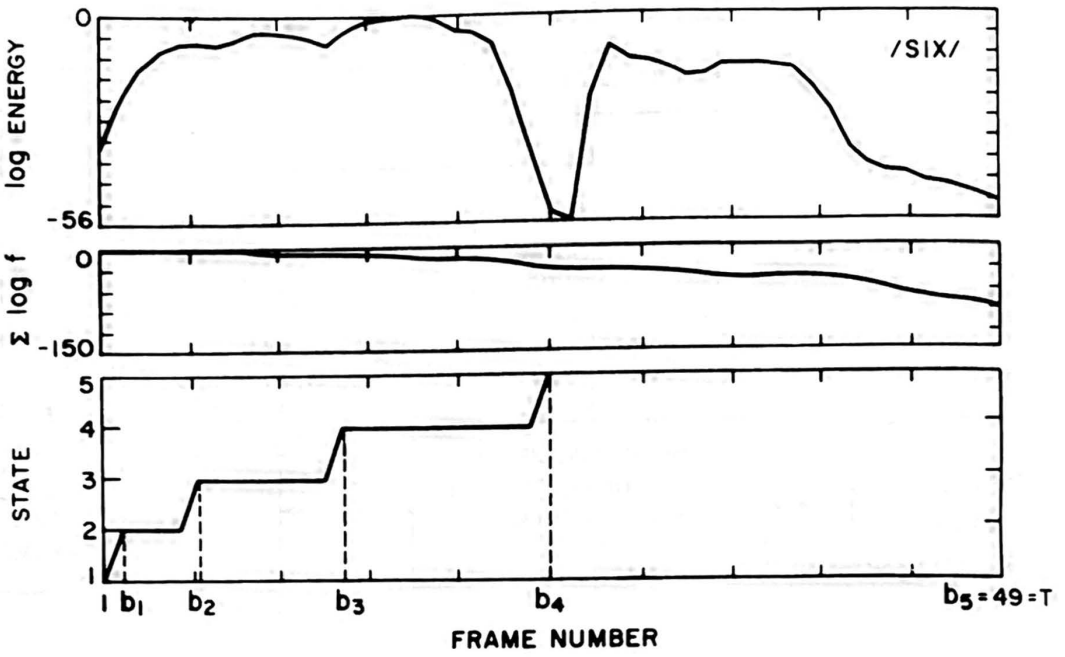


Figure 6.18 Plots of (a) log energy; (b) accumulated log likelihood; and (c) state assignment for one occurrence of the word "six" (after Rabiner et al. [38]).

statistical similarity of the HMMs). If the model distance score exceeds a threshold, then the old model λ is replaced by the new (reestimated) model $\bar{\lambda}$, and the overall training loop is repeated. If the model distance score falls below the threshold, then model convergence is assumed and the final model parameters are saved.

6.15.3 Incorporation of State Duration into the HMM

In Section 6.9 we discussed the theoretically correct method of incorporating state duration information into the mechanics of the HMM ([39]). We also showed that the cost of including duration density was rather high; namely a D^2-fold increase in computation and a D-fold increase in storage. Using a value of $D = 25$ (as is required for word recognition), the cost of the increased computation tended to make the techniques not worth using. Thus the following alternative procedure was formulated for incorporating state duration information into the HMM.

For this alternative procedure, the state duration probability $p_j(d)$ was measured directly from the segmented training sequences used in the segmental K-means procedure of the previous section. Hence the estimates of $p_j(d)$ are strictly heuristic ones. A typical set of histograms of $p_j(d)$ for a five-state model of the word "six" is shown in Figure 6.19. (In this figure the histograms are plotted versus normalized duration (d/T), rather than absolute duration d.) The first two states account for the initial $/s/$ in "six"; the third state accounts for the transition to the vowel $/i/$; the fourth state accounts for the vowel; and the fifth state accounts for the stop and the final $/s/$ sound.

The way in which the heuristic duration densities were used in the recognizer was

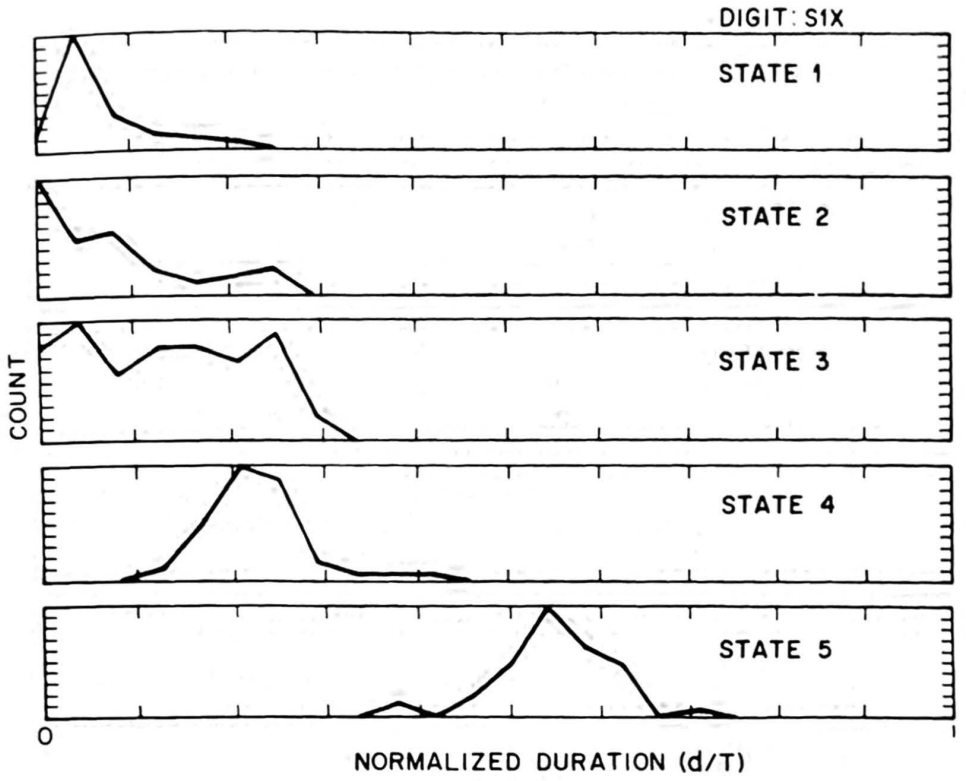


Figure 6.19 Histograms of the normalized duration density for the five states of the digit "six" (after Rabiner et al. [38]).

as follows. First the normal Viterbi algorithm is used to give the best segmentation of the observation sequence of the unknown word into states via a backtracking procedure. The duration of each state is then measured from the state segmentation. A postprocessor then increments the log-likelihood score of the Viterbi algorithm, by the quantity

$$\log \hat{P}(\mathbf{q}, \mathbf{O}|\lambda) = \log P(\mathbf{q}, \mathbf{O}|\lambda) + \alpha_d \sum_{j=1}^{N} \log [p_j(d_j)] \tag{6.123}$$

where α_d is a scaling multiplier on the state duration scores, and d_j is the duration of state j along the optimal path as determined by the Viterbi algorithm. The incremental cost of the postprocessor for duration is essentially negligible, and experience has shown that recognition performance is essentially as good as that obtained using the theoretically correct duration model.

6.15.4 HMM Isolated-Digit Performance

We conclude this section on isolated word recognition using HMMs by giving a set of performance results (in terms of average word error rate) on the task of recognizing isolated digits in a speaker-independent manner. For this task, a training set consisting of 100 occurrences of each digit by 100 talkers (i.e., a single occurrence of each digit per talker) was used. Half the talkers were male and half were female. For testing the

TABLE 6.1. Average Digit Error Rates for Several Recognizers and Evaluation Sets

Recognizer Type	Evaluation Set			
	Original Training	TS2	TS3	TS4
LPC/DTW	0.1	0.2	2.0	1.1
LPC/DTW/VQ	–	3.5	–	–
HMM/VQ	–	3.7	–	–
HMM/CD	0	0.2	1.3	1.8
HMM/AR	0.3	1.8	3.4	4.1

algorithm, we used the initial training set, as well as three other independent test sets with the following characteristics:

TS2 The same 100 talkers as were used in the training; 100 occurrences of each digit

TS3 A new set of 100 talkers (50 male, 50 female); 100 occurrences of each digit

TS4 Another new set of 100 talkers (50 male, 50 female); 100 occurrences of each digit

The results of the recognition tests are given in Table 6.1. The recognizers are the following:

LPC/DTW Conventional template-based recognizer using dynamic time warping (DTW) alignment

LPC/DTW/VQ Conventional recognizer with vector quantization of the feature vectors ($M = 64$)

HMM/VQ HMM recognizer with $M = 64$ codebook

HMM/CD HMM recognizer using continuous density model with $M = 5$ mixtures per state

HMM/AR HMM recognizer using autoregressive observation density

It can be seen that, when using a VQ, the performance of the isolated word recognizer degrades in both the conventional and HMM modes. It can also be seen that the performances of the conventional template-based recognizer, and the HMM recognizer with a continuous density model are comparable. Finally Table 6.1 shows that the autoregressive density HMM gives poorer performance than the standard mixture density model.

6.16 SUMMARY

In this chapter we have attempted to present the theory of hidden Markov models from the simplest concepts (discrete Markov chains) to the most sophisticated models (variable

duration, continuous density models). It has been our purpose to focus on physical explanations of the basic mathematics; hence we have avoided long, drawn-out proofs or derivations of the key results, and concentrated primarily on trying to interpret the meaning of the math, and how it could be implemented in practice in real-world systems. We have also attempted to illustrate one application of the theory of HMMs to a simple problem in speech recognition—namely, isolated word recognition.

REFERENCES

[1] L.E. Baum and T. Petrie, "Statistical inference for probabilistic functions of finite state Markov chains," *Ann. Math. Stat.*, 37: 1554-1563, 1966.

[2] L.E. Baum and J.A. Egon, "An inequality with applications to statistical estimation for probabilistic functions of a Markov process and to a model for ecology," *Bull. Amer. Meteorol. Soc.*, 73: 360-363, 1967.

[3] L.E. Baum and G.R. Sell, "Growth functions for transformations on manifolds," *Pac. J. Math.*, 27 (2): 211-227, 1968.

[4] L.E. Baum, T. Petrie, G. Soules, and N. Weiss, "A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains," *Ann. Math. Stat.*, 41 (1): 164-171, 1970.

[5] L.E. Baum, "An inequality and associated maximization technique in statistical estimation for probabilistic functions of Markov processes," *Inequalities*, 3: 1-8, 1972.

[6] J.K. Baker, "The dragon system—An overview," *IEEE Trans. Acoustics, Speech, Signal Proc.*, ASSP-23 (1): 24-29, February 1975.

[7] F. Jelinek, "A fast sequential decoding algorithm using a stack," *IBM J. Res. Develop.*, 13: 675-685, 1969.

[8] L.R. Bahl and F. Jelinek, "Decoding for channels with insertions, deletions, and substitutions with applications to speech recognition," *IEEE Trans. Information Theory*, IT-21: 404-411, 1975.

[9] F. Jelinek, L.R. Bahl, and R.L. Mercer, "Design of a linguistic statistical decoder for the recognition of continuous speech," *IEEE Trans. Information Theory*, IT-21: 250-256, 1975.

[10] F. Jelinek, "Continuous speech recognition by statistical methods," *Proc. IEEE*, 64: 532-536, April 1976.

[11] R. Bakis, "Continuous speech word recognition via centisecond acoustic states," in *Proc. ASA Meeting* (Washington, DC), April 1976.

[12] F. Jelinek, L.R. Bahl, and R.L. Mercer, "Continuous speech recognition: Statistical methods," in *Handbook of Statistics, II*, P.R. Krishnaiad, Ed. Amsterdam, The Netherlands: North-Holland, 1982.

[13] L.R. Bahl, F. Jelinek, and R.L. Mercer, "A maximum likelihood approach to continuous speech recognition," *IEEE Trans. Pattern Anal. Machine Intell.*, PAMI-5: 179-190, 1983.

[14] J.D. Ferguson, "Hidden Markov Analysis: An Introduction," in *Hidden Markov Models for Speech*, Institute for Defense Analyses, Princeton, NJ, 1980.

[15] A.J. Viterbi, "Error bounds for convolutional codes and an asymptotically optimal decoding algorithm," *IEEE Trans. Information Theory*, IT-13: 260-269, April 1967.

[16] G.D. Forney, "The Viterbi algorithm," *Proc. IEEE*, 61: 268-278, March 1973.

[17] A.P. Dempster, N.M. Laird, and D.B. Rubin, "Maximum likelihood from incomplete data via the EM algorithm," *J. Roy. Stat. Soc.*, 39 (1): 1-38, 1977.

[18] S.E. Levinson, L.R. Rabiner, and M.M. Sondhi, "An Introduction to the Application of the Theory of Probabilistic Functions of a Markov Process to Automatic Speech Recognition," *Bell System Tech. J.*, 62 (4): 1035–1074, April 1983.

[19] L.A. Liporace, "Maximum likelihood estimation for multivariate observations of Markov sources," *IEEE Trans. Information Theory*, IT-28 (5): 729-734, 1982.

[20] B.H. Juang, "Maximum likelihood estimation for mixture multivariate stochastic observations of Markov chains," *AT&T Tech. J.*, 64 (6): 1235-1249, July-Aug. 1985.

[21] B.H. Juang, S.E. Levinson, and M.M. Sondhi, "Maximum likelihood estimation for multivariate mixture observations of Markov chains," *IEEE Trans. Information Theory*, IT-32 (2): 307-309, March 1986.

[22] A.B. Poritz, "Linear predictive hidden Markov models and the speech signal," in *Proc. ICASSP 82* (Paris, France), 1291-1294, May 1982.

[23] B.H. Juang and L.R. Rabiner, "Mixture autoregressive hidden Markov models for speech signals," *IEEE Trans. Acoustics, Speech, Signal Proc.*, ASSP-33 (6): 1404-1413, December 1985.

[24] M J. Russell and R.K. Moore, "Explicit modeling of state occupancy in hidden Markov models for automatic speech recognition," in *Proc. ICASSP 85* (Tampa, FL), 5-8, March 1985.

[25] S.E. Levinson, "Continuously variable duration hidden Markov models for automatic speech recognition," *Computer Speech and Language*, 1 (1): 29-45, March 1986.

[26] L.R. Bahl, P.F. Brown, P.V. deSouza, and L.R. Mercer, "Maximum mutual information estimation of hidden Markov model parameters for speech recognition," in *Proc. ICASSP 86* (Tokyo, Japan), 49-52, April 1986.

[27] Y. Ephraim, A. Dembo, and L.R. Rabiner, "A minimum discrimination information approach for hidden Markov modeling," *IEEE Trans. Information Theory*, 35 (5): 1001-1003, September 1989.

[28] Y. Ephraim and L. Rabiner, "On the Relations Between Modeling Approaches for Speech Recognition," *IEEE Trans. Information Theory*, 36 (2): 372-380, March 1990.

[29] B.H. Juang and L.R. Rabiner, "A probabilistic distance measure for hidden Markov models," *AT&T Tech. J.*, 64 (2): 391-408, February 1985.

[30] F. Jelinek and R.L. Mercer, "Interpolated estimation of Markov source parameters from sparse data," in *Pattern Recognition in Practice*, E.S. Gelesma and L.N. Kanal, Eds. Amsterdam, The Netherlands: North-Holland, 1980, 381-397.

[31] C.-H. Lee, C.-H. Lin, and B.H. Juang, "A study on speaker adaptation of the parameters of continuous density hidden Markov models," *IEEE Trans. Signal Processing*, 39 (4): 806–841, April 1991.

[32] R.O. Duda and P.E. Hart, *Pattern Classification and Scene Analysis*, John Wiley & Sons, New York, 1973.

[33] L.R. Bahl, P.F. Brown, P.V. deSouza, and R.L. Mercer, "A new algorithm for the estimation of hidden Markov model parameters," *Proc. ICASSP 88*, 493–496, New York, April 1988.

[34] S. Katagiri, C.H. Lee, and B.H. Juang, "New Discriminative Training Algorithms Based on the Generalized Probabilistic Descent Method," *Proc. 1991 Workshop Neural Networks for Signal Processing*, 299–308, IEEE Signal Processing Society, Princeton, September 1991.

Ex.1022 / Page 239 of 333
AMAZON.COM, INC.

[35] Y. Linde, A. Buzo, and R.M. Gray, "An algorithm for vector quantizer design," *IEEE Trans. Communications*, COM-28: 84–95, January 1980.

[36] J. Breiman, J.H. Friedman, R.A. Olshen, and C.J. Stone, *Classification and Regression Trees*, Wadsworth & Brooks, Monterey, CA, 1984.

[37] K.F. Lee, *Automatic Speech Recognition—The Development of the SPHINX System*, Kluwer Academic Publishers, Boston, 1989.

[38] L.R. Rabiner, B.H. Juang, S.E. Levinson, and M.M. Sondhi, "Recognition of Isolated Digits Using Hidden Markov Models With Continuous Mixture Densities," *AT&T Tech. J.*, 64 (6): 1211–1234, July-Aug. 1985.

[39] L.R. Rabiner, "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition," *Proc. IEEE*, 77 (2): 257–286, February 1989.

Chapter 7

SPEECH RECOGNITION BASED ON CONNECTED WORD MODELS

7.1 INTRODUCTION

Up to this point we have been discussing discrete utterance (or, as it is often called, isolated word or phrase) recognition. The assumption was that the speech to be recognized comprised a single word or phrase and was to be recognized as a complete entity with no explicit knowledge or regard for the phonetic content of the word or phrase. Hence, for a vocabulary of V words (or phrases), the recognition algorithm consisted of matching (via time alignment) the measured sequence of spectral vectors of the unknown spoken input against each of the set of spectral patterns for the V words and selecting the pattern whose accumulated time aligned spectral distance was smallest as the recognized word. Another implicit assumption was that each spoken utterance had a clearly defined beginning and ending point that could be found using some type of speech endpoint detector. As a result, pattern matching could be reliably performed without having to be concerned about uncertainties in the endpoints of the patterns being compared. For many applications, notably those referred to as "command-and-control" applications, in which the user is required to speak the command words one at a time (i.e., with distinct pauses between command words), this paradigm works well and is entirely appropriate. (We will discuss command-and-control applications in Chapter 9.) However, for applications in which the speech to be recognized consists of a sequence of words from the recognition vocabulary (e.g., strings of digits) such a paradigm is often unacceptable for practical reasons. Figure 7.1 illustrates this point for a three-digit sequence in which the upper panel shows the log

Ex.1022 / Page 241 of 333
AMAZON.COM, INC.

