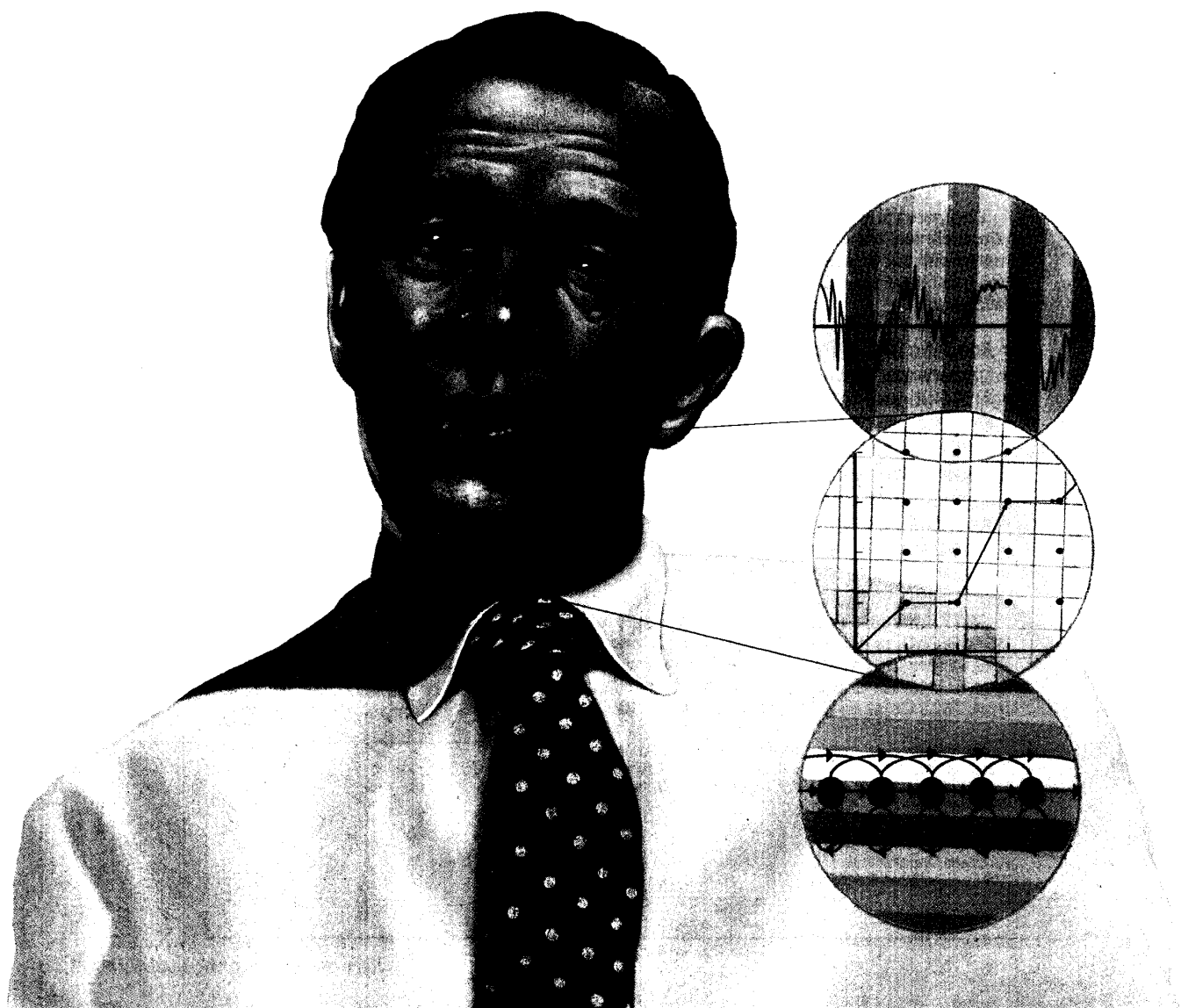


# The Application of Dynamic Programming to Connected Speech Recognition

*Harvey F. Silverman and David P. Morgan*

---



---

The Principle of Optimality and one method for its application, dynamic programming, was popularized by Bellman in the early 1950's. Dynamic programming was soon proposed for speech recognition and applied to the problem as soon as digital computers with sufficient memory were available, around 1962. Today, most commercially available recognizers and many of the systems being developed in research laboratories use dynamic programming, typically to address the problem of the time alignment between a speech segment and some template or synthesized speech artifact. In this tutorial paper, the application of dynamic programming to connected-speech recognition is introduced and discussed. The deterministic form, used for template matching for connected speech, is described in detail. The stochastic form, ordinarily called the Viterbi algorithm, is also introduced.