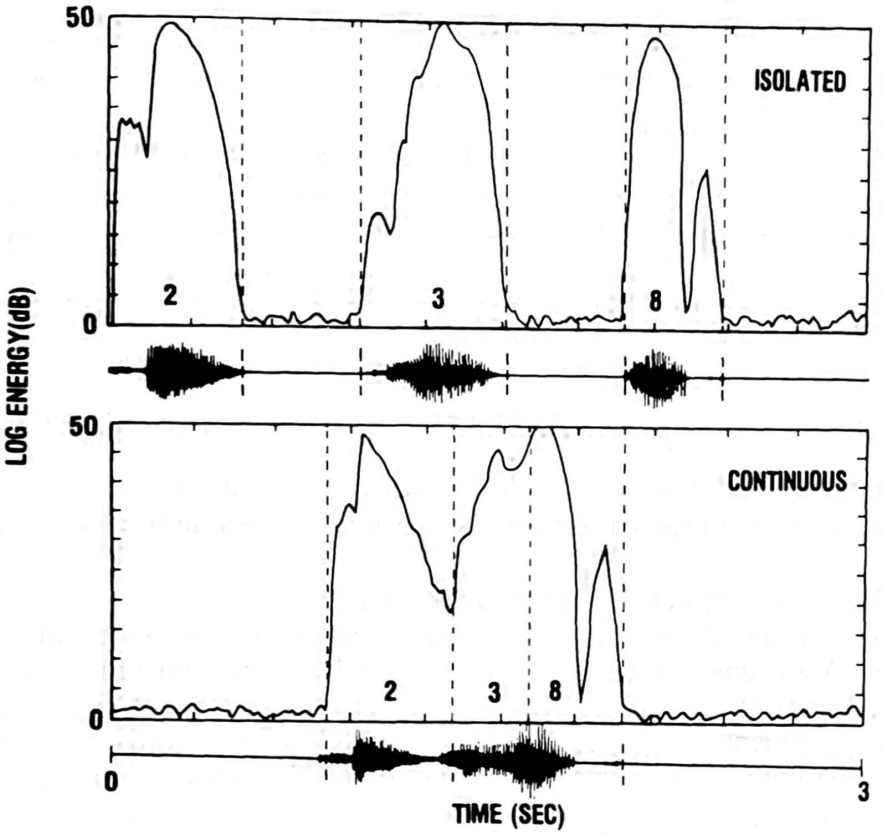


Figure 7.1 Illustration of an isolated string of digits (upper panel) and a fluently spoken version of the same digit string (lower panel).

energy contour of a digit string spoken as a sequence of isolated digits where each digit is followed by a discernible pause (i.e., period of no speaking), and the lower panel shows the same digit string spoken in a fluent (continuous) manner. It should be self-evident that speaking a sequence of isolated digits is unnatural (from the point of view of human speech production) and grossly inefficient (it takes about 2.5 seconds to speak the isolated digit sequence versus less than 1 second to speak the same string of digits in a fluent manner). Hence, it is important to be able to extend the techniques described in earlier chapters of this book so that accurate recognition of fluent speech becomes possible.

From the point of view of speech-recognition algorithms, there are two interesting classes of fluent speech strings. One class is the set of strings derived from small-to-moderate-size vocabularies, including digit strings, spelled letter sequences, combinations of alphanumerics, and strings for accessing limited databases based on small-to-moderate-size vocabularies and highly constrained word syntax. This set has the property that the basic speech-recognition unit can be the word (or phrase), much as in the case of isolated word- (or phrase-) recognition systems. The other class is the set of continuous speech drawn from moderate-to-large-size vocabularies where the basic speech-recognition unit cannot be the word because of complexity constraints. In such a case, subword speech units are necessary for the implementation of the speech-recognition system. We defer a discussion of the second class of speech strings to Chapter 8, where we provide a complete description of techniques used in the recognition of large vocabulary continuous speech. In

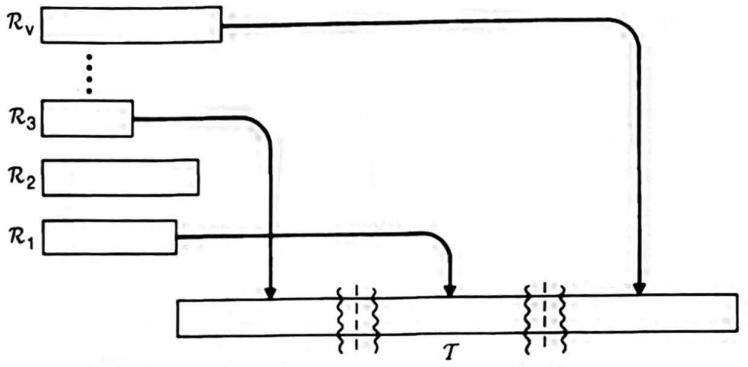


Figure 7.2 Illustration of the connected word-recognition problem.

this chapter we discuss in depth the class of speech-recognition systems commonly referred to as connected word speech recognizers, because the basic unit of recognition is a whole word.

We can now formulate the basic problem in connected word recognition. To do this we refer to Figure 7.2, which gives a pictorial description of the connected word-recognition problem. We assume that we are given the spectral vectors from a fluently spoken string of words, $\mathcal{T} = \{\mathbf{t}(1), \mathbf{t}(2), \ldots, \mathbf{t}(M)\}$, and we are also given the spectral patterns for each of V reference patterns, ($\mathcal{R}_1$ to $\mathcal{R}_V$) corresponding to V unique words in the vocabulary. The connected word-recognition problem can be compactly stated as follows:

> Given a fluently spoken sequence of words, how can we determine the *optimum* match in terms of a concatenation of word reference patterns?

To solve the connected word-recognition problem we must resolve the following problems:

1. We don't usually know the number of words, L, in the string (although we usually have a good bound on the range, e.g., one to seven words).
2. We don't know word utterance boundaries within the spoken string; i.e. except for the beginning of the first word in the string, and the end of the last word in the string, we don't know precisely where any word begins or ends.
3. The word boundaries are often fuzzy or nonunique. That is, it is often difficult, if not impossible, to specify (i.e., find accurately and automatically) the word boundaries because of sound coarticulation. Thus, for example, the boundary between the digit 3 and the digit 8 in Figure 7.1 is fuzzy because the ending sound /i/ in 3 coarticulates strongly with the initial sound /e^y/ in 8. We indicate such fuzziness of the boundaries by the squiggly lines in Figure 7.2.
4. For a set of V word-reference patterns, and for a given value of L (the number of words in the string), there are V^L possible combinations of composite matching patterns (possible L word sequences); for anything but extremely small values of V and L the exponential number of composite matching patterns implies that the connected word-recognition problem *cannot* be solved by exhaustive means.

The bottom line is that a nonexhaustive matching algorithm is required to solve the con-

nected word-recognition problem. In this chapter we discuss several algorithms that have this property and are capable of solving the connected word-recognition problem with various degrees of efficiency.

Although we will spend a great deal of time in this chapter discussing efficient algorithms for optimal matching of sequences of word patterns, it is also important to address the problem of word reference pattern training using connected word sequences. Unlike the isolated word case, for which training procedures were discussed in Chapters 5 and 6, for connected word sequence training we encounter a fundamental difficulty, namely the uncertainty in word boundaries within the spoken word sequence. This makes the connected word sequence training problem somewhat more difficult; therefore in this chapter we also describe an embedded word training method based on the segmental k-means algorithm.

7.2 GENERAL NOTATION FOR THE CONNECTED WORD-RECOGNITION PROBLEM

We denote the spectral sequence of vectors of the test pattern as

$$\mathcal{T} = \{\mathbf{t}(1), \mathbf{t}(2), \ldots, \mathbf{t}(M)\} = \{\mathbf{t}(m)\}_{m=1}^{M} \tag{7.1}$$

in which each $\mathbf{t}(m)$ is an appropriate spectral vector (e.g., filter bank, LPC, ear model, etc.). Similarly, we denote the set of word reference patterns (either templates or statistical models) as $\mathcal{R}_i$, $1 \leq i \leq V$ for a V word vocabulary, where each pattern is of the form

$$\mathcal{R}_i = \{\mathbf{r}_i(1), \mathbf{r}_i(2), \ldots, \mathbf{r}_i(N_i)\} \tag{7.2}$$

where N_i is the duration (in frames or states) of the i^{th} word reference pattern. (For convenience we will develop the algorithms on the basis of using word templates; we will see later in this chapter that the required modifications for statistical models are essentially trivial.)

The connected word-recognition problem can now be stated as a problem in finding the optimum sequence of word reference patterns, $\mathcal{R}^*$, that best matches $\mathcal{T}$. With no loss of generality we assume that there are L word patterns in $\mathcal{R}^*$ (where we vary L from the minimum to the maximum possible values and take the optimum overall values of L). Hence the best sequence of reference patterns, $\mathcal{R}^*$, is a concatenation of L reference patterns, i.e.,

$$\mathcal{R}^* = \{\mathcal{R}_{q^*(1)} \oplus \mathcal{R}_{q^*(2)} \oplus \mathcal{R}_{q^*(3)} \oplus \cdots \oplus \mathcal{R}_{q^*(L)}\} \tag{7.3}$$

in which each index, $q^*(\ell)$ is in the range $[1, V]$.

To determine $\mathcal{R}^*$—that is, to determine the sequence of word indices $q^*(\ell)$, $1 \leq \ell \leq L$, that give the best matching sequence—consider constructing an arbitrary "super-reference" pattern $\mathcal{R}^s$ of the form

$$\mathcal{R}^s = \mathcal{R}_{q(1)} \oplus \mathcal{R}_{q(2)} \oplus \mathcal{R}_{q(3)} \oplus \cdots \oplus \mathcal{R}_{q(L)} = \{\mathbf{r}^s(n)\}_{n=1}^{N^s} \tag{7.4}$$

in which N^s is the total duration of the concatenated reference pattern $\mathcal{R}^s$. The time aligned

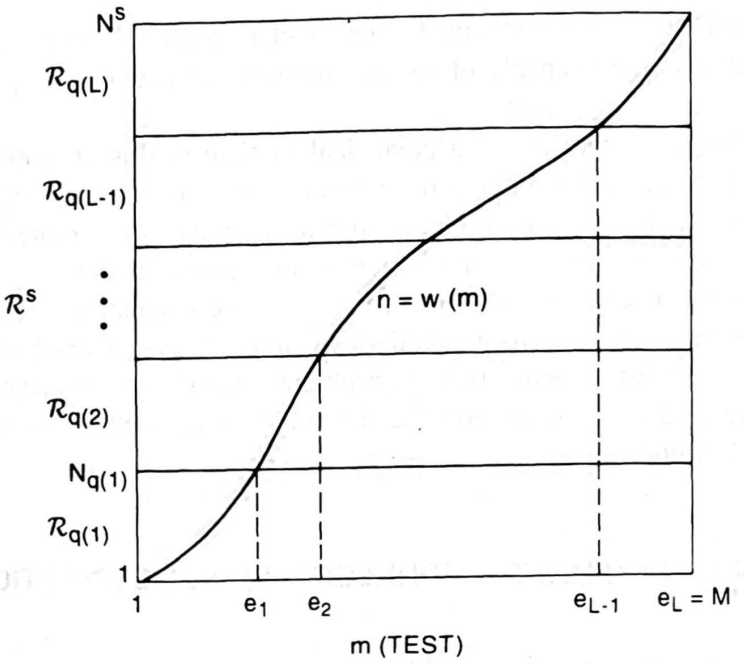


Figure 7.3 Determination of the optimum alignment of super-reference pattern $\mathcal{R}^s$ to $\mathcal{T}$, along with corresponding word boundary frames.

distance between $\mathcal{R}^s$ and $\mathcal{T}$ is readily obtained via dynamic time warping, as illustrated in Figure 7.3. The distance obtained is

$$D(\mathcal{R}^s, \mathcal{T}) = \min_{w(m)} \sum_{m=1}^{M} d(\mathbf{t}(m), \mathbf{r}^s(w(m))) \tag{7.5}$$

in which $d(\cdot, \cdot)$ is a local spectral distance measure, $w(\cdot)$ is a warping function for the time index and the techniques of Chapter 4 can be used to evaluate D. By appropriate path backtracking, as illustrated in Figure 7.3, the word boundary frames in the input string can be found on the basis of the word boundary frames in the super-reference pattern. Hence the last frame in the first reference word, $\mathbf{r}_{q(1)}(N_{q(1)})$ maps to frame e_1 in the test pattern; similarly, the last frame in the second reference word $\mathbf{r}_{q(2)}(N_{q(2)})$ maps to frame e_2 in the test pattern, etc.

To determine the global best match, $\mathcal{R}^*$, we optimize Eq. (7.5) over every possible value of local reference patterns, $q(1), q(2), \ldots, q(L)$, and over every possible value of L, i.e., $L_{\min} \le L \le L_{\max}$, giving

$$\begin{aligned} D^* &= \min_{\mathcal{R}^s} D(\mathcal{R}^s, \mathcal{T}) \\ &= \min_{L_{\min} \le L \le L_{\max}} \min_{\substack{q(1), q(2), \ldots, q(L) \\ 1 \le q(i) \le V}} \min_{w(m)} \sum_{m=1}^{M} d(\mathbf{t}(m), \mathbf{r}^s(w(m))) \end{aligned} \tag{7.6}$$

with

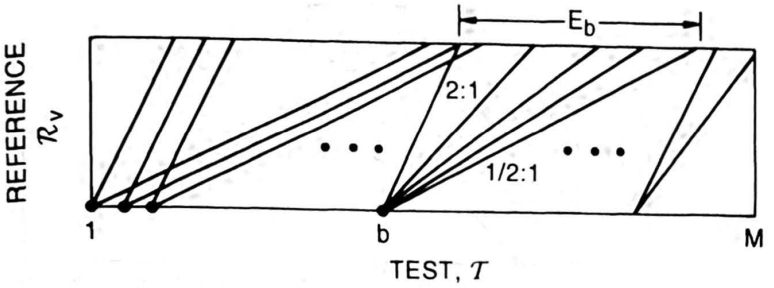


Figure 7.4 Computation ranges for matching $\mathcal{R}_v$ against portions of $\mathcal{T}$.

$$\mathcal{R}^* = \arg\min_{\mathcal{R}^s} D(\mathcal{R}^s, \mathcal{T}). \tag{7.7}$$

For a straightforward evaluation of Eq. (7.6), the computation required is

$$C_L \approx \left[\frac{M \cdot \sum_{\ell=1}^{L} N_{q(\ell)}}{3}\right] V^L = \frac{M \cdot L \cdot \overline{N}}{3} V^L \text{ (grid points)} \tag{7.8}$$

where the bracketed term is the dynamic time warping computation for matching a given $\mathcal{R}^s$ to $\mathcal{T}$, the second term (V^L) is the number of possible combinations in $\mathcal{R}^s$, and $\overline{N}$ is the average number of frames in a word reference pattern. To get a feeling for how impractical the computation of Eq. (7.8) actually is, consider typical values of $M = 300$ frames, $L = 7$ words, $\overline{N} = 40$ frames, and $V = 10$ words. With these values we get $C_L \cong 2.8 \times 10^{11}$ (grid points). This computation (for a single test string) is already excessive for most modern machines. Hence, as discussed previously, alternatives to exhaustive evaluation of D^* are required.

Fortunately, several algorithms have been proposed that solve Eq. (7.6) without the exponential growth in computation. These include

- the two-level dynamic programming approach [1,2]
- the level building approach [3,4]
- the one-stage approach and subsequent generalizations [5–9].

In the next several sections we discuss these individual algorithms.

7.3 THE TWO-LEVEL DYNAMIC PROGRAMMING (TWO-LEVEL DP) ALGORITHM

The basic idea of the two-level DP algorithm is to break up the computation of Eq. (7.6) into two stages (called levels). At the first level the algorithm matches each individual word reference pattern, $\mathcal{R}_v$, against an arbitrary portion of the test string, $\mathcal{T}$. To understand the computation involved, consider Figure 7.4.

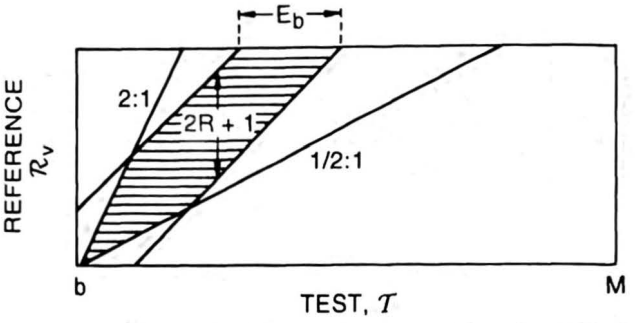


Figure 7.5 Use of range limiting to reduce the size of individual time warps.

For the range of beginning test frames of the match, b, $1 \leq b \leq M$, and for the range of ending test frames, e, $1 \leq e \leq M$ ($e > b$), and for word reference pattern, $\mathcal{R}_v$, $1 \leq v \leq V$, we need to compute

$$\hat{D}(v, b, e) = \min_{w(m)} \sum_{m=b}^{e} d(\mathbf{t}(m), \mathbf{r}_v(w(m))) \tag{7.9}$$

as illustrated in Figure 7.4. (It should be clear from our discussions in Chapter 4 that for each beginning frame b, a single time warp provides the time aligned distance, $\hat{D}$, for a range, E_b, of ending frames e, $b + \Delta \leq e \leq b + \delta$ where Δ and δ are related to N_v, the number of frames in the reference pattern $\mathcal{R}_v$, and the expansion/contraction factor of the DTW algorithm. Hence, using a 2-to-1 expansion and a $^1/_2$ to 1 compression, we would have $b + N_v/2 \leq e \leq b + 2N_v$ as the range on ending frames for beginning frame b. Clearly if $b + 2N_v > M$ then e is truncated at M, the end of the test pattern.)

Equation (7.9) gives the minimum distance for every possible vocabulary pattern, $\mathcal{R}_v$, between each possible pair of beginning and ending frames (b, e). Clearly, we can eliminate v by finding the best (smallest distance) match between b and e for any v, giving

$$\tilde{D}(b, e) = \min_{1 \leq v \leq V} [\hat{D}(v, b, e)] = \text{best score} \tag{7.10}$$

$$\tilde{N}(b, e) = \arg \min_{1 \leq v \leq V} [\hat{D}(v, b, e)] = \text{best reference index} \tag{7.11}$$

thereby significantly reducing the data storage (by a factor of v) with no loss of optimality. We can further reduce computation of Eq. (7.9) by using a range-limited DTW algorithm as illustrated in Figure 7.5. With appropriate choice of the range variable, R, the ending region, E_b, can be made significantly smaller than for a normal $2:1$ expansion, $^1/_2 : 1$ compression DTW algorithm.

Given the array of best scores, $\tilde{D}(b, e)$, of Eq. (7.10), the second level of the computation pieces together the individual reference pattern scores to minimize the overall accumulated distance over the entire test string. This can be accomplished using dynamic programming as follows. Consider ending frame e, as illustrated in Figure 7.6. We define

Ex.1022 / Page 247 of 333
AMAZON.COM, INC.

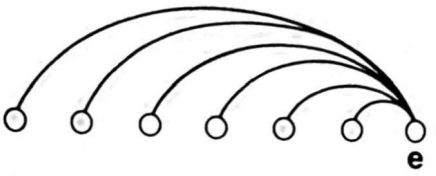


Figure 7.6 Series of paths ending at frame e.

the distance of the best path ending at frame e using a concatenated sequence of ℓ reference patterns as $\bar{D}_\ell(e)$, which can be derived as

$$\bar{D}_\ell(e) = \min_{1 \le b < e} [\tilde{D}(b, e) + \bar{D}_{\ell-1}(b-1)] \tag{7.12}$$

i.e., the best path ending at frame e using exactly ℓ reference patterns is the one with minimum distance over all possible beginning frames, b, of the concatenation of the best path ending at frame $b-1$ using exactly $\ell - 1$ reference patterns plus the distance of Eq. (7.10) of the best path from frame b to frame e (of $\mathcal{T}$). The recursion of Eq. (7.12) is identical to the dynamic programming recursion principle discussed in Chapter 4. In this case the "local" distance used in the recursion is the word distance $\tilde{D}(b, e)$ (with assumed beginning and ending frames) which is calculated using Eq. (7.10) based on the first level dynamic programming results. (For single-word pattern matching the local distance is simply the beginning to ending frame distance.)

Based on the recursion of Eq. (7.12) we can formulate the following DP algorithm for determining the overall best path:

Step 1, Initialization

$$\bar{D}_0(0) = 0, \qquad \bar{D}_\ell(0) = \infty, \qquad 1 \le \ell \le L_{\max}$$

Step 2, Loop on e for $\ell = 1$

$$\bar{D}_1(e) = \tilde{D}(1, e), \qquad 2 \le e \le M$$

Step 3, Recursion, Loop on e for $\ell = 2, 3, \ldots, L_{\max}$

$$\bar{D}_2(e) = \min_{1 \le b < e} [\tilde{D}(b, e) + \bar{D}_1(b-1)], \quad 3 \le e \le M$$
$$\bar{D}_3(e) = \min_{1 \le b < e} [\tilde{D}(b, e) + \bar{D}_2(b-1)], \quad 4 \le e \le M$$
$$\bar{D}_\ell(e) = \min_{1 \le b < e} [\tilde{D}(b, e) + \bar{D}_{\ell-1}(b-1)], \quad \ell + 1 \le e \le M$$

Step 4, Final Solution

$$D^* = \min_{1 \le \ell \le L_{\max}} [\bar{D}_\ell(m)]$$

and use $\tilde{N}(b, e)$ to backtrack to obtain actual sequence of reference patterns in $\mathcal{R}^s$.

To gain a better understanding of how the above recursion is carried out in practice,

Ex.1022 / Page 248 of 333
AMAZON.COM, INC.

the following exercise gives an explicit example of decoding a given matrix of distances, $\tilde{D}(b, e)$, into best strings.

Exercise 7.1

Assume we are given the 15×15 matrix of distances, $\tilde{D}(b, e)$, representing best accumulated distances between frames b and e, as shown below

beginning frame, b \ ending frame, e	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	–	–	9	10	13	17	22	25	29	33	37	41	45	50	60
2	–	–	–	7	10	14	17	21	25	29	32	36	40	45	53
3	–	–	–	–	9	13	16	19	23	26	28	32	35	40	47
4	–	–	–	–	–	6	8	9	11	12	14	19	23	28	33
5	–	–	–	–	–	–	4	6	8	10	12	15	19	23	30
6	–	–	–	–	–	–	–	9	12	16	19	23	27	30	33
7	–	–	–	–	–	–	–	–	15	18	22	27	32	37	45
8	–	–	–	–	–	–	–	–	–	12	16	21	25	29	33
9	–	–	–	–	–	–	–	–	–	–	6	8	10	13	16
10	–	–	–	–	–	–	–	–	–	–	–	5	7	9	12
11	–	–	–	–	–	–	–	–	–	–	–	–	4	6	8
12	–	–	–	–	–	–	–	–	–	–	–	–	–	2	4
13	–	–	–	–	–	–	–	–	–	–	–	–	–	–	2
14	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
15	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

1. Find the best path for a 1, 2, and 3 word match.
2. What are the total distance scores for the 3 string lengths? Which string length is most likely?

Solution 7.1

Although we can trivially determine the paths for the best one- and two-word matches, we must use the dynamic programming solution to determine the best three-word match. Hence it is worthwhile using the dynamic programming solution for all three matches. From the recursion above we get

$$\bar{D}_1(e) = \tilde{D}(1, e), \qquad 2 \le e \le 15$$

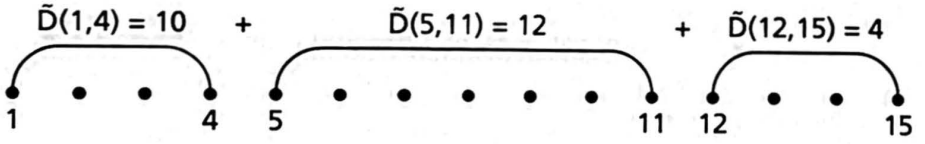
$$\bar{D}_2(e) = \min_{1 \le b < e} [\tilde{D}(b, e) + \bar{D}_1(b-1)], \qquad 3 \le e \le 15$$

$$\bar{D}_3(e) = \min_{1 \le b < e} [\tilde{D}(b, e) + \bar{D}_2(b-1)], \qquad 4 \le e \le 15$$

From the matrix of distances we get

e	$\bar{D}_1(e)$	$\bar{D}_2(e)$	$\bar{D}_3(e)$
3	9	-	-
4	10	-	-
5	13	-	-
6	17	$15 = (6 + 9)$	-
7	22	$14 = (4 + 10)$	-
8	25	$16 = (6 + 10)$	-
9	29	$18 = (8 + 10)$	$30 = (\bar{D}_2(6) + 15)$
10	33	$20 = (10 + 10)$	$26 = (\bar{D}_2(7) + 12)$
11	37	$22 = (12 + 10)$	$22 = (\bar{D}_2(8) + 6)$
12	41	$25 = (15 + 10)$	$23 = (\bar{D}_2(9) + 5)$
13	45	$29 = (19 + 10)$	$24 = (\bar{D}_2(10) + 4)$
14	50	$33 = (23 + 10)$	$24 = (\bar{D}_2(11) + 2)$
15	60	$40 = (30 + 10)$	$26 = (\bar{D}_2(12) + 4)$

1. For a one-word match we have $D_1^* = \bar{D}_1(15) = 60$ with path (1, 15). For a two-word match we have $D_2^* = \bar{D}_2(15) = 40$ with path (1, 4)(5, 15); i.e., word 1 spans frames 1 to 4 and word 2 spans frames 5 to 15. For a three-word match we have $D_3^* = \bar{D}_3(15) = 26$ with path (1, 4) (5, 11) (12, 15); i.e. word 1 spans frames 1 to 4, word 2 spans frames 5 to 11, and word 3 spans frames 12 to 15.
2. The distance scores for one-, two-, and three-word matches are 60, 40, and 26. Clearly, string length 3 has the smallest total distance and is most likely. The overall path of the 3 word string is



7.3.1 Computation of the Two-Level DP Algorithm

The total computation of the two-level DP algorithm is essentially the computation required to determine $\hat{D}(v, b, e)$. This computation is essentially the computation of $V \cdot M$ time warps corresponding to a single time warp for each reference pattern (V) and for each starting frame (M). The size of each time warp (which provides distance score for the range of ending frames associated with each starting frame) is approximately $\overline{N}(2R + 1)$ for an R frame range limited DTW search with a reference pattern of average length $\overline{N}$ frames. Hence the total computation of the two-level DP algorithm is

$$C_{2L} = V \cdot M \cdot \overline{N}(2R + 1) \text{ (grid points)} \tag{7.13}$$

in which the computation for each grid point constitutes a distance computation and the associated combinatorics. The storage of the two-level DP algorithm is essentially the

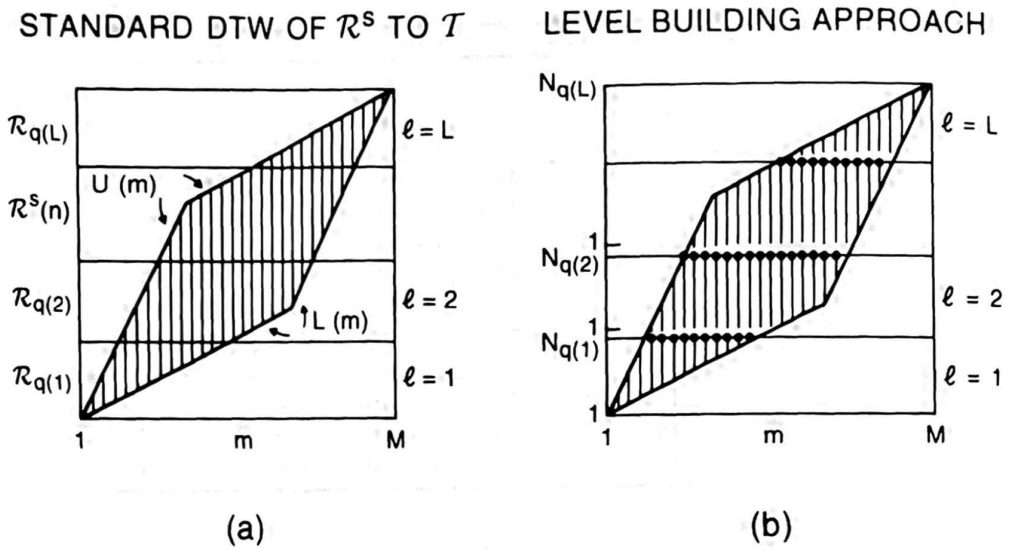


Figure 7.7 Illustration of standard DTW alignment of super-reference and test patterns (a), and level building alignment (b) (after Myers and Rabiner [4]).

storage of the reduced distance matrix $[\tilde{D}(b, e)]$, since only temporary storage of the full array $\hat{D}(v, b, e)$ is required. The required storage of the range reduced algorithm is

$$S_{2L} = 2M(2R + 1) \tag{7.14}$$

since for each of the M starting frames, distance scores and best word identifications are saved for an ending range of $(2R + 1)$ frames.

For a typical set of values—i.e., $M = 300$ frames, $\overline{N} = 40$ frames, $V = 10$ words, $R = 5$ frames—we require computation for 1,320,000 grid points and storage of 6600 locations for $\tilde{D}(b, e)$.

7.4 THE LEVEL BUILDING (LB) ALGORITHM

At the beginning of this chapter we discussed an exhaustive approach to connected word recognition in which we construct a set of super-reference patterns, $\mathcal{R}^s$, by concatenating every possible combination of word patterns (in sequence), and performing a global time alignment of each $\mathcal{R}^s$ to the test pattern $\mathcal{T}$. Figure 7.7a illustrates the computation associated with time aligning $\mathcal{R}^s$ to $\mathcal{T}$, namely iterative computation of accumulated distance at each grid point within a constrained region (the parallelogram of Figure 7.7a) of the alignment plane. Usually the computation is performed in a frame-synchronous manner; that is, the computation for a given test frame, m, is performed along a vertical stripe representing the allowable range of super-reference frames. Following this the computation for test frame $m + 1$ occurs, then for frame $m + 2$, etc., until the entire test frame sequence is used.

An alternative way of aligning $\mathcal{R}^s$ and $\mathcal{T}$ is shown in part b of Figure 7.7. Instead of performing the computation strictly along vertical stripes, the computation, for a given test frame m, is truncated at a fixed horizontal level (which corresponds to the super reference

frame at the end of the first word in $\mathcal{R}^s$). (The use of the word "level" here should not be confused with its usage in the two-level DP algorithm of the previous section. In the two-level DP algorithm the first level is the calculation of local warping distances for tentative word candidates, and the second level is the search for the optimal concatenated word path.) This procedure is iterated for all test frames for which we can define a range of reference frames within the allowed alignment region. For each test frame whose vertical stripe intersects the horizontal level corresponding to the end of the first word, we store the accumulated distance so that the DTW computation can be picked up at the next level (word in the string) by beginning the search at an appropriate test frame and iterating the search along vertical stripes corresponding to regions of intersection with the second word in the super-reference pattern. This procedure is iterated through all levels (words in $\mathcal{R}^s$) until the complete alignment grid has been covered.

The differences between the sequence of computation of the standard DTW alignment along complete vertical stripes (as in Figure 7.7a) and the level building alignment along partial vertical stripes (as in Figure 7.7b) are essentially minor for a single super-reference pattern. The efficiency of the approach comes with the realization that the computation at each level can be performed on all V reference patterns *before* proceeding to the next level. The advantage of doing this is that after doing the V warps at a given level, we can compare accumulated distance scores *across vocabulary words*, and retain only the minimum (best) score at each path ending frame, and still pick up the computation at the next level with no loss of optimality. The key difference between the level building algorithm and the two-level DP algorithm is that (with no loss of information) partial word decisions are made during the DP search, and these decisions are used to reduce the search range in later stages of the LB algorithm, whereas the two-level DP algorithm accumulates all the word scores until the end of the entire utterance and then decides the word string in a separate and independent second level DP calculation. The advantage we accrue is that, at each level, we do exactly V time warps, so that the total computation is on the order of $V \cdot L$ time warps for an L-word string. Since L is always significantly smaller than the number of test frames, M, we see that the level building approach inherently involves significantly less computation than the two-level DP algorithm (which required $V \cdot M$ time warps).

With this simplified overview of the level building approach, we now formally describe the implementation of the level building algorithm.

7.4.1 Mathematics of the Level Building Algorithm

We define $\bar{D}_\ell^v(m)$ as the minimum accumulated distance, at level ℓ, using reference pattern $\mathcal{R}_v$, to frame m of the test pattern, $\mathcal{T}$. Clearly $\bar{D}_\ell^v(m)$ is defined for $1 \le \ell \le L_{\max}$, $1 \le v \le V$ and $1 \le m \le M$. Consider the implementation of level 1 as shown in Figure 7.8.

The first reference pattern, $\mathcal{R}_1$, is aligned with $\mathcal{T}$ beginning at frame 1 of $\mathcal{T}$ using a standard DTW alignment procedure. The warping paths intersect the last frame in $\mathcal{R}_1$ (frame N_1) at a range of test frames, namely $m_{11}(1) \le m \le m_{12}(1)$. (For example if we used a DTW procedure with a minimum and maximum expansion of $\frac{1}{2}$ and 2 we would have $N_1/2 \le m \le 2N_1$.) For each of these warp path ending frames we store

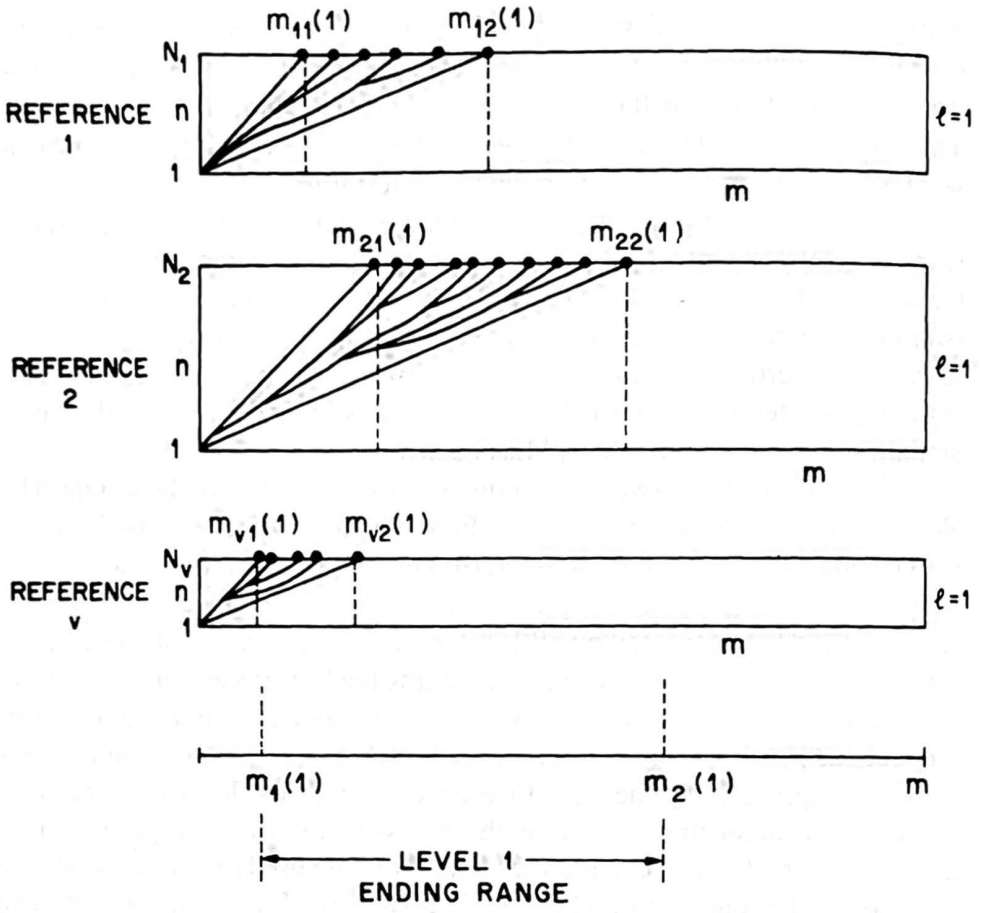


Figure 7.8 Implementation of level 1 of the level building algorithm (after Myers and Rabiner [4]).

the accumulated distances, $\bar{D}_1^1(m)$. Similarly, for the second reference pattern, $\mathcal{R}_2$, with duration N_2 frames, we begin at frame 1 of $\mathcal{T}$ and obtain the best warping paths to the range $m_{21}(1) \le m \le m_{22}(1)$. (Clearly $m_{21}(1) \ne m_{11}(1)$ and $m_{22}(1) \ne m_{12}(1)$, since N_2 and N_1 are generally different. If $N_2 = N_1$, then $m_{21}(1) = m_{11}(1)$ and $m_{22}(1) = m_{12}(1)$.) This procedure is iterated for all V reference patterns at level 1. Thus the output of level 1 is the array of accumulated distances and the corresponding ranges, i.e.,

$$\begin{aligned} &\bar{D}_1^1(m), \quad m_{11}(1) \le m \le m_{12}(1) \\ &\bar{D}_1^2(m), \quad m_{21}(1) \le m \le m_{22}(1) \\ &\vdots \\ &\bar{D}_1^V(m), \quad m_{V1}(1) \le m \le m_{V2}(1). \end{aligned} \tag{7.15}$$

We can now define the ending range of level 1, $m_1(1) \le m \le m_2(1)$, as the composite range of m over which $\bar{D}_1^v$ is defined (and is not infinity), namely,

$$m_1(1) = \min_{1 \le v \le V} [m_{v1}(1)] \tag{7.16}$$

$$m_2(1) = \max_{1 \le v \le V} [m_{v2}(1)] \tag{7.17}$$

and for each frame, m, in the composite ending range, $m_1(\ell) \le m \le m_2(\ell)$, we need to store

$$\bar{D}_\ell^B(m) = \min_{1 \le v \le V} [\bar{D}_\ell^v(m)] \quad \text{– best distance at level } \ell \text{ to frame } m \tag{7.18}$$

$$\bar{N}_\ell^B(m) = \arg \min_{1 \le v \le V} [\bar{D}_\ell^v(m)] \quad \text{– reference pattern index which gave distance at level } \ell \text{ to frame } m \tag{7.19}$$

$$\bar{F}_\ell^B(m) = \bar{F}_\ell^{\bar{N}_\ell^B(m)}(m) \quad \text{– backpointer to best ending frame at previous level that achieves } \bar{D}_\ell^B(m) \tag{7.20}$$

(By definition $\bar{F}_1^B(m) = 0$ for all m since the ending frame of the 0^{th} level is 0.) By storing only $\bar{D}_\ell^B(m)$, $\bar{N}_\ell^B(m)$ and $\bar{F}_\ell^B(m)$, we significantly reduce the storage at each level and yet we retain all the information required to pick up the optimal path through the entire grid as shown below.

The second-level computation does not begin until the first-level computation is finished. To see how the computation is picked up at level 2, consider Figure 7.9, which shows a series of time warps that span a set of beginning frames and provide a new set of ending frames.

For each reference pattern, $\mathcal{R}_v$, the range of starting frames is $m_1(1) \le m \le m_2(1)$ because every ending point of the first level is a possible starting point in the second level. Hence for each frame m in the beginning range, and for each frame at the beginning of the reference pattern, we must consider paths coming from both the current reference pattern and the previous level. Other than the broadened beginning region, each DTW at level 2 is essentially identical to those at level 1. Thus for reference $\mathcal{R}_1$ the range of ending frames at level 2 is $m_{11}(2) \le m \le m_{12}(2)$; for reference $\mathcal{R}_2$ the range of ending frames at level 2 is $m_{21}(2) \le m \le m_{22}(2)$, etc. We again derive the ending range at the end of level 2 as

$$m_1(2) = \min_{1 \le v \le V} [m_{v1}(2)] \tag{7.21}$$

$$m_2(2) = \max_{1 \le v \le V} [m_{v2}(2)] \tag{7.22}$$

and for each frame in $m_1(2) \le m \le m_2(2)$ we determine the best distance $\bar{D}_2^B(m)$, the reference with the best distance, $\bar{N}_2^B(m)$, and the backpointer $\bar{F}_2^B(m)$.