---

Dynamic programming (DP) is a mathematical concept used for the analysis of sequential-decision processes [1]. As with all good ideas, the basic principle had been discussed and applied throughout the history of mathematics. However, the exploitation of the concept may be traced to the post-war era, including Masse's study of water resources in 1946 [2], Wald's study of sequential-decision problems in statistics in 1947 [3], a related study by Arrow, Blackwell, and Girshick in 1949 [4], studies of the control of inventories by Arrow, Harris, and Marshak in 1951 [5], by Dvoretzky, Kiefer and Wolfowitz in 1952 [6], [7], and the study by Bellman of functional equations in 1952 [8]. While Isaacs [9] introduced a *tenet of transition* in 1951, Bellman developed another version which he called *The Principle of Optimality*. The latter energetically applied his principle to analyze hundreds of optimization problems in the area of mathematics, operations research, economics and engineering, among others. Many of these works are compiled into two books, the first in 1957 [10], and the second (with Dreyfus) in 1962 [11]. The **name** of the algorithm, *dynamic programming*, was probably affixed because, at that time, decision-making, mathematical methods which could be solved on early digital computers often had the word "programming" in their name, such as linear programming and quadratic programming. Decision making for dynamical systems, or feedback decision making, probably implied "dynamic programming" [12]. The words in the name sometimes confuse modern practitioners, given the numerous, current aspects of "programming" and the typical non-dynamical-system applications of the technique.

Theoretically, the idea of dynamic programming is based on a very simple property of multistage decision processes called the Principle of Optimality:

#### The Principle of Optimality:

*An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must*

*constitute an optimal policy with regard to the state resulting from the first decision* [11].

The Principle implies that a final optimal policy (or path) depends only upon the initial state and upon making only optimal decisions for any intermediate decisions along the way. This is best seen by means of a simplified -- meaning somewhat contrived -- example such as the sentence recognition problem described in the next section. Following that, the framework for using dynamic programming for speech recognition will be developed, first in general, and subsequently for discrete utterance recognition (DUR) and connected speech recognition (CSR). The *Viterbi algorithm*, which is a form of dynamic programming for a stochastic system, is described in the final section.

#### AN EXAMPLE TO INTRODUCE DYNAMIC PROGRAMMING CONCEPTS

Consider the problem of constructing the "best" sentence from the output of a simple DUR recognizer having a four word vocabulary -- **cat, fat, sat, that** -- mathematically referred to as words 1-4. That is  $w(1) = \text{cat}$ , etc. Furthermore, in this example, four words have been spoken and recognition decisions, in the form of the 16 probabilities,  $P(i,n)$  that  $w(i)$  was uttered at time  $n$  where  $i, n \in [1,4]$ , are known and given in

	Time(1)	Time(2)	Time(3)	Time(4)
<b>w(1)='cat'</b>	0.229	0.242	0.300	0.226
<b>w(2)='fat'</b>	0.257	0.253	0.200	0.290
<b>w(3)='sat'</b>	0.243	0.227	0.267	0.290
<b>w(4)='that'</b>	0.271	0.273	0.233	0.194

Table 1.

Probability that word  $i$  is Uttered at Time  $n, P(i,n)$

Table I. Notice, as the vocabulary items are similar, the probabilities vary little from being equally likely.

It is also assumed that a large number of sentences made up from the vocabulary have been analyzed, and that the following table of probabilities for the *bigram* statistics,  $Q(i|j)$  -- the probability that word  $i$  is uttered at time  $n+1$ , given that word  $j$  has been uttered at time  $n$  (Table II).

Word at $n+1$	Word Uttered at Time $n$				
	$w(0) = \text{silence}$	$w(1) = \text{'cat'}$	$w(2) = \text{'fat'}$	$w(3) = \text{'sat'}$	$w(4) = \text{'that'}$
$w(1) = \text{'cat'}$	0.10	0.05	0.50	0.10	0.45
$w(2) = \text{'fat'}$	0.40	0.10	0.20	0.30	0.45
$w(3) = \text{'sat'}$	0.10	0.50	0.10	0.30	0.05
$w(4) = \text{'that'}$	0.40	0.35	0.20	0.30	0.05

**Table II.**  
Probability that Word  $i$  is Uttered at Time  $n+1$ , given that Word  $j$  is Uttered at Time  $n$ ,  $Q(i|j)$

One should note that the word  $w(0) \equiv \text{silence}$  has been added in Table II in order to give the problem a tractable starting condition. These data may be combined into an optimality problem; that is, one desires to find the most likely four-word sentence given what is known from the recognizer, Table I, and what is known about the language or grammar, Table II. Any possible sentence in this grammar manifests itself through an ordered set of word numbers,  $i_n$ ,  $n \in [1, 4]$ . For example, if  $i_1 \dots i_4$  were 3,1,4,1 respectively, then the sentence would be **Sat cat that cat!** For simplicity in notation, let  $i$  represent the vector of these four numbers for some allowed sentence. Then, given some  $i$ , one may define a *correctness measure*  $C$ , that quantifies the possibility that this sentence was uttered. Namely,

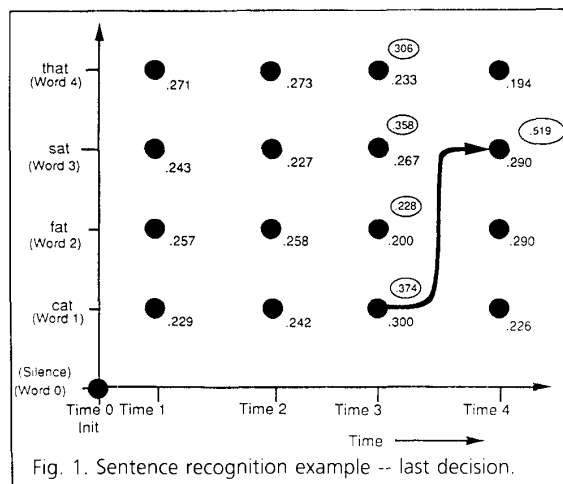
$$C(\vec{i}) = \sum_{n=1}^4 Q(i_n | i_{n-1}) P(i_n, n) \quad (2.1)$$

where  $i_0 \equiv 0$ . The optimality problem is then to find that particular sentence, denoted by  $\vec{i}$ , which maximizes the measure  $C$ . Namely,

$$\vec{i} = \underset{\vec{i}}{\operatorname{argmax}} C(\vec{i}) \quad (2.2)$$

The brute-force approach to the example problem would have one evaluate  $C$  over all  $4^4 = 256$  allowable sentences, and simply select the best one. This is not a terrible price to pay here, but if only a ten-word vocabulary, rather than four, were to be used to make ten-word sentences, then  $10^{10}$  sentences would have to be evaluated! It is not difficult to see that the total search gets out-of-hand very quickly.

The principle of optimality formalizes ideas of common sense in selecting the best sentence for this example. While there are several approaches to solving the problem, one very logical scenario becomes apparent to the logical individual. Refer to Figure 1 in which the example problem is shown graphically. One wants to determine the best path starting at the circle *init*, through each column, and ending at one of the circles of the column *Time 4*. Numbers to the lower right of the circles are the recognition probabilities lifted from Table I. In Figure 1 at *Time 4*, assume one hypothesized that **sat** was the last word in the sentence. Then, as one is only looking for the *highest* (best) value of  $C$ , one can see that the best four-word sentence ending in **sat** must be a path extension from one of the best



allowed three-word sentences ending at each member of the vocabulary at *Time 3*! This realization that only one path (rather than, here, each of the 16 paths which are possible from *init* to a word at *Time 3*) needs to be considered for extension is the critical dynamic-programming idea. Thus, if  $D(i,j)$  is defined as the maximum value of  $C$  for a sentence of  $i$  words to have ended in word  $j$ , then this idea implies that  $D(i,j)$  may be recursively computed from

$$D(i,j) = \max_{1 \leq k \leq 4} \{D(i-1,k) + Q(j|k) \cdot P(j,i)\}. \quad (2.3)$$