We can continue the level building procedure through all levels until level $L_{\max}$ in the above manner and we obtain, as the final solution, D^* as

$$D^* = \min_{1 \le \ell \le L_{\max}} [\bar{D}_\ell^B(m)]. \tag{7.23}$$

It should be clear that by performing the computation in levels (i.e., a word at a time) and by doing appropriate minimization within levels, we avoid much of the computation of the two-level DP algorithm described in the previous section. However, the negative feature of the level building algorithm is that the computation is level synchronous, not time synchronous; that is, we can go back to a given test frame at many levels. Hence, real-time hardware implementations of level building are difficult, if not impossible, to

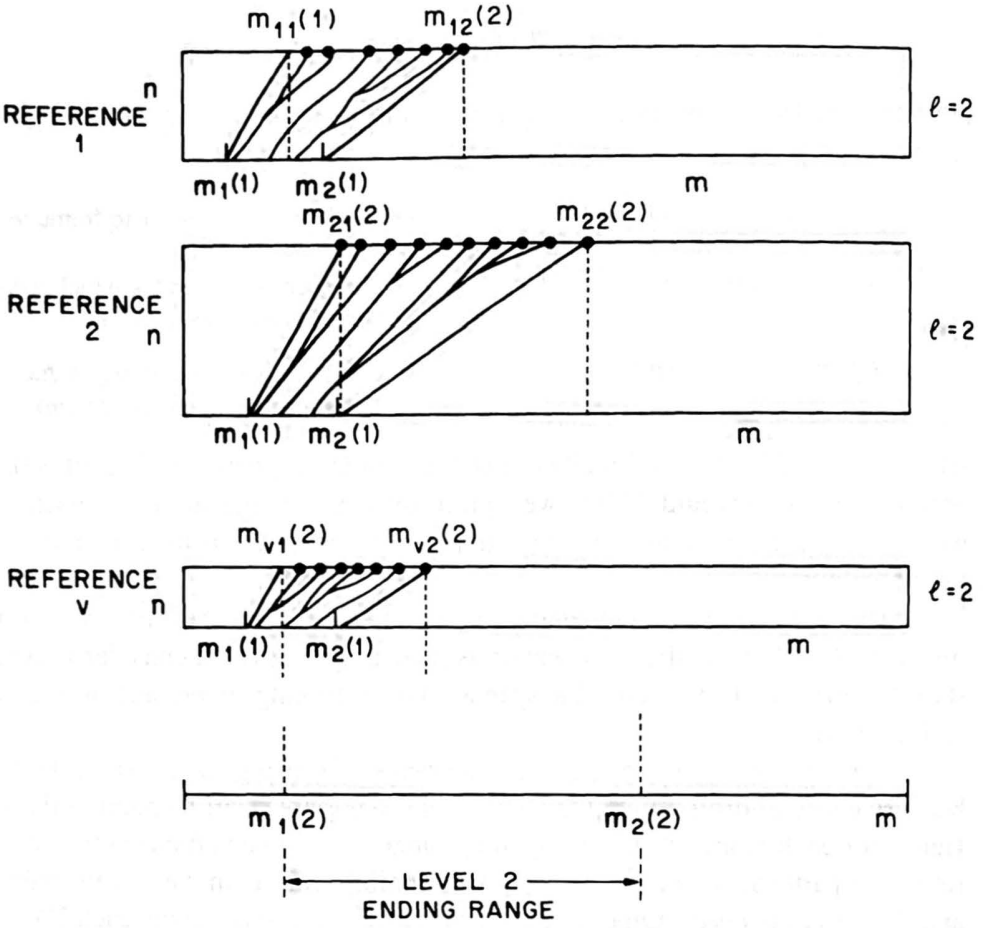


Figure 7.9 Implementation of level 2 of the level building algorithm (after Myers and Rabiner [4]).

implement. (We will return to these issues later in this section.)

To gain a better understanding of the basic concepts of the level building algorithm, consider the simple example shown in Figure 7.10. Here we assume that the vocabulary consists of two words (which we call A and B for simplicity) and that the two reference patterns, $\mathcal{R}_A$ and $\mathcal{R}_B$, are of equal length. We also assume we are only interested in an $\ell = 4$ level solution. (This greatly simplifies the range of the dynamic programming search parallelogram which, as seen in Figure 7.10, is essentially identical to the case for isolated word template matching. In the next section, where multiple level considerations are discussed, the dynamic programming search range will be shown to be significantly more complicated to specify.) Since both patterns are of equal length, the ending regions for both words, at each level, are identical. Hence at each level ending region we choose the reference pattern (A or B) that gave the smallest accumulated distance to that frame. In this simple example, there are 6 ending frames at level 1, with the best path corresponding to $\mathcal{R}_A$ for the first two frames, and $\mathcal{R}_B$ for the next 4 frames. At level 2 there are 10 ending frames; at level 3 there are 6 ending frames, and finally at level 4 there is only one ending frame corresponding to frame M, the end of the test utterance. By tracing back the (best)

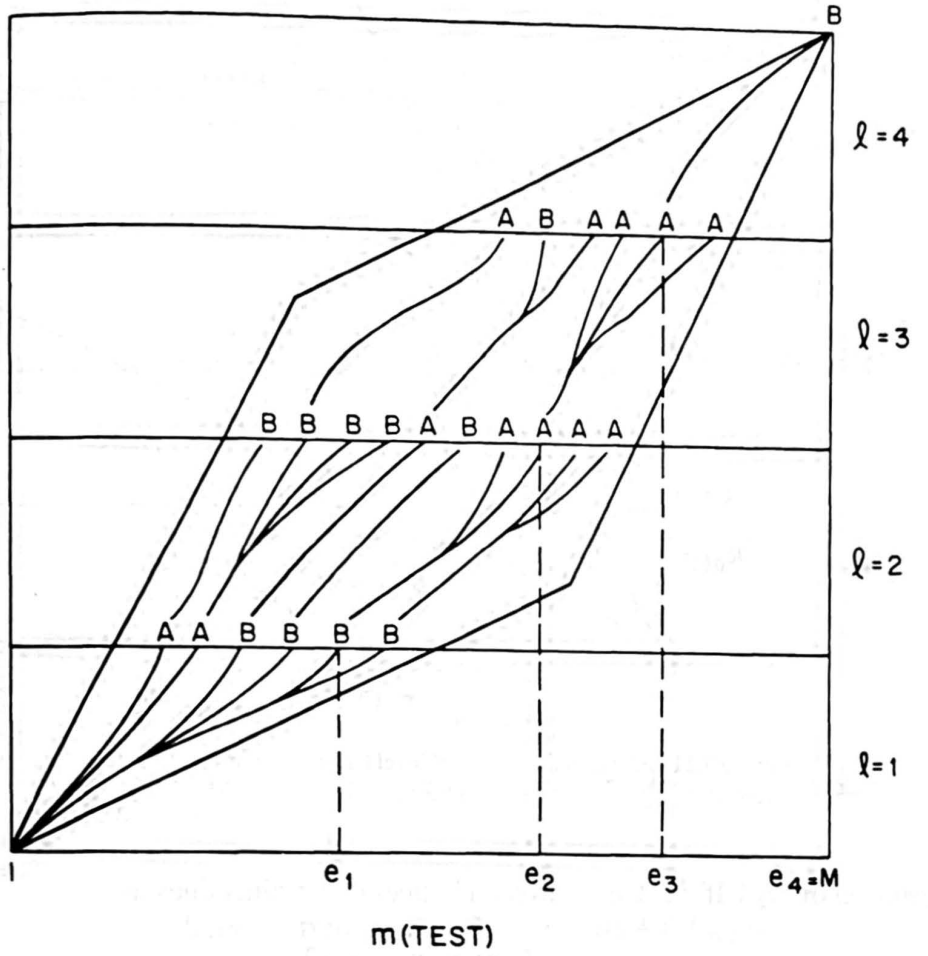


Figure 7.10 Simple example illustrating level building on two reference patterns of equal length (after Myers and Rabiner [4]).

path ending at $m = M$ we see that the sequence of reference patterns giving the best score is

$$\mathcal{R}^* = \mathcal{R}_B \oplus \mathcal{R}_A \oplus \mathcal{R}_A \oplus \mathcal{R}_B$$

with test frames e_1, e_2, e_3, and $e_4 = M$ corresponding to the last frames of the 4 words in the sequence.

7.4.2 Multiple Level Considerations

In the more realistic case in which we want the level building solutions for all feasible levels (i.e., where there is a possible solution), there are some very simple techniques that can be used to eliminate unnecessary computation. To understand this issue consider the "standard" warping range of the level building algorithm as shown in Figure 7.11. (We assume here that we use a DTW algorithm with a maximum expansion of 2 and a minimum

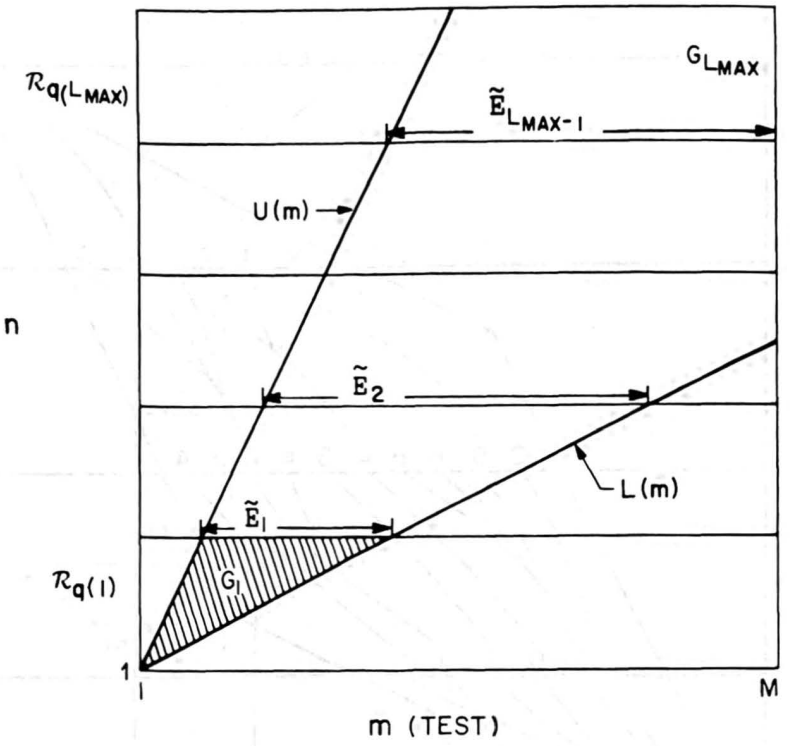


Figure 7.11 Computation region of level building algorithm for a fixed-length reference pattern (after Myers and Rabiner [4]).

expansion of $^1/_2$.) If we define lower and upper constraints lines as

$$L(m) = (m+1)/2 \tag{7.24}$$

$$U(m) = 2(m-1)+1 \tag{7.25}$$

then for a fixed-length reference pattern we get the computation and ending regions shown in Figure 7.11. (We denote the computation region at level ℓ as G_ℓ with ending region $\tilde{E}_\ell$.) It should be clear that some of the computation of Figure 7.11 is unnecessary, since there do not exist paths from some of the computed points to ends of reference patterns at any level.

We can use the constraint that in order to do the computation at any grid point, the path from that grid point must be capable of reaching the end of the reference pattern before the end of the test pattern as shown in Figure 7.12. Here we have drawn lines, at each level, of slope 2, from the last frame of the test pattern and the last frame of the reference pattern, and used them as constraints on the grid. We have also drawn a line, at level L_{max}, of slope $^1/_2$ from the last test frame and the last reference frame to further constrain the grid at the last level. The lower and upper constraints of the simplified grid can be described by the equations

$$L(m) = \max\left[\frac{m+1}{2}, 2(m-M)+\theta(\ell)\right] \tag{7.26}$$

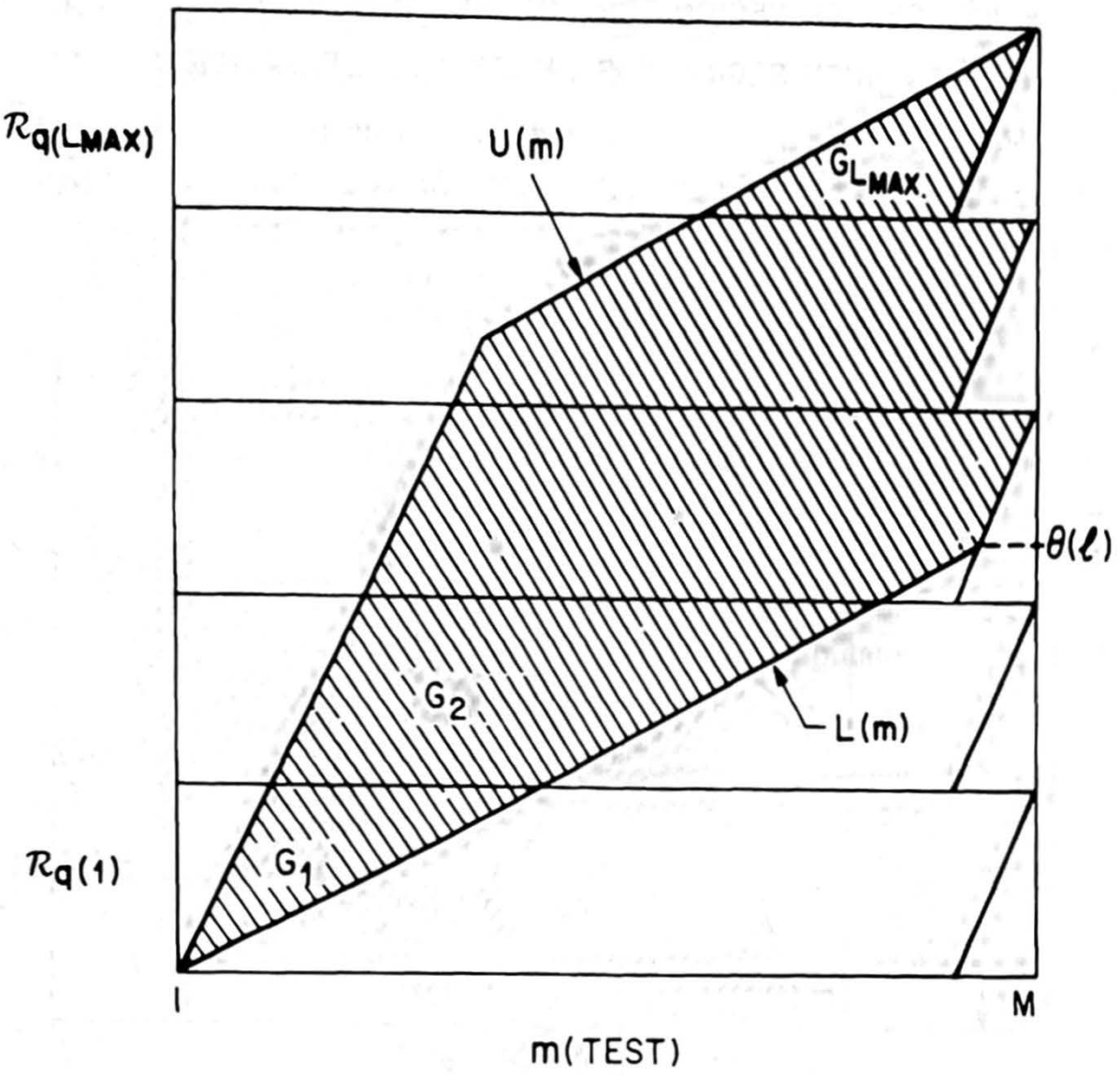


Figure 7.12 Reduced computation region using upper- and lower-level constraints (after Myers and Rabiner [4]).

$$U(m) = \min\left[2(m-1), \frac{1}{2}(m-M) + \theta(L_{\max})\right] \tag{7.27}$$

in which the reference pattern length function, $\theta(\ell)$, is the accumulated number of frames of the reference patterns used up to level ℓ where $(m+1)/2 = 2(m-M) + \theta(\ell)$ at some frame $m < M$. The resulting region of computation is somewhat reduced from that shown in Figure 7.11.

Since the length of each reference pattern is different (in general) the actual computation regions for a level building search based on variable length reference patterns is shown in Figure 7.13. Here we show the computation regions, at each level, for the shortest and for the longest reference patterns. Fundamentally, the pattern of computation is similar to that of Figure 7.11.

7.4.3 Computation of the Level Building Algorithm

From the above discussion it should be clear that the basic computation of the level building algorithm is a series of V time warps at each level, where V is the size of the vocabulary. If we assume the maximum number of levels in $L_{\max}$, then we need $V\,L_{\max}$ time warps. An overbound on size of each time warp is $\overline{N}M/3$ grid points where $\overline{N}$ is the average length of each reference pattern and M is the number of frames in the test pattern. Hence the total

Ex.1022 / Page 258 of 333
AMAZON.COM, INC.

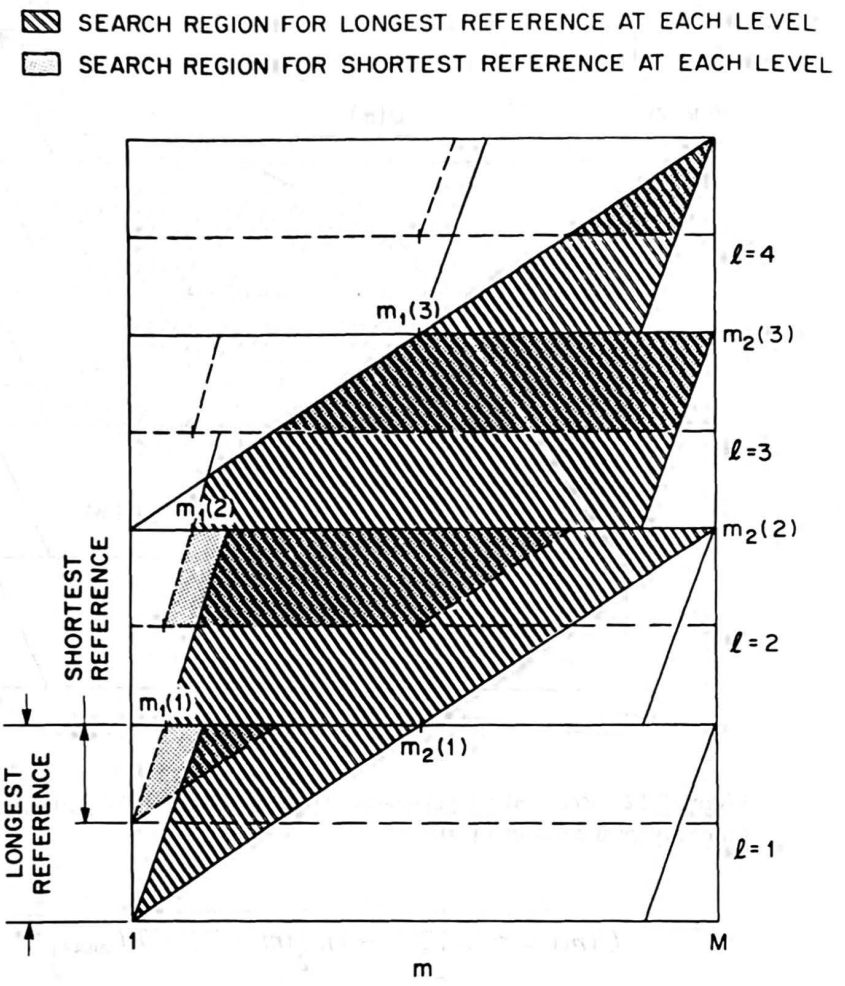


Figure 7.13 Overall computation pattern of level building algorithm for variable length reference patterns (after Myers and Rabiner [4]).

computation of the level building algorithm is

$$C_{LB} = V \cdot L_{\max} \cdot \overline{N} \cdot M/3 \qquad \text{grid points} \tag{7.28}$$

with storage

$$S_{LB} = 3M \cdot L_{\max} \tag{7.29}$$

since we need storage for $\bar{D}^B$, $\bar{N}^B$, and $\bar{F}^B$ at each value of m and for each level ℓ.

Using the same values for M, $\overline{N}$, and V as used in determining the computation and storage of the two-level DP method, namely $M = 300$, $\overline{N} = 40$, $V = 10$, and using $L_{\max} = 7$, we get $C_{LB} = 280,000$ and $S_{LB} = 6300$. The basic computation of the level building algorithm is a factor of 4.7 less than that of the two-level DP method; the storage of both methods is about the same.

The above calculations are based on full DTWs at each level for each reference pattern. Because of the overlaps of regions in the (m, n) plane, at different levels, some of the assumed computation is redundant. The following exercise illustrates this point.