For the specific case shown in Figure 1 for four-word sentences ending in **sat**,

$$D(4,3) = \max_{1 \leq k \leq 4} [D(3,k) + Q(3|k) \cdot P(3,4)]. \quad (2.4)$$

The numbers in the ellipses in Figure 1 are the values  $D(i,j)$ , and the computations using Equation 2.4 are given below in Table III.

k	Word	D(3,k)	+	Q(3 k)	·	P(3,4)	=	Value for Max
1	cat	0.374	+	0.500	·	0.290	=	0.519
2	fat	0.228	+	0.100	·	0.290	=	0.257
3	sat	0.358	+	0.300	·	0.290	=	0.445
4	that	0.306	+	0.050	·	0.290	=	0.321

**Table III.**  
Computations for Figure 1

Taking the maximum over the last column, it becomes clear that the maximum value of  $C$  occurs when the *best* three-word path ending with **cat** is extended as shown in Figure 1. Similarly, by hypothesizing that the last word in the sentence is each of the other three words, the same procedure may be used to discover optimal, with respect to the measure  $C$ , extensions to each. Moreover, the recursion of Equation 2.3 may be applied to compute optimal sentences of length  $n$ , ending at each word in the vocabulary. This procedure must be implemented from left-to-right under the assumption that all sentences originate at Time 0 in the state *init*. The results from

implementing this forward recursion -- the principal cost of the dynamic-programming algorithm -- are shown in Figure 2. Once again, the values of  $D(i,j)$  are given in the ellipses. The optimal extension paths are depicted with arrows indicating the left-to-right nature of this forward process.

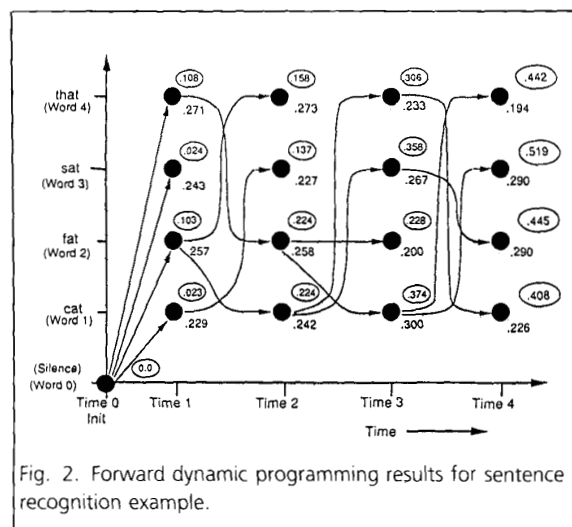


Fig. 2. Forward dynamic programming results for sentence recognition example.

The best, with respect to measure  $C$ , four-word sentence may be determined by finding the maximum value of  $D(4,j)$  over  $j$ . Here, the sentence is found to end in the word **sat** and have a value of 0.519. Unfortunately, the result of the computations shown in Figure 2 does not give all the information generally needed; only the value  $D(i,j)$  and the final word of the sentence are known immediately. When it is necessary, as in this example, that the optimal path be known, then an additional backward search is required. This property is what causes dynamic programming to be known as a two-stage procedure. In order to perform the backward search, the optimal predecessor to each word in a sentence must have been remembered. That is, for each black circle (node) in Figure 2 (except **init**), some indicator, such as the word number of the optimal predecessor, must be kept for the back-tracking stage. As long as each node holds this *back-pointer* information, one may quickly retrace the optimal path recursively. This is shown in Figure 3, and the *best* sentence is decoded as **That fat cat sat**. One should note that the back-tracking process requires only a very small amount of work relative to the forward dynamic-programming process.

The principle cost for the DP search lies in the forward stage; this computational cost is much smaller than that of a full search. In the example, a full search involved the computation of 256 sentences, each requiring the computation of the sum of four products. The forward DP process computes only 16 values, albeit that each one is somewhat more complex; i.e., in the worst case, four sums of products, four other additions, and a maximization are required as one can see from Equation 2.4. This small amount of additional complexity is, however, usually very acceptable given the savings over the full search. In general terms, if one had  $N$  words in a sentence and an  $M$ -word vocabulary, then the full-search method would require  $O(N \cdot$

$M^4)$  simple products to be computed. DP would require  $O[N \cdot N \cdot M]$  slightly more complex computations.

A few more observations are in order before moving on to the more specific applications to speech recognition. First, in the example, **all** words are considered as potential predecessors in

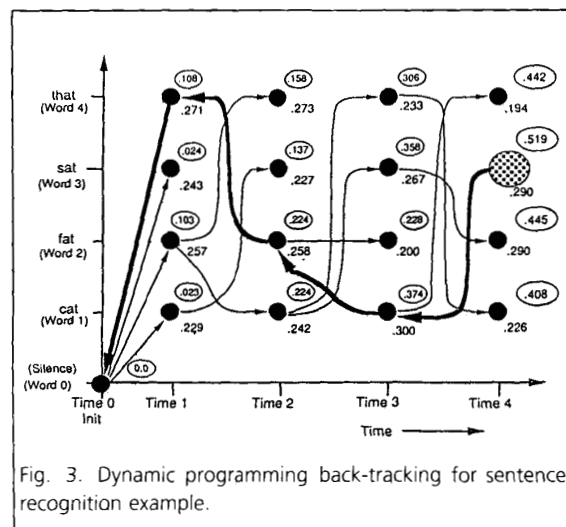
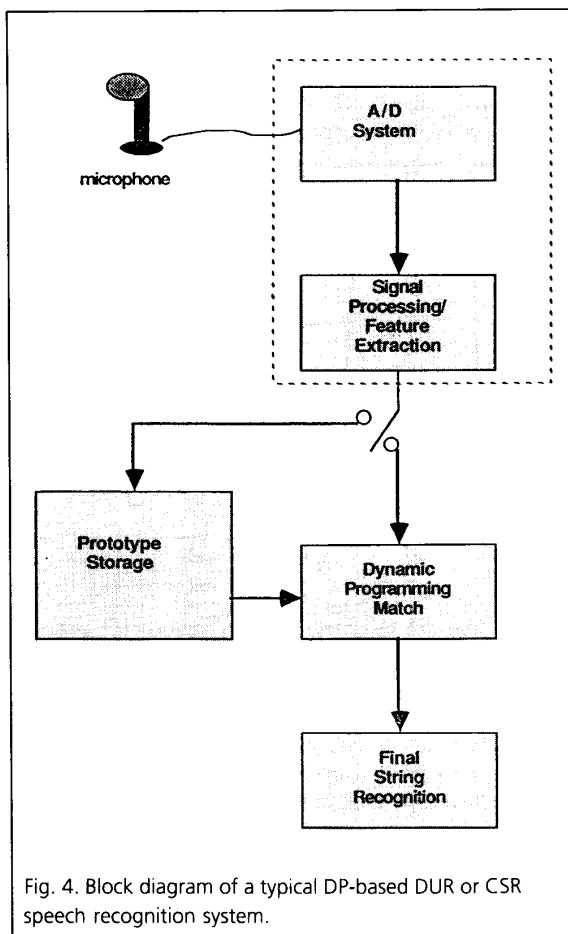


Fig. 3. Dynamic programming back-tracking for sentence recognition example.

each computation of the forward process. Typically, the set of allowed predecessors is limited by some set of constraints. For example, two verbs or articles in a row might not be permitted. This kind of local imposition is called a *local constraint* and will be seen to be an important concept for speech recognition applications. Another concept is that of a *global constraint*. Usually, a global constraint may be derived by looking at the extremes of the total search area; global limits are imposed on the basis of prior knowledge in combination with fallout from the local constraint. In the example, no global constraint is evident; this concept will become clear, however, when speech applications are discussed.

## DYNAMIC PROGRAMMING AS APPLIED TO SPEECH RECOGNITION

Dynamic programming has been an important tool both for discrete-utterance recognition (DUR) and connected-speech recognition (CSR) Systems. A block diagram for a typical DP-based speech recognition system is shown in Figure 4. The task of the first two blocks of the system is to develop a pattern, or *feature vector* for some short time interval of the speech signal. Typically, feature vectors are based on from 5 to 50ms of the speech data and are of dimension of from 5 to 50. Analysis intervals are usually overlapped (correlated) [13], although there are some systems, typically synchronous, where the analysis intervals are not overlapped (independent) [14]. For a recognizer to work near real time, this phase of the system must operate in real time. Until the advent of inexpensive DSP chips [15], [16], [17], [18], this aspect of a recognizer was always a problem; early, "inexpensive" recognizers used analog solutions [19].



about the semantic component of the speech [25]. Given this variety, it is fortunate that the DP mechanism is essentially independent of the specific feature vector selected. For purposes of this tutorial, it is probably easiest for the reader to think of a feature vector as the  $L$  log-energy values of the spectrum of a short interval of speech.