Exercise 7.2

In implementing different levels of the level building algorithm, we have assumed all levels are independent. Thus, at each level, we have performed a full DTW, for each reference pattern, where the size of the DTW was $\overline{N} \cdot M/3$ (grid points). An alternative, and, we hope, more efficient, approach is to realize that for each reference pattern, a significant portion of the computation at each level (namely that of distance computation) may have previously been performed at an earlier level, and therefore only the combinatorics part of the DTW is truly independent at each level.

1. Show that for the assumed parameters of the system (i.e., M = number of test frames = 300, $\overline{N}$ = average number of frames per reference pattern = 40, V = size of vocabulary = 10, $L_{\max}$ = maximum number of levels = 7), the alternative implementation of storing all previously computed distances is more efficient than the standard implementation. (Assume that the cost of the combinatorics in a DTW procedure is negligible.)
2. What is the ratio of computation of the standard implementation to the stored distance implementation?
3. If we assume the cost of the combinatorics at each grid point is $^1/_5$ the cost of a distance computation, how do the results to parts 1 and 2 change?

Solution 7.2

1. The simplest way of exploiting previously computed distances is to precompute the entire grid of distances from each reference pattern frame to each test frame. (Clearly this is *not* the most efficient implementation, since many of these full grid distances will never be required in practice.) If we do this we need

$$C_{\text{DIST}} = V \cdot M \cdot \overline{N} \text{ (distances)} = 120{,}000.$$

The number of combinatorics is just the number of grid points used in the level building method; hence

$$C_{\text{COMB}} = V \cdot L_{\max} \cdot \overline{N} \cdot \frac{M}{3} \text{ (grid points)} = 280{,}000.$$

The total computation of this approach is then

$$C_{\text{TOTAL}} = C_{\text{DIST}} + \alpha \cdot C_{\text{COMB}}$$

in which α is the weight for combinatorics. If we assume combinatorics are negligible, $\alpha = 0$, we get

$$C_{\text{TOTAL}} = C_{\text{DIST}} = 120{,}000$$

which is significantly less than the 280,000 distances for the standard implementation (i.e., one per grid point).

2. The ratio of computation of the two approaches is

$$\frac{C_{\text{LB}}}{C_{\text{TOTAL}}} = \frac{V \cdot L_{\max} \cdot \overline{N} \cdot M/3}{V \cdot M \cdot \overline{N}} = \frac{L_{\max}}{3} = 2.33$$

and is independent of V, M, and $\overline{N}$ but only depends on $L_{\max}$.

3. If we assume the cost of combinatorics is $^1/_5$ the cost of a distance computation we get

$$C_{\text{TOTAL}} = V \cdot M \cdot \bar{N} + V \cdot L_{\max} \cdot \bar{N} \cdot \frac{M}{3} \frac{1}{5} = 176{,}000$$

with the ratio in computation being

$$\frac{C_{\text{LB}}}{C_{\text{TOTAL}}} = \frac{V \cdot L_{\max} \cdot \bar{N} \cdot M/3}{V \cdot M \cdot \bar{N} + V \cdot L_{\max} \cdot \bar{N} \cdot \frac{M}{3}\frac{1}{5}} = \frac{L_{\max}}{3(1 + \frac{L_{\max}}{15})} = 1.59$$

7.4.4 Implementation Aspects of Level Building

Even though we have already shown that the computation for the level building approach to connected word recognition is significantly less than that of the two-level DP approach of Section 7.2, there are several ways of even further reducing the computational load of the algorithm. These include the following:

1. Beginning range reduction in which we reduce the size of the initial region (range of m) for which valid paths to a given level can begin.
2. Global range reduction in which we reduce the size of the region (width of the template) that is tracked, within a level, to determine a best path to each possible level ending frame, m.
3. Test pattern ending range in which we *increase* the range over which the global path can match the connected word string; this procedure provides some robustness to ending frame errors.
4. Reference pattern uncertainty ranges in which we increase the range of search at the beginning and end of reference patterns to allow for some degree of word coarticulation at reference word boundaries.

The way in which we implement these features is as follows.

1. **Beginning range reduction—M_T**
   For the complete level building algorithm, at level $\ell - 1$, we retain the best score, $\bar{D}^B_{\ell-1}(m)$, for each frame m in the ending region $m_1(\ell - 1) \leq m \leq m_2(\ell - 1)$. It should be obvious that, in general, the best global path is not at either boundary but somewhere in the middle range of m. Hence if we could eliminate some of the ending range at level $\ell - 1$, the search at level ℓ would involve less computation. (To see this, consider the limiting case where we make the ending region, at each level, a single frame; then the computation at each new level is a simple DTW with a single initial frame, much as occurs at level 1.)
   To determine how to reduce the ending range at level $\ell - 1$ we need to normalize the best accumulated distance scores by the number of frames. Thus we first find the locally best (minimum) normalized accumulated distance as:

$$\phi_{\ell-1} = \min_{m_1(\ell-1) \leq m \leq m_2(\ell-1)} \left[\frac{\bar{D}^B_{\ell-1}(m)}{m} \right]. \tag{7.30}$$

We now define a reduced-range level threshold as $M_T \cdot \phi_{\ell-1}$ where M_T is a defined parameter, and search the range $m_1(\ell-1) \le m \le m_2(\ell-1)$ to find the indices S_ℓ^1 and S_ℓ^2 such that

$$S_\ell^1 = \arg \max_{m_1(\ell-1)\le m\le m_2(\ell-1)} \left[\frac{\bar{D}_{\ell-1}^B(m)}{m} > M_T \cdot \phi_{\ell-1} \quad \forall\, m \le S_\ell^1 \right] \tag{7.31}$$

$$S_\ell^2 = \arg \max_{m_1(\ell-1)\le m\le m_2(\ell-1)} \left[\frac{\bar{D}_{\ell-1}^B(m)}{m} > M_T \cdot \phi_{\ell-1} \quad \forall\, m \ge S_\ell^2 \right]. \tag{7.32}$$

To see what is achieved by the beginning range reduction, consider Figure 7.14, which shows a sequence of levels and the resulting ending ranges with and without range reduction (top graph), and the way in which the reduced range is determined (bottom graph). For the level shown, the best normalized distance score, $\phi_{\ell-1}$, is determined as the smallest value of the curve, and the threshold $M_T \cdot \phi_{\ell-1}$ is shown as a level above $\phi_{\ell-1}$ (clearly $M_T \ge 1.0$). The initial reduced range point, S_ℓ^1, is the last index (beginning from $m = m_1(\ell-1)$) where the curve is always above the threshold; similarly the index S_ℓ^2 is the last index (beginning from $m = m_2(\ell-1)$) where the curve is always above the threshold. The reduced range means that the computation at the next level is smaller as seen at the top of Figure 7.14. Depending on the value of M_T, the reduced range can get smaller (as $M_T \to 1$) or larger (as $M_T \to \infty$). It should be clear that too small a value of M_T will allow the best path to be prematurely eliminated; hence proper choice of M_T is essential.

2. **Global range reduction—ϵ**
   The idea behind global range reduction is to reduce the search range along the reference axis, for each test frame, by tracking the global minimum, and allowing only a range around the global minimum. Thus for each test frame, m, at each level ℓ, and for each reference pattern, $\mathcal{R}_v$, we determine the local minimum, $c(m)$, as

$$c(m) = \arg \min_{c(m-1)-\epsilon\le n\le c(m-1)+\epsilon} [D_\ell^v(m-1, n)] \tag{7.33}$$

   in which $D_\ell^v(m-1, n)$ is defined to be the best distance at level ℓ using reference $\mathcal{R}_v$ at test frame $m-1$ to reference frame n (as determined by the local alignment path) and with $c(1)$ defined to be 1. Figure 7.15 illustrates the global range reduction for a typical level building search. For global range reduction to be effective we must have the reduced range, $2\epsilon + 1$, be smaller than the typical reference pattern width.

3. **Test pattern ending range—δ_{END}**
   The idea here is to allow a range of test pattern ending frames, rather than restricting the ending frame to $m = M$. This feature provides a measure of robustness to test pattern endpoint errors. If we extend the end of the test pattern by δ_{END} frames, the global level building solution is modified to be

$$D^* = \min_{1\le\ell\le L_{\max}} \min_{M-\delta_{\text{END}}\le m\le M} [\bar{D}_\ell^B(m)]. \tag{7.34}$$

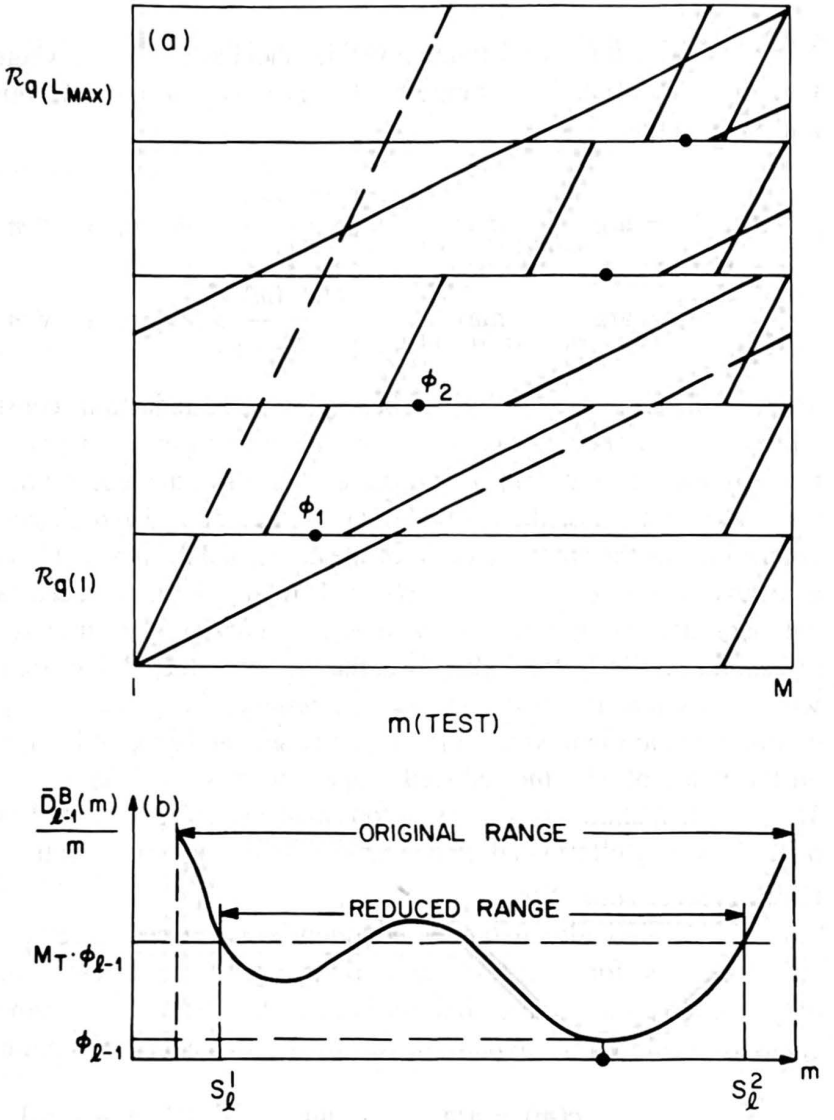


Figure 7.14 Illustration of beginning range reduction (after Myers and Rabiner [4]).

4. **Reference pattern uncertainty regions—δ_{R_1}, δ_{R_2}**
   To account for coarticulation of words across word boundaries, the level building algorithm allows a range of beginning and ending frames of the reference pattern. In this manner, at any level, the path can begin over the range $1 \leq n \leq 1+\delta_{R_1}$ (where n is the reference pattern index), and end at any frame in the range $N_v - \delta_{R_2} \leq n \leq N_v$. For appropriate values of δ_{R_1} and δ_{R_2}, it is possible to have a path which skips $(\delta_{R_1} + \delta_{R_2})$ frames at highly coarticulated word boundaries (e.g., the boundary between six and seven in the string "six-seven"). Figure 7.16 illustrates the use of reference pattern uncertainty regions in level building.

A summary of the four implementation aspects of level building, as discussed in this section, is shown in Figure 7.17 in which all four features are combined in a single

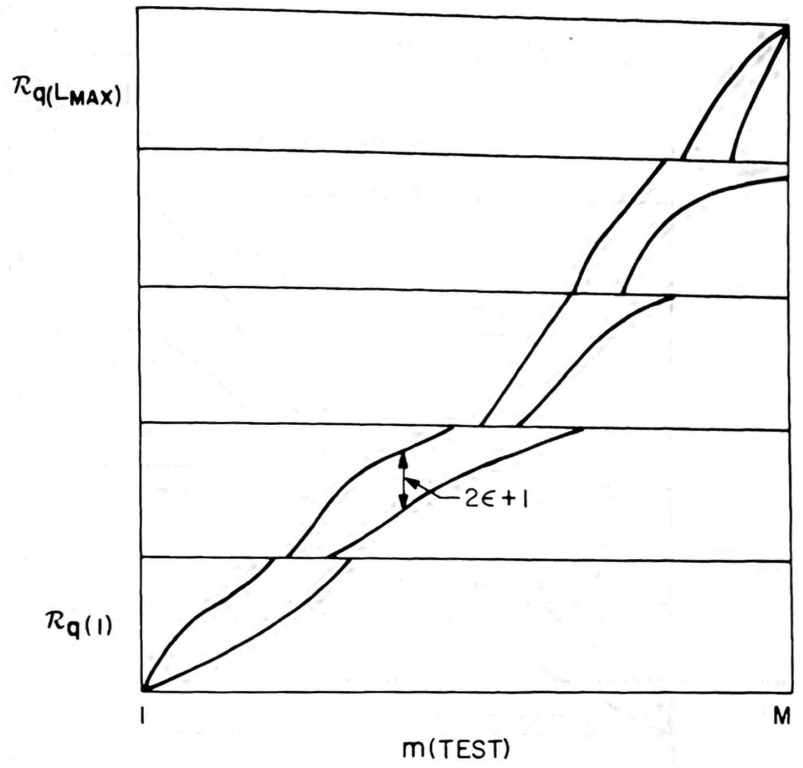


Figure 7.15 Illustration of global range reduction (after Myers and Rabiner [4]).

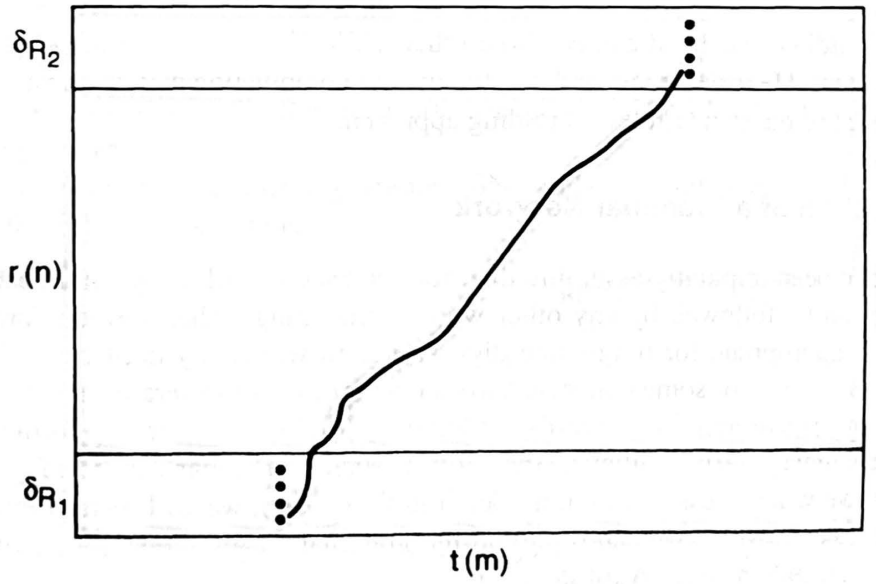


Figure 7.16 Illustration of the use of reference pattern uncertainty regions (after Myers and Rabiner [4]).

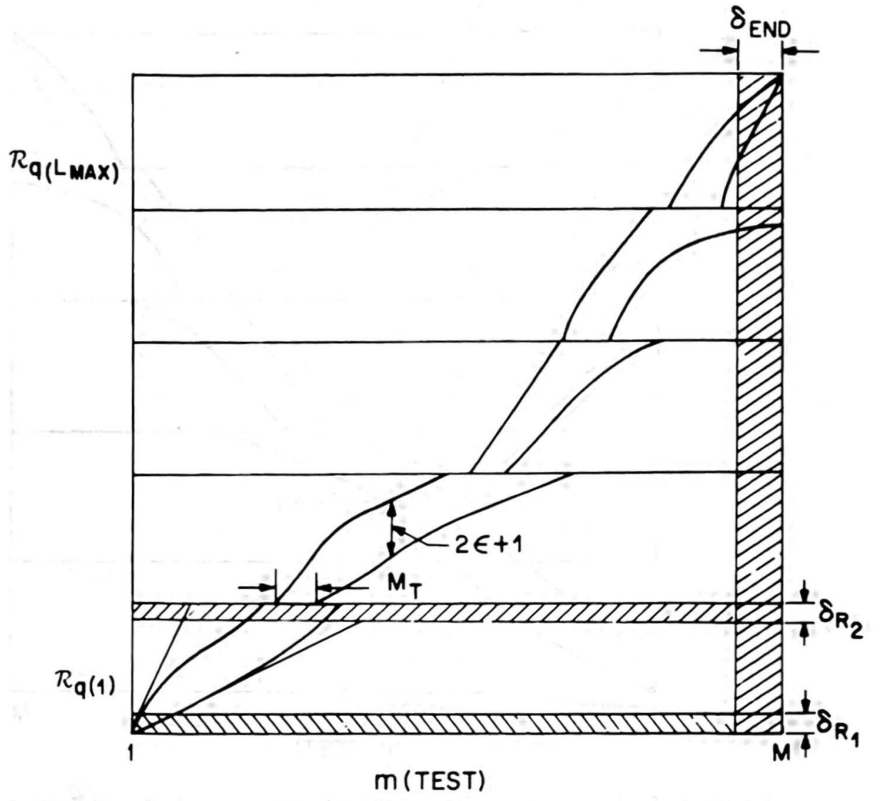


Figure 7.17 Summary of level building implementation of computation reduction methods (after Myers and Rabiner [4]).

level building search. It can be shown that with judicious choice of the implementation parameters, M_T, ϵ, δ_{END}, δ_{R_1} and δ_{R_2}, the overall computation can be substantially reduced from that of the standard level building approach.

7.4.5 Integration of a Grammar Network

We have been implicitly assuming that, for connected word recognition, each word in the string can be followed by any other word in the string. This implicit form of grammar is most appropriate for things like digit strings in which any digit can follow any other digit. However, for some connected word recognition tasks there is an explicit set of rules (grammar) governing which words can logically follow other words to form valid sentences in the language [10]. Although the form of such a grammar can be of several different types (we will discuss this in more detail in Chapter 8), we will restrict ourselves here to those tasks in which we can represent the grammar by a finite state network (FSN) or a finite state automata (FSA) of the form

$$G = A(Q, V, \delta, q_0, Z) \tag{7.35}$$

where

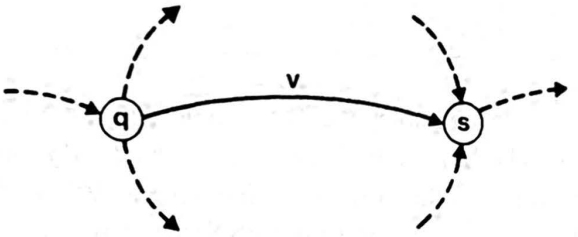
$$Q = \text{set of states}$$

$$V = \text{set of vocabulary words}$$
$$\delta = \text{set of transitions}$$
$$q_0 \in Q = \text{initial state}$$
$$Z \subseteq Q = \text{set of terminal states}$$

and the set of transitions obeys the rule

$$\delta(q, v) = s \tag{7.36}$$

meaning that word v drives the state from q to s



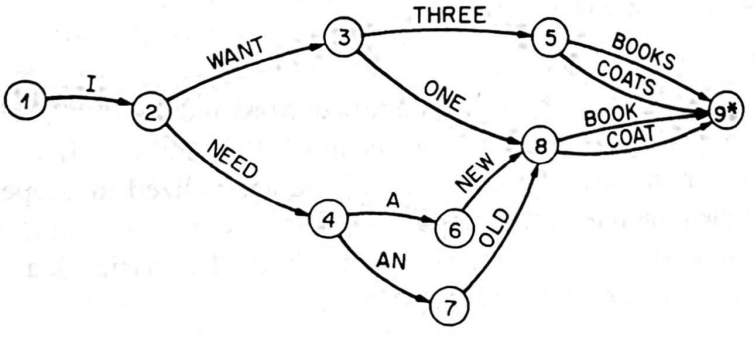
To integrate the FSN grammar network into the following level building algorithm we must do the following:

1. Identify levels with states rather than word position so that word candidates at the ℓ^{th} level need not be temporally contiguous to those at the $(\ell + 1)^{st}$ level
2. Partition the vocabulary so that only the reference patterns for words leaving the ℓ^{th} state are matched at the ℓ^{th} level
3. Retain state backtracking pointers for recovering the best matching string.