Any feature vector is a function of  $l$ , the index on the component of the vector (index on frequency for log-energy features) and a function of the time index  $i$ . The latter may or may not be linearly related to real time. For *asynchronous analysis*, the speech interval for analysis is advanced a fixed amount for each feature vector which implies  $i$  and time are linearly related. For *synchronous analysis* the interval of speech used for each feature vector varies as a function of the pitch and/or events in the speech signal itself, in which case  $i$  will only index feature vectors and must be related to time through a lookup table. All this implies,

$$i^{\text{th}} \text{ feature vector} \equiv f(i, l) \quad (3.1)$$

Each pattern or data stream for recognition, therefore, may be visualized as an ensemble of feature vectors. For the case of a log-energy-spectrum feature vector, a digital *spectrogram* results, an example of which is given in Figure 5 for the words "speech sounds." Time is plotted along the abscissa and frequency along the ordinate. High-energy is indicated by the darker print levels.

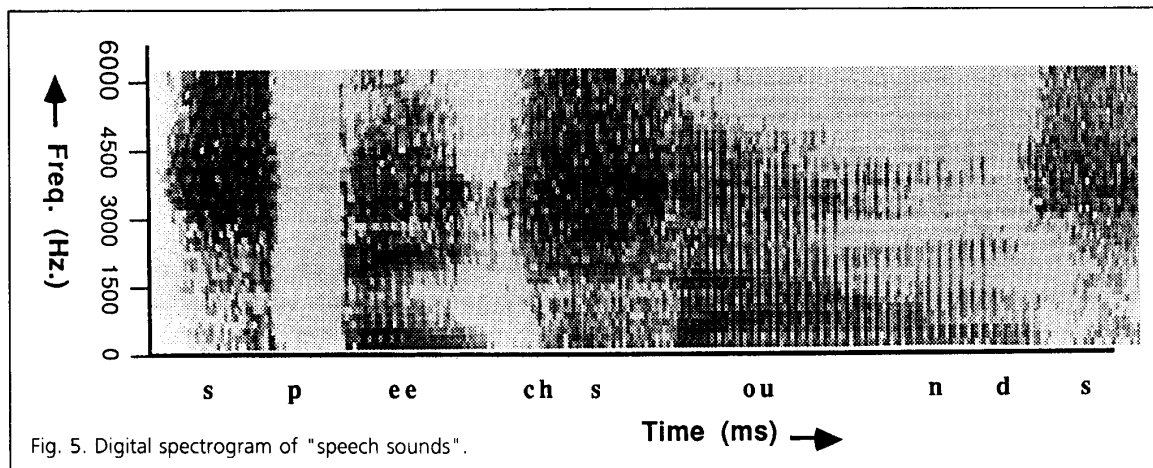
In DP DUR and CSR, it is assumed that some set of pre-stored ensembles of feature vectors is available. Each member is called a *prototype* and the set is indexed by  $k$ .

$$k^{\text{th}} \text{ Prototype} \equiv P_k \quad (3.2)$$

Many feature sets have been proposed and implemented, including many types of **spectral** features (log-energy estimates in several frequency bands) [20], [21], features based on a model of the human ear [22], [23], features based on LPC analysis [24] and features developed from phonetic knowledge

The  $l^{\text{th}}$  component of the  $i^{\text{th}}$  feature vector for the  $k^{\text{th}}$  prototype is therefore,  $P_k(i, l)$ . Similarly, the data for recognition are represented as the *candidate* feature vector ensemble,

$$\text{Candidate} \equiv C. \quad (3.3)$$



The  $j^{\text{th}}$  component of the  $i^{\text{th}}$  feature vector of the candidate will be  $C(i,j)$ . The problem for recognition is to compare