(It should be noted that for the most efficient computation, the states in Q should be topologically sorted.)

To illustrate how a simple grammar FSN can be integrated into the LB algorithm, consider the following example:

1.	I	5.	ONE	9.	BOOKS	13.	OLD
2.	WANT	6.	A	10.	COAT		
3.	NEED	7.	AN	11.	COATS		
4.	THREE	8.	BOOK	12.	NEW		



Current State	Words Used	Predecessor State	Currect Level	Predecessor Levels
2	I	1	1	0
3	WANT	2	2	1
4	NEED	2	3	1
5	THREE	3	4	2
6	A	4	5	3
7	AN	4	6	3
8	ONE	3	7	2
8	NEW	6	8	5
8	OLD	7	9	6
9*	BOOK, COAT	8	10	7,8,9
9*	BOOKS, COATS	5	11	4

If we study this simple example, we see that levels 2 and 3 both pick up from level 1; similarly both levels 4 and 7 pick up from level 2. By building up the computation by levels (keeping track of the correct predecessor level), and by backtracking the final result by states (according to the grammar FSN) we can essentially use all the techniques described to efficiently find the best grammatically correct string.

7.4.6 Examples of LB Computation of Digit Strings

Figures 7.18 and 7.19 show two examples of connected digit strings matched using the LB algorithm. Figure 7.18 is for the string "51560" and shows the computation building up level by level. For this example, the locally best path at each level (shown by the digit to the right of the last test frame) is actually the globally best digit. At the end of level four the algorithm provided the four best choices with the string "5157" having the lowest score of 0.553. At the end of level 5 there were six string choices with the correct string "51560" having the lowest average accumulated distance of 0.333.

The example in Figure 7.19 is for the string "99211," which did not provide a match nearly as good as the previous example. We see here that the locally best string at each level, namely "90111" is not globally best, and actually is incorrect in two positions. Although the algorithm gets the correct string as the best score, the second best string is the string "901," which has two digit deletions, and one digit-substitution error.

7.5 THE ONE-PASS (ONE-STATE) ALGORITHM

The third general approach to the connected word recognition problem is a procedure which was originally proposed by Vintsyuk in 1971 [5] and which has been "rediscovered" several times in the last two decades [6–8] and generalized in scope [9]. The algorithm has been called the one-pass procedure or the one-state algorithm, or most recently, the frame-synchronous level building (FSLB) method. The basic idea behind the algorithm is illustrated in Figure 7.20, which shows a grid with the test pattern, $\mathcal{T}$, mapped to

Ex.1022 / Page 267 of 333
AMAZON.COM, INC.

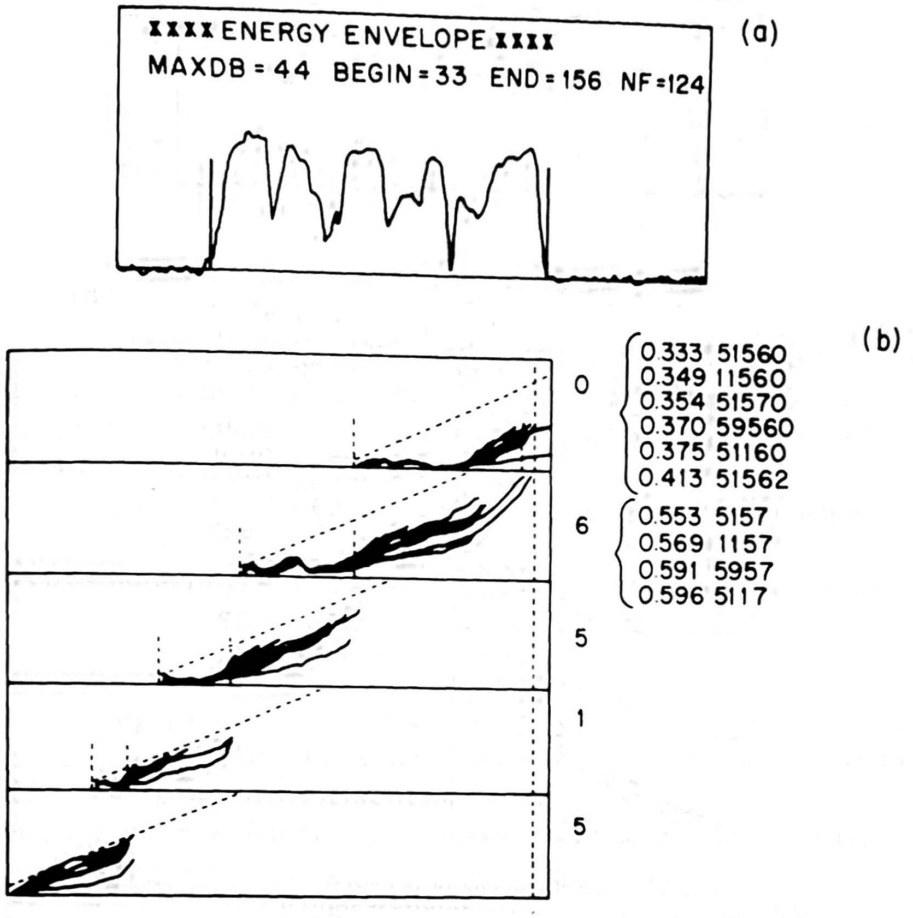


Figure 7.18 Level building of the string "51560" (after Myers and Rabiner [4]).

the horizontal axis, and the set of reference patterns, $\{\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_V\}$ mapped to the vertical axis.

Using the standard notation of m to represent the test frame index, $1 \leq m \leq M$, v to represent the reference pattern ($\mathcal{R}_v$) index, $1 \leq v \leq V$, and n to represent the reference frame index of pattern $\mathcal{R}_v$, $1 \leq n \leq N_v$, then for each test frame we calculate the accumulated distance, $d_A(m, n, v)$ as:

$$d_A(m, n, v) = d(m, n, v) + \min_{n-2 \leq j \leq n} (d_A(m-1, j, v)). \tag{7.37}$$

For $2 \leq n \leq N_v$, $1 \leq v \leq V$, where $d(m, n, v)$ is the local distance between test frame $\mathbf{t}(m)$ and reference from $\mathbf{r}_v(n)$, and we assume a maximum path expansion of 2 to 1 (hence we search back by 2 reference pattern frames in the combinatoric stage). The recursion of Eq. (7.37) is carried out for all internal frames of each reference pattern (i.e., $n \geq 2$). At the reference pattern boundary, i.e., $n = 1$, we have the simple recursion

$$d_A(m, 1, v) = d(m, 1, v) + \min \left[\min_{1 \leq r \leq V} [d_A(m-1, N_r, r)], d_A(m-1, 1, v) \right]. \tag{7.38}$$

Thus the combinatorics for internal and boundary frames are as shown in Figure 7.21. Figure 7.21a shows that for internal frames the combinatorics choose the best internal path

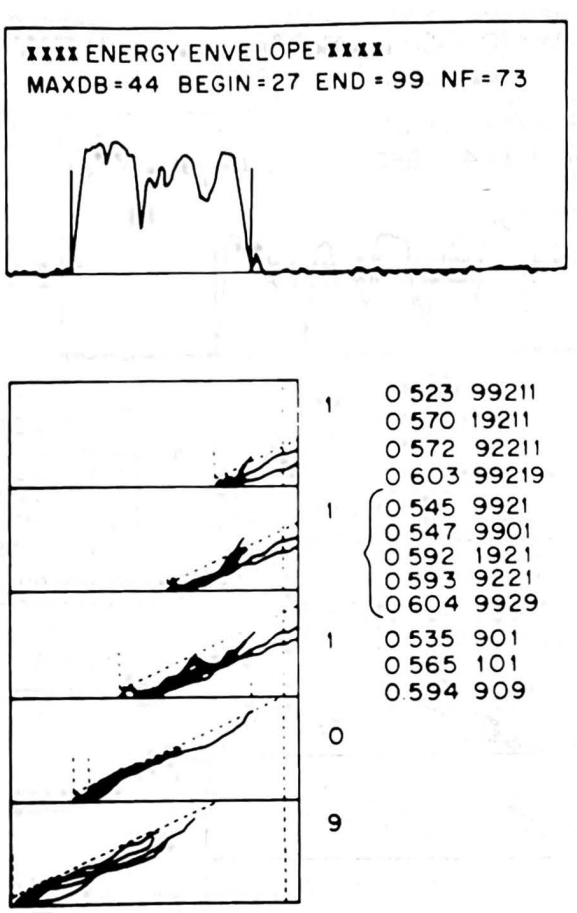


Figure 7.19 Level building of the string "99211" (after Myers and Rabiner [4]).

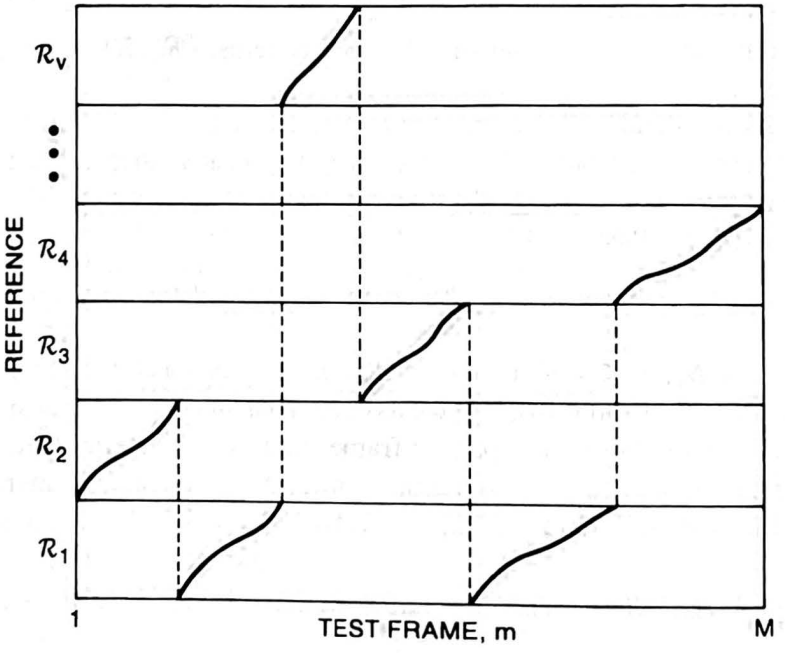


Figure 7.20 The one-pass connected word recognition algorithm (after Bridle et al. [6]).

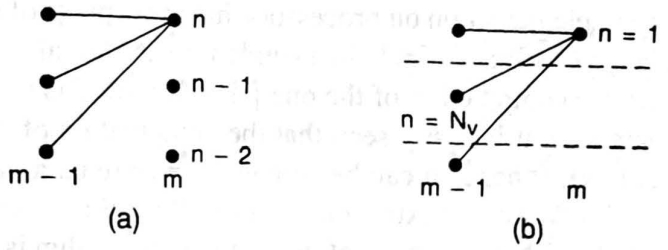


Figure 7.21 Combinatorics for the one-pass algorithm.

within the reference pattern, whereas at boundary frames the combinatorics choose either a straight (horizontal) path from within the reference pattern (subject to the constraint that the path cannot remain constant for more than one frame) or the best ending frame of any reference pattern. (Within the dynamic programming framework, of course, incorporation of a set of *local constraints* that differ from those of Eq. (7.38) is possible, subject to pragmatic considerations.)

The final solution for the best path (corresponding to the best word string) is

$$D^* = \min_{1 \le v \le V} [d_A(M, N_v, v)]. \tag{7.39}$$

Thus the one-pass algorithm computes best paths to every reference pattern frame at every test frame and eventually is able to backtrack the best score (from Eq. (7.39)) to give the best word sequence, as shown in Figure 7.20.

The major problem with the one-pass algorithm is that no mechanism is provided for controlling the resulting string length—that is, for giving a best path for a string of arbitrary length. The algorithm inherently finds a single best path whose string length is whatever it turns out to be. Thus there is no simple way of exploiting given constraints on string length within the fundamental procedure.

There is, however, a simple and straightforward way of incorporating level (i.e., string-length) constraint in the computation. We do this by extending the accumulated distance to include level information—that is, we extend the recursion to compute the accumulated distance at level ℓ as

$$d_A^\ell(m, n, v) = d(m, n, v) + \min_{n-2 \le j \le n} [d_A^\ell(m-1, j, v)] \tag{7.40}$$

where the computation is for $1 \le \ell \le L_{\max}$, $2 \le n \le N_v$, $1 \le v \le V$, $1 \le m \le M$. At each boundary frame the computation now becomes

$$d_A^\ell(m, 1, v) = d(m, 1, v) + \min \left[\min_{1 \le r \le V} d_A^{\ell-1}(m-1, N_v, r), d_A^\ell(m-1, 1, v) \right] \tag{7.41}$$

with

$$D^* = \min_{1 \le \ell \le L_{\max}} \min_{1 \le v \le V} [d_A^\ell(M, N_v, v)]. \tag{7.42}$$

The key difference is in Eq. (7.41), which only allows a path to an ending frame at level $(\ell - 1)$ to become a path at a beginning frame at level ℓ.

The main advantage of the one-pass algorithm is that the computation for a given test frame, m, can be done frame synchronously; hence the one-pass algorithm is well

suited to real-time implementation on processors that are capable of doing all the necessary computations of Eqs. (7.40) and (7.41) in a single frame interval. Although at first glance it seems as though the computation of the one-pass algorithm is significantly greater than that of the LB approach, it is easily seen that the computation of $d(m, n, v)$ of Eq. (7.40) is independent of level, ℓ; hence it can be computed once (e.g., at level 1) and stored, and used in subsequent levels with no extra computation. Because of its suitability for real-time implementation the level-based version of the one-pass algorithm is generally the one used for connected word recognition tasks.

7.6 MULTIPLE CANDIDATE STRINGS

In the previous sections we have been discussing ways of determining the globally best path through the extended grid, corresponding to the best match to the spoken word string. There are many ways of obtaining multiple candidate strings from which we can determine, at least in theory, the second-best string match, the third-best string match, etc. Multiple string candidates are particularly useful when using grammar networks in order to provide robust recognition decisions. The way in which we obtain multiple candidate strings is simple; namely, we keep track of both the *best* distance, and the *second-best* distance to each ending frame at each level (to get the second-best string match). Since we have computed all reference pattern distances already (as needed to determine the best distance), all that is required is additional storage (to keep track of the array of second-best distances) and the bookkeeping to determine the second best string. (In Chapter 4 we discussed the general problem of using dynamic programming methods to determine the N-best paths through a grid. What is described here is essentially what was called the parallel algorithm in Chapter 4, Section 4.7.5. The main difference is that here the path ranking is in terms of word candidate strings instead of frame-based time warping paths. This difference gives rise to some extra optimality considerations as discussed in this section.) We illustrate the procedure in Figure 7.22, which shows a simple two-level LB search with tracking of the two best strings at each grid point. At the end of level 2 there are two distinct paths to the last frame of the test pattern—namely, the best distance (labeled 1 and shown as a solid path) and the second best distance (labeled 2 and shown as a dashed line). The best path from level 2 branches off to two other paths in level 1, with one being a best path and the other being a second-best path. Similarly, the second-best path from level 2 branches off to two other paths in level 1. Thus a total of four paths are followed—namely, the 11 path (best from level 2, best from level 1), the 12 path (best from level 2, second best from level 1), the 21 path (second best from level 2, best from level 1 to the beginning point of the level 2 path) and the 22 path (second best from level 2, second best from level 1). The overall best path is, by definition, the 11 path through the grid. The second-best path, according to this method, is (with some pathological exceptions, to be explained next) either the 12 path, or the 21 path; the 22 path cannot ever be as good as the 21 path.

The procedure described above can be extended trivially to the best three candidates at each level, in which case a total of 3^L scores are obtained after L levels. This is illustrated at the bottom of Figure 7.22 for a two-level match in which there are nine string candidates.

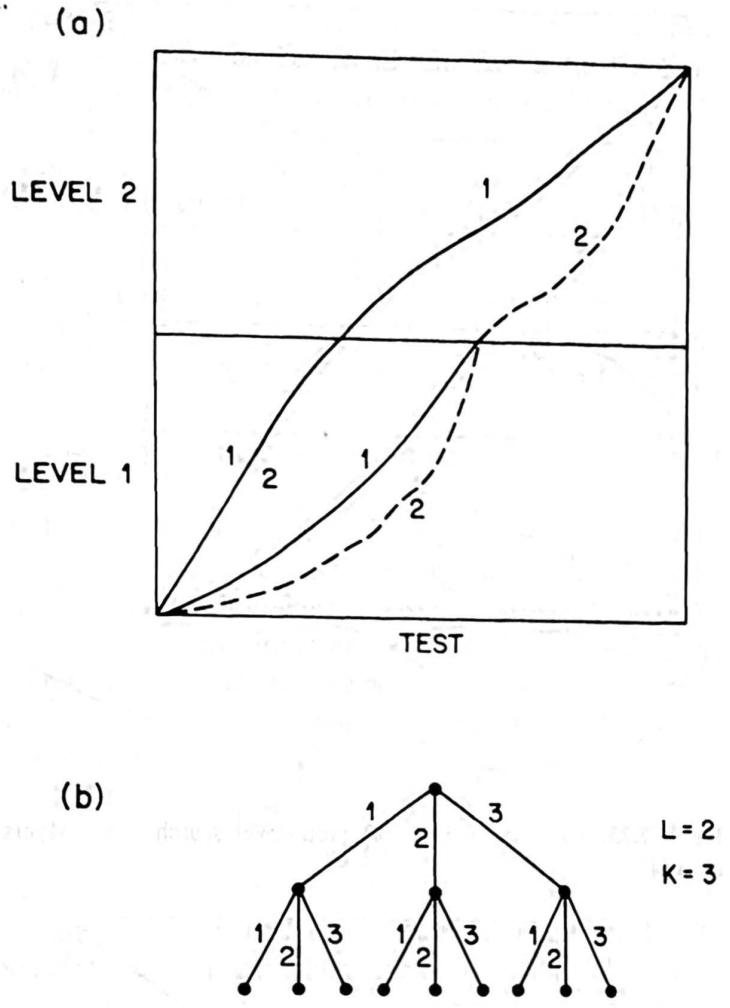


Figure 7.22 Description of procedure for determining multiple candidate scores (after Myers and Rabiner [4]).

Figure 7.23 shows a case of using $L = 4$ levels with two best candidates. There are now four possible choices for the second-best string, including the 1112 path, the 1121 path, the 1211 path and the 2111 path. A sorting algorithm can be used to order the K^L paths obtained when keeping track of K candidates at L levels.

It should be noted that the procedure described above, implemented using the level building algorithm, *cannot* guarantee that the computed "second-best" string is actually the true second-best string. This is because in keeping track of multiple strings to any ending frame the procedure inherently requires that the candidate strings come from different reference patterns. (Recall that in the LB algorithm, partial word decisions are made at each level before reaching the end of the test pattern.) The real requirement should be that they come from different overall strings (i.e., any word in the built-up strings can be different, not just the immediate last word). Hence, in theory, the two best paths to a given ending frame could indeed be from the same reference pattern. Figure 7.24 [10] illustrates the level building flaw via an example in which the true second-best string is the sequence

Ex.1022 / Page 272 of 333
AMAZON.COM, INC.

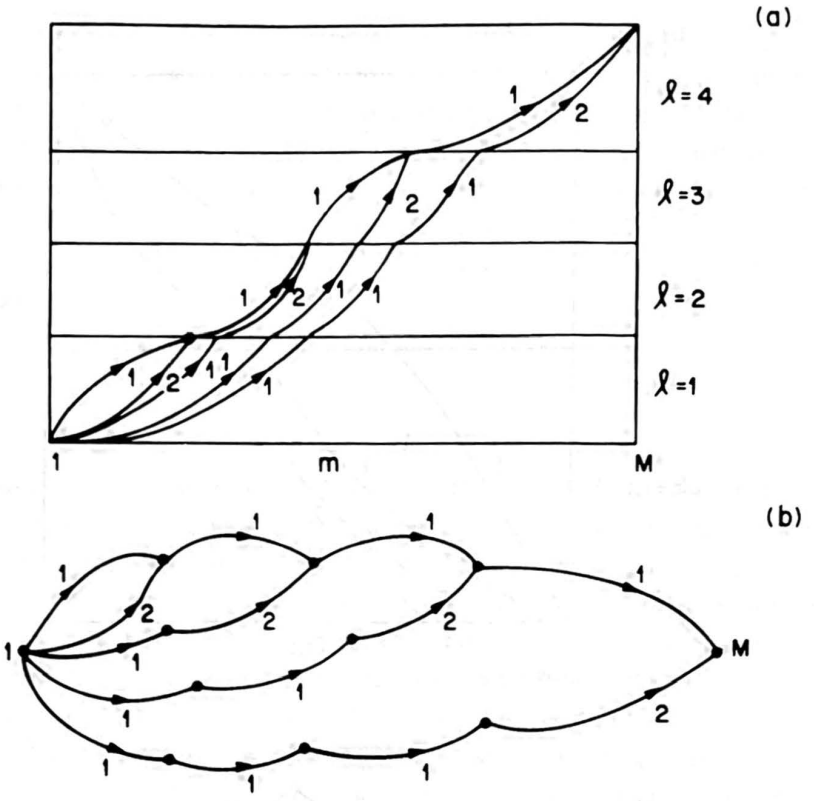


Figure 7.23 Candidate strings for a four-level search (after Myers and Rabiner [4]).

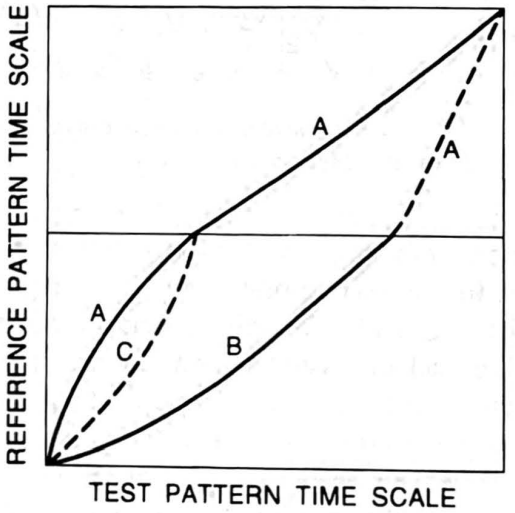


Figure 7.24 Illustration of level building flaw for determining the second-best candidate string (after Myers and Rabiner [10]).

BA (i.e., reference pattern *B* followed by *A*); however, since the best string match is *AA*, the reference pattern *A* is not allowed as the second candidate to the last frame at level 2. This situation occurs extremely infrequently; however, it can and does occur from time to time.

7.7 SUMMARY OF CONNECTED WORD RECOGNITION ALGORITHMS

We have presented three approaches to solving the "connected word recognition" problem—namely, the two-level DP algorithm, the LB method, and the (frame synchronous) one-pass method. The algorithms all are fundamentally identical in that they provide the identical best matching string with the identical matching score. Basically the algorithms differ in computational efficiency, storage requirements, and ease of realization in real-time hardware.

Although we have concentrated primarily on word template patterns, it is easy to see that the basic procedures are virtually identical for statistical models like HMMs. To see this, consider the case of level building using N-state HMMs instead of N_v-frame reference patterns. If we denote the test frame index as t, $1 \le t \le T$, (rather than n as is conventionally done for HMMs), and we denote the test frame vector as $\mathbf{o}_t$ (rather than $\mathbf{t}_n$ as we have done throughout this chapter), then the local log likelihood for state j of reference model λ^v is (for an M-mixture density)

$$b_j^v(\mathbf{o}_t) = \log \left[\sum_{m=1}^{M} c_m \prod_{d=1}^{D} e^{-(\mathbf{o}_t(d) - \mu_{jm}(d))^2 / 2\,\mathbf{U}_{jm}(d)} \right]. \tag{7.43}$$

The level building computation is therefore a calculation of $P_\ell^v(t)$, $1 \le t \le T$, $1 \le v \le V$, $1 \le \ell \le L_{\max}$, the accumulated log likelihood to frame t, at level ℓ, for reference model λ^v along the best path along with $F_\ell^v(t)$, the backpointer indicating where the best path started at the beginning of the level. At the end of each level we compute the "level best" scores as

$$P_\ell^B(t) = \max_{1 \le v \le V} P_\ell^v(t), \qquad 1 \le t \le T \tag{7.44}$$

$$N_\ell^B(t) = \arg \max_{1 \le v \le V} P_\ell^v(t), \qquad 1 \le t \le T \tag{7.45}$$

$$F_\ell^B(t) = F_\ell^{N_\ell^B(t)}(t), \qquad 1 \le t \le T \tag{7.46}$$

with solution

$$P^* = \max_{1 \le \ell \le L_{\max}} [P_\ell^B(T)]. \tag{7.47}$$

Figure 7.25 illustrates the grid for the level building computation for a set of N-state HMMs. The regularity of the grid (the trellis shape) is due to the lack of constraint about remaining in a state for more than one or two frames for statistical models.

Finally it should be clear that all three algorithms are equally amenable to inclusion (integration) of an FSN grammar network to constrain allowable word sequences. Consider

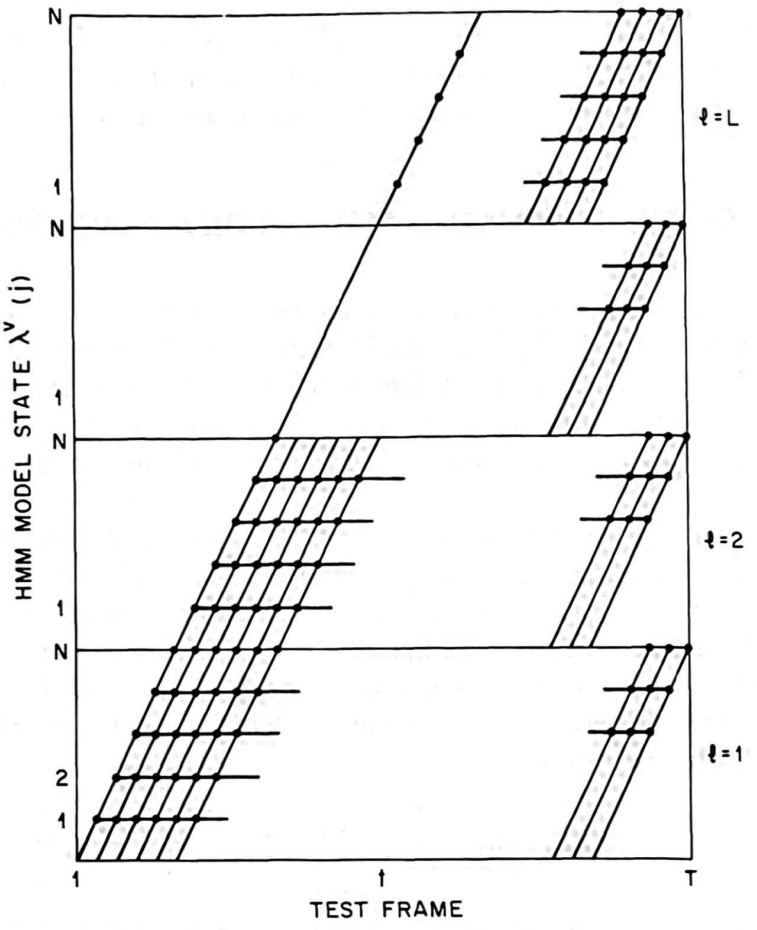


Figure 7.25 Use of HMMs in the level building procedure.

an arbitrary FSN grammar network with a typical node, g, as shown in Figure 7.26. The input to the grammar node is a series of word arcs (corresponding to words in the vocabulary) from predecessor grammar nodes $i-1, i$ and $i+1$. Since all these inputs merge at grammar node g, the basic computation at this grammar node is to determine the maximum likelihood path over the set, $P(g)$, (which constitutes the paths of all words coming into g), and to propagate this path to all successor nodes, namely $j-1, j, j+1$ through the appropriate word arcs. The grammar node computation is iterated over all nodes in the grammar in an organized fashion (so that all computation necessary for node g is done before considering node g).

The way in which computation for the grammar FSN is integrated into the connected word algorithm is shown in Figure 7.27. For each test frame, t, the spectral vector is computed, and then the local likelihood (or distance) scores are computed for every reference pattern state (or frame). In parallel the local combinatorics (within reference patterns) are performed (the local distance scores are added at the end of the computation). The grammar network scores (corresponding to word transitions) are then computed and the procedure is iterated until the last test frame, at which point a backtracker is used to determine the best matching string.

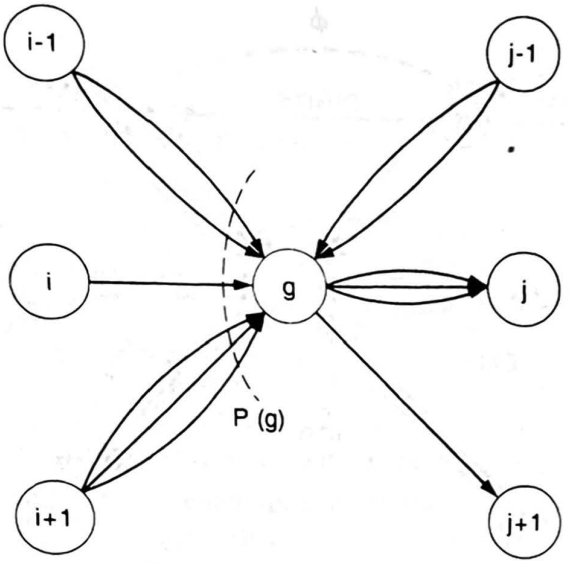


Figure 7.26 A typical grammar node of an FSN grammar network (after Lee and Rabiner [9]).

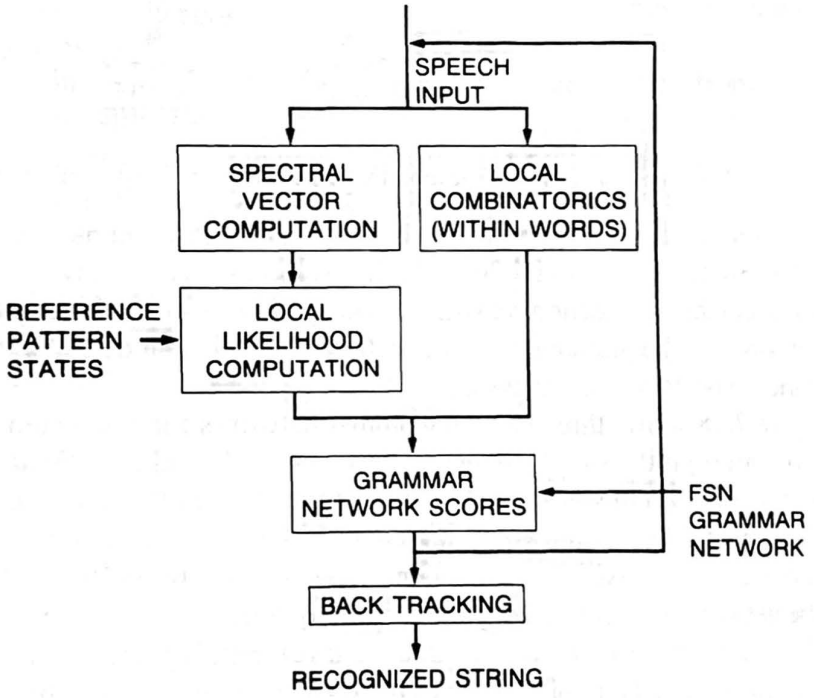


Figure 7.27 Block diagram of connected word recognition computation.

7.8 GRAMMAR NETWORKS FOR CONNECTED DIGIT RECOGNITION

One of the most important applications of connected word recognition is connected digit recognition because of its potential application to credit card entry, all-digit dialing of telephone numbers, personal identification number (PIN) entry, catalog ordering, and so on.

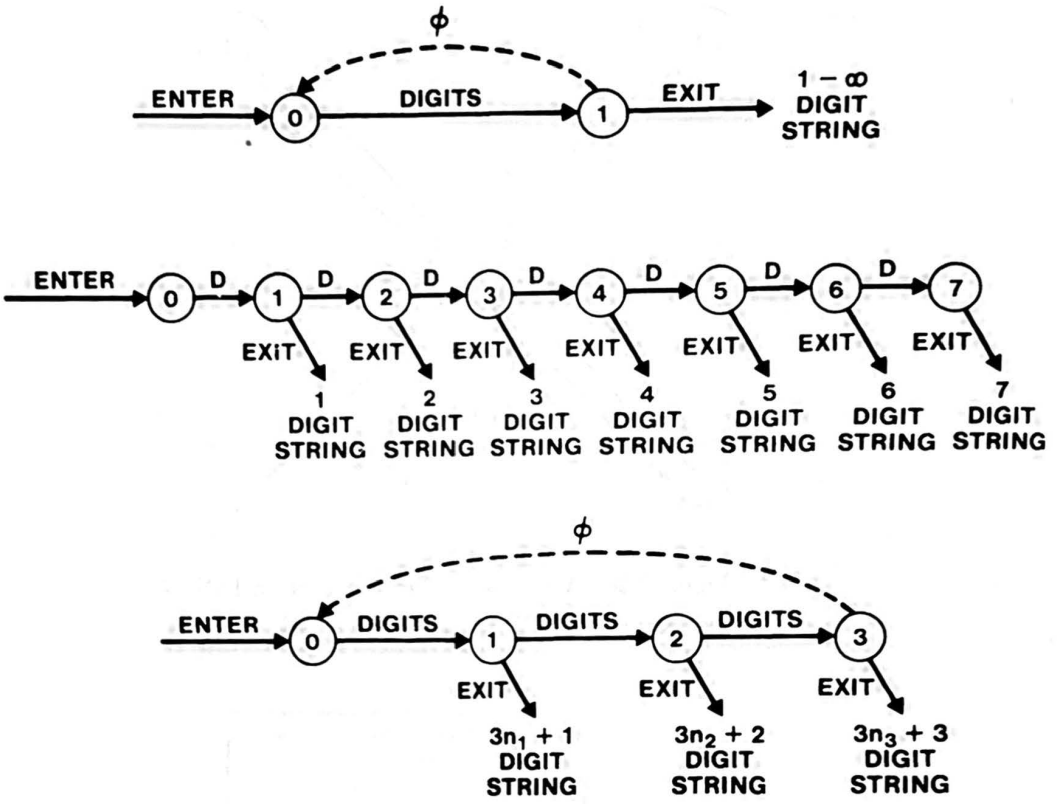


Figure 7.28 Three possible grammar networks for connected digit recognition (after Lee and Rabiner [9]).

Thus in this and the next section we will concentrate on aspects of this important application, including forms of the grammar network that are often used, and some implementation and performance aspects of current systems.

Figure 7.28 shows three general grammar networks for connected digit strings. The network of part (a) is the simple search of the one-pass algorithm *without* level information. Thus the network will always find the best string, but cannot control the string length, and therefore is unusable for known string length tasks (e.g., choosing the best seven-digit string to represent a spoken telephone number).

The network of Figure 7.28, part b explicitly breaks out digit strings of length one to seven digits, and therefore corresponds to the level building approach with $L_{\max} = 7$. As we have seen, the local combinatorics search requires about 7 times the computation of the simple grammar network of part a; however, local distance computation remains the same.

The network of Figure 7.28, part c represents a reasonable compromise between the networks of parts a and b in that it breaks out three distinct levels corresponding to string lengths of $3n_1 + 1$, $3n_2 + 2$, and $3n_3 + 3$ digits where n_1, n_2 and n_3 are arbitrary integers. The idea here is that if the input is an n-digit string, the most likely errors are single-digit insertions or deletions, at which point one of the three outputs will most likely not have the single insertion or deletion, and will be the correct string. The computation is only three times the combinatorics search of part a, with again the same computation

Ex.1022 / Page 277 of 333
AMAZON.COM, INC.

for local distances.

7.9 SEGMENTAL K-MEANS TRAINING PROCEDURE

The discussion in the previous sections focused primarily on efficient and optimal solutions to the so-called decoding problem, in which the likelihood (or distance) of a given speech pattern (the unknown test pattern) corresponding to a *string of words* is evaluated. Of equal importance is the connected word training problem in which the object is to derive appropriate word reference patterns or models from a labeled training set of many connected word sequences. The major problem here is the lack of an exact (precise) correspondence between speech segments and the spoken words they correspond to. One could consider manually segmenting and labeling each spoken training utterance into the individually spoken words of each string. However, this process is a tedious one that is well known to be error prone because of inconsistencies in determination of the exact boundaries between adjacent words in the string. Hence what is required for connected word training is a fully automatic procedure for both segmentation of a connected word string into individual words, and model (pattern) training from the segmented strings. Such an algorithm is described in this section.

The way in which the individual digit models are derived from connected word training strings is a procedure called the segmental k-mean training procedure, which is a straightforward variation on the well-known k-means iteration (e.g., as used for vector quantizer design) [11–13]. The basic idea is to have a training set of labeled connected digit strings, and an initial set of digit models (e.g., isolated digit models). (The procedure to be described works even without an initial set of digit models, but we will not describe how this is accomplished here.)

The segmental k-means training procedure (as shown in Figure 7.29) works as follows (when used for training HMMs):

1. Given the initial model (the set of word pattern files) and the (labeled) training files, any of the connected word recognition procedures is used to segment each training string into individual digit tokens which are stored in appropriate files according to the identity of the digits. (This is the segmentation phase into words.)
2. Each file of word tokens (e.g., the file for all the 1s in the training set) is then segmented into states and within each state the parameters of the mixture density (the mixture weights, means, and covariances) are determined using a standard VQ clustering procedure. The result of this procedure (which we call the word pattern building algorithm) is an updated set of word models.
3. A test for convergence is made, either based on a set of testing files or based on the likelihood scores of the training set files. If the convergence test shows continuing improvement in performance, the procedure is iterated (i.e., steps 1 and 2 are repeated using the updated set of word models); otherwise the procedure is terminated and the updated set of models is the final set of models.

Ex.1022 / Page 278 of 333
AMAZON.COM, INC.

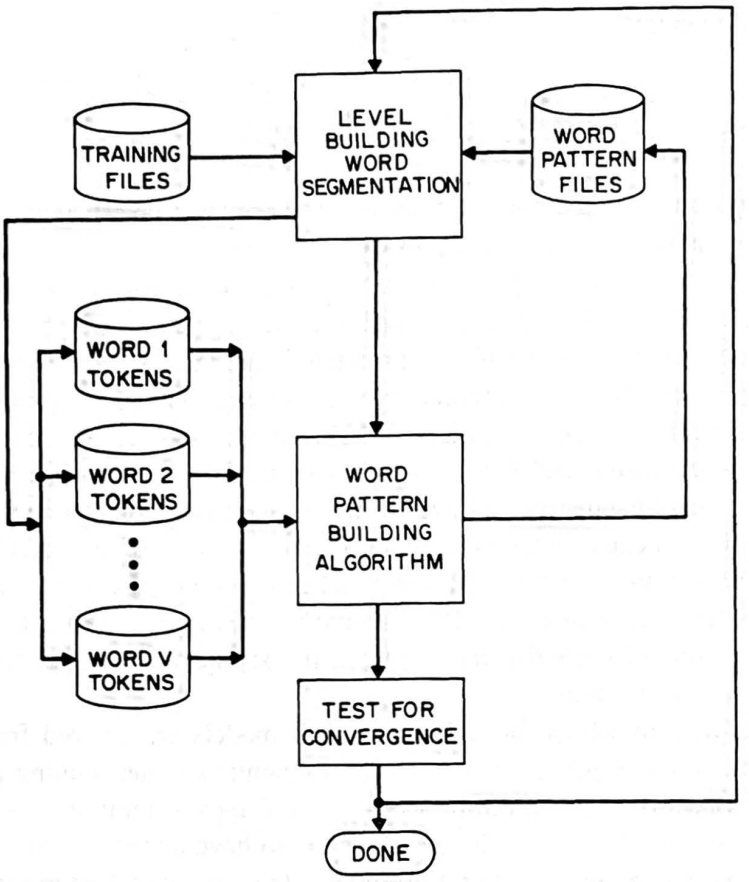


Figure 7.29 The segmental k-means training algorithm for connected word strings (after Rabiner et al. [13]).

7.10 CONNECTED DIGIT RECOGNITION IMPLEMENTATION

A block diagram of a canonic system for connected digit recognition is shown in Figure 7.30. There are three basic steps in the recognition process, namely:

1. **Spectral analysis**, in which the speech signal, $s(n)$, is converted to an appropriate spectral representation, e.g., filter-bank vector, LPC-based vector, ear model vector.
2. **Connected word pattern matching**, in which the sequence of spectral vectors of the unknown (test) connected digit string is matched against whole word (single digit) patterns using any of the algorithms discussed in this chapter. The output of this process is a set of candidate digit strings, generally of different lengths, ordered by distance (likelihood, probability) score.
3. **Postprocessing**, in which the candidate digit strings are subjected to further processing (e.g., based on digit durations, word stress, etc.) so as to eliminate unreasonable (unlikely) candidates. The postprocessor chooses the most likely digit string from the ordered list of candidates which passed the postprocessor tests.

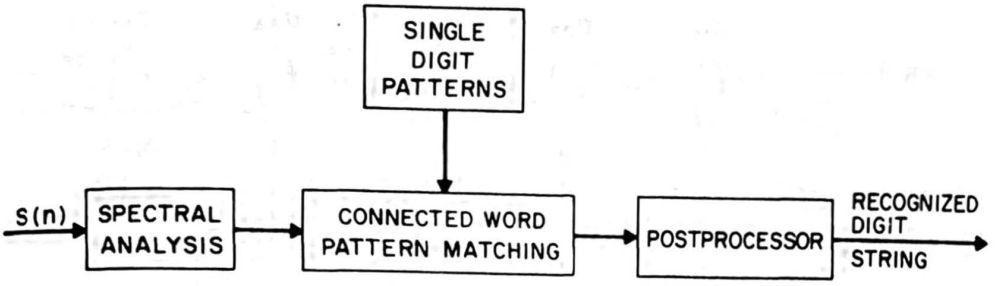


Figure 7.30 Block diagram of connected digit recognition method (after Rabiner et al. [13]).

In the remainder of this section we present an overview of the techniques that are currently used to provide the best performance (highest string accuracy) on this task.

7.10.1 HMM-Based System for Connected Digit Recognition

The system that provides the highest reported string accuracy on a standard testing set is one based on LPC cepstral analysis and HMMs. In particular, the spectral analysis uses an LPC front end with the following characteristics:

- sampling rate—6.67 kHz
- analysis window size—300 samples (45 msec)
- analysis window shift—100 samples (15 msec)
- LPC order—8
- cepstrum order—12
- delta cepstrum order—12
- delta-delta (second difference) cepstrum order—12
- cepstral window—raised, sinelike window

The observation vector used was a 38-component vector consisting of 12 cepstral coefficients, 12 delta cepstrum coefficients, 12 delta-delta cepstral coefficients, delta log energy, and delta-delta log energy. (The log energy was used directly in the evaluation of model likelihoods on the basis of measured histograms, rather than as a component of the observation vector.)

The hidden Markov models used for each digit model were left-to-right models of the type shown in Figure 7.31. Each model had N states (N varied from 5 to 10 for different digits), and within states a continuous mixture density was used to characterize the observation vector, where the number of mixture components per state was as few as 3 (for speaker-trained models) and as many as 64 (for speaker independent models). In addition to the spectral density, an empirically derived log energy probability density (e.g., a histogram) was used within each state (with appropriate weighting), as well as an empirically derived state duration density. For the postprocessor a single Gaussian digit duration density was used based on the measured mean duration and variance from the training set.

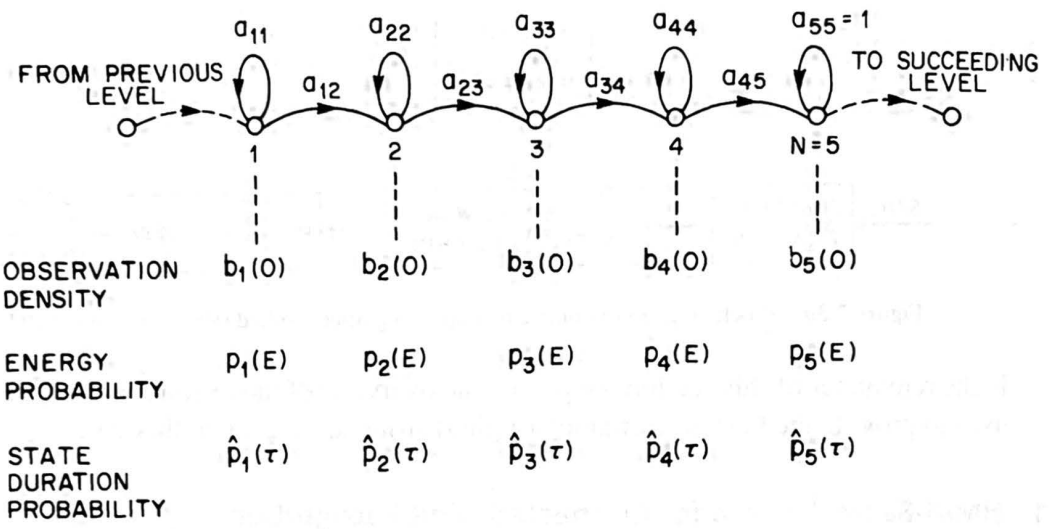


Figure 7.31 Connected digit HMM (after Rabiner et al. [13]).

7.10.2 Performance Evaluation on Connected Digit Strings

To evaluate the performance of connected digit recognition algorithms ([13–15]), two databases have been used:

1. **DB50**, consisting of 50 adult talkers (25 male, 25 female), from the local, nontechnical population of Murray Hill, New Jersey. Each talker provided from 600 to 1150 digit strings, for a total of 47,336 strings. The digit strings were recorded off of local, dialed-up, telephone lines (100-3200 Hz bandwidth), and were variable in length from 1 to 7 digits. Within each string the selection of digits was random from the set of zero to nine (oh was excluded). All digit strings were spoken fluently with no pauses.
2. **TI Set**, consisting of a training set of 112 talkers (55 male, 57 female) with 22 regional accents, and a testing set of 113 additional talkers (56 male, 57 female) from the same 22 regions with no overlap between training and testing set talkers. Each talker spoke 77 connected digit strings of length 1–5 or 7 digits. The input provided was wideband (100–7000 Hz) but was lowpass filtered to telephone bandwidth at Bell Labs for compatibility with the DB50 set. There was a random selection of digits within each string from the set of 11 digits including both zero and oh. Even though talkers were requested to speak each string fluently with no pauses, a few strings had internal pauses.

The first data base, DB50, was used for tests in a speaker-trained and a multispeaker mode, whereas the second database, TI Set, was used for speaker-independent tests. To contrast the spoken material in the two sets, Figure 7.32 shows a plot of the average speaking rate (measured in words per minute) of the two datasets as a function of the number of digits in the string. It can be seen that the speaking rate of the TI talkers is somewhat lower than that of the DB50 talkers (by from 10-20 wpm).

Ex.1022 / Page 281 of 333
AMAZON.COM, INC.

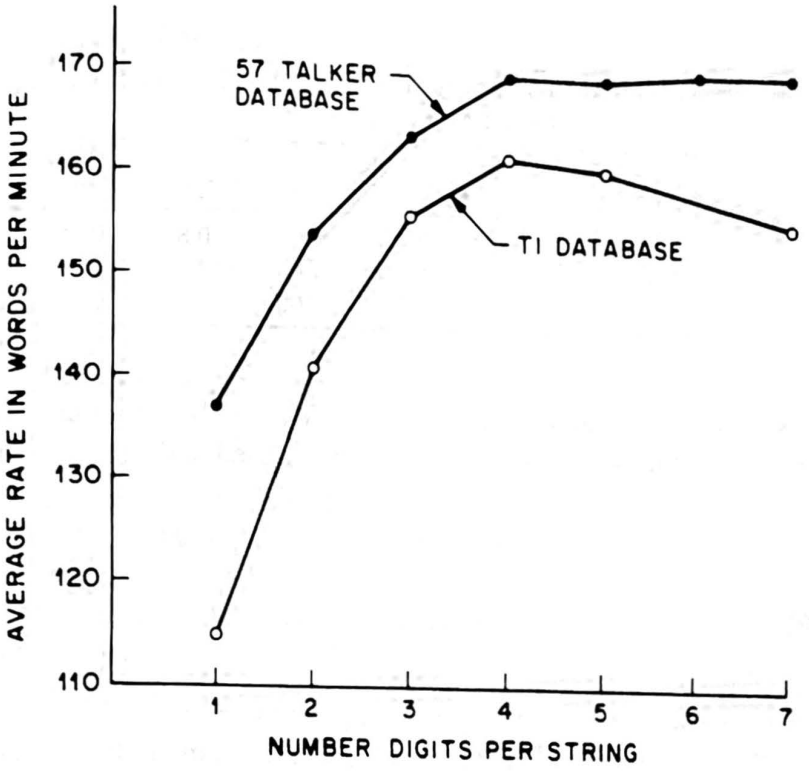


Figure 7.32 Average speaking rate of talkers in the two connected digit databases as a function of the number digits per string (after Rabiner et al. [13]).

The conditions used in the performance evaluations were as follows (when used for training HMMs):

1. **Speaker-Trained Mode (50 Talkers)**—For each talker, half the strings (randomly selected) were used for training, the other half for testing. A single HMM per digit was used. (For this test the observation vector did not use either energy or delta-delta cepstral components.)
2. **Multispeaker Mode (50 Talkers)**—One-fourth of the training set, as in the speaker-trained mode, was used for training 6 models per digit (using clustering techniques); the same testing set was used as in the speaker trained mode. (Again the delta-delta cepstral components or energy were not used in the observation vectors.)
3. **Speaker-Independent Mode (225 Talkers)**—The specified training and testing sets were used to create a single HMM per digit (based on the 38 parameter observation vectors). No silence model was created to account for pauses within digit strings.

The resulting performance of the connected digit recognizer is shown in Table 7.1. For each of the three testing conditions, results are given for self-test (namely, testing on the training data) and for the true testing set, in terms of average *string* error rates (%) for unknown length (UL) strings (i.e., allowing insertions and deletions as well as substitutions), as well as for known length (KL) strings (i.e., only allowing strings of the correct length). Performance in all three modes is comparable with string error rates around 1–3% for unknown length strings, and 0.4–1.7% for known length strings. (Performance

TABLE 7.1. Average String Error Rates (%) for Connected Digit Recognition Tests

Mode	Training Data		Testing Data	
	UL	KL	UL	KL
Speaker Trained	0.4	0.16	0.8	0.35
Multi-Speaker	1.7	1.0	2.85	1.65
Speaker Independent	0.3	0.05	1.4	0.8

scores for speaker-trained and multispeaker modes are underbounds, since the tests were not performed with the extended observation vector used in the speaker-independent mode tests.)

7.11 SUMMARY

In this chapter we have shown how the results presented in earlier chapters on time alignment of single words and phrases can be extended to the problem of aligning an input consisting of a sequence of words to a concatenation of individual word patterns. We have shown how an optimal matching can be achieved using one of several different procedures, and we have also shown how the basic ideas can be applied to either templates or statistical models. By way of example, we showed how the techniques described have been applied to the problem of recognizing a fluently spoken string of digits with high performance.

REFERENCES

[1] H. Sakoe, "Two-Level DP Matching—A Dynamic Programming-Based Pattern Matching Algorithm for Connected Word Recognition," *IEEE Trans. Acoustics, Speech, Signal Proc.*, ASSP-27 (6): 588–595, December 1979.

[2] H. Sakoe, "A Generalized Two-Level DP-Matching Algorithm for Continuous Speech Recognition," *Trans. of the IEE of Japan*, E65 (11): 649–656, November 1982.

[3] C.S. Myers and L.R. Rabiner, "A Level Building Dynamic Time Warping Algorithm for Connected Word Recognition," *IEEE Trans. Acoustics, Speech, Signal Proc.*, ASSP-29 (2): 284–297, April 1981.

[4] C.S. Myers and L.R. Rabiner, "Connected Digit Recognition Using a Level-Building DTW Algorithm," *IEEE Trans. Acoustics, Speech, Signal Proc.*, ASSP-29 (3): 351–363, June 1981.

[5] T.K. Vintsyuk, "Element-Wise Recognition of Continuous Speech Consisting of Words From a Specified Vocabulary," *Kibernetika* (Cybernetics), No. 2: 133–143, March–April 1971.

[6] J.S. Bridle, M.D. Brown, and R.M. Chamberlain, "An Algorithm for Connected Word Recognition," *Proc. ICASSP 82*, Paris, 899–902, May 1982.

Ex.1022 / Page 283 of 333
AMAZON.COM, INC.

[7] J.S. Bridle, M.D. Brown, and R.M. Chamberlain, "Continuous Connected Word Recognition Using Whole Word Templates," *The Radio and Electronic Engineer*, 53 (4): 167–175, April 1983.

[8] H. Ney, "The Use of a One-Stage Dynamic Programming Algorithm for Connected Word Recognition," *IEEE Trans. Acoustics, Speech, Signal Proc.*, ASSP-32 (2): 263–271, April 1984.

[9] C.H. Lee and L.R. Rabiner, "A Frame-Synchronous Network Search Algorithm for Connected Word Recognition," *IEEE Trans. Acoustics, Speech, Signal Proc.*, 37 (11): 1649–1658, November 1989.

[10] C.S. Myers and L.R. Rabiner, "A Comparative Study of Several Dynamic Time-Warping Algorithms for Connected-Word Recognition," *Bell System Tech. J.*, 60 (7): 1389-1409, September 1981.

[11] B.H. Juang and L.R. Rabiner, "The Segmental *K*-Means Algorithm for Estimating Parameters of Hidden Markov Models," *IEEE Trans. Acoustics, Speech, Signal Proc.*, 38 (9): 1639–1641, September 1990.

[12] L.R. Rabiner, J.G. Wilpon, and B.H. Juang, "A Segmental *K*-Means Training Procedure for Connected Word Recognition," *AT&T Tech. J.*, 64 (3): 21–40, May 1986.

[13] L.R. Rabiner, J.G. Wilpon, and B.H. Juang, "A Model-Based Connected Digit Recognition System Using Either Hidden Markov Models or Templates," *Computer Speech, and Language*, 1 (2): 167–197, December 1986.

[14] L.R. Rabiner, J.G. Wilpon., and F.K. Soong, "High Performance Connected Digit Recognition Using Hidden Markov Models," *IEEE Trans. Acoustics, Speech, Signal Proc.*, 37 (8): 1214–1225, August 1989.

[15] J.G. Wilpon, C.H. Lee, and L.R. Rabiner, "Improvements in Connected Digit Recognition Using Higher Order Spectral and Energy Features," *Proc. ICASSP 91*, Toronto, Ontario, Canada, May 1991.

Chapter 8

LARGE VOCABULARY CONTINUOUS SPEECH RECOGNITION

8.1 INTRODUCTION

Throughout this book we have developed a wide range of tools, techniques, and algorithms for attacking several fundamental problems in speech recognition. In the previous chapter we saw how the different techniques came together to solve the connected word recognition problem. In this chapter we extend the concepts to include issues needed to solve the large vocabulary, continuous speech recognition problem. We will see that the fundamental ideas need modification because of the use of subword speech units; however, a great deal of the formalism for recognition, based on word units, is still preserved.

The standard approach to large vocabulary continuous speech recognition is to assume a simple probabilistic model of speech production whereby a specified word sequence, W, produces an acoustic observation sequence Y, with probability $P(W, Y)$. The goal is then to decode the word string, based on the acoustic observation sequence, so that the decoded string has the maximum a posteriori (MAP) probability, i.e.,

$$\hat{W} \ni P(\hat{W}|Y) = \max_{W} P(W|Y). \tag{8.1}$$

Using Bayes' Rule, Equation (8.1) can be written as

$$P(W|Y) = \frac{P(Y|W)P(W)}{P(Y)}. \tag{8.2}$$

Ex.1022 / Page 285 of 333
AMAZON.COM, INC.

Since $P(Y)$ is independent of W, the MAP decoding rule of Eq. (8.1) is

$$\hat{W} = \arg\max_{W} P(Y|W)P(W). \tag{8.3}$$

The first term in Eq. (8.3), $P(Y|W)$, is generally called the acoustic model, as it estimates the probability of a sequence of acoustic observations, conditioned on the word string. The way in which we compute $P(Y|W)$, for large vocabulary speech recognition, is to build statistical models for subword speech units, build up word models from these subword speech unit models (using a lexicon to describe the composition of words), and then postulate word sequences and evaluate the acoustic model probabilities via standard concatenation methods. Such methods are discussed in Sections 8.2–8.4 of this chapter.

The second term in Eq. (8.3), $P(W)$, is generally called the language model, as it describes the probability associated with a postulated sequence of words. Such language models can incorporate both syntactic and semantic constraints of the language and the recognition task. Often, when only syntactic constraints are used, the language model is called a grammar and may be of the form of a formal parser and syntax analyzer, an N-gram word model ($N = 2, 3, \ldots$), or a word pair grammar of some type. Generally such language models are represented in a finite state network so as to be integrated into the acoustic model in a straightforward manner. We discuss language models further in Section 8.5 of this chapter.

We begin the chapter with a discussion of subword speech units. We formally define subword units and discuss their relative advantages (and disadvantages) as compared to whole-word models. We next show how we use standard statistical modeling techniques (i.e., hidden Markov models) to model subword units based on either discrete or continuous densities. We then show how such units can be trained automatically from continuous speech, without the need for a bootstrap model of each of the subword units. Next we discuss the problem of creating and implementing word lexicons (dictionaries) for use in both training and recognition phases. To evaluate the ideas discussed in this chapter we use a specified database access task, called the DARPA Resource Management (RM) task, in which there is a word vocabulary of 991 words (plus a silence or background word), and any one of several word grammars can be used. Using such a system, we show how a basic set of subword units performs on this task. Several directions for creating subword units which are more specialized are described, and several of these techniques are evaluated on the RM task. Finally we conclude the chapter with a discussion of how task semantics can be applied to further constrain the recognizer and improve overall performance.

8.2 SUBWORD SPEECH UNITS

We began Chapter 2 with a discussion of the basic phonetic units of language and discussed the acoustic properties of the phonemes in different speech contexts. We then argued that the acoustic variability of the phonemes due to context was sufficiently large and not well understood, that such units would not be useful as the basis for speech models for recognition. Instead, we have used whole-word models as the basic speech unit, both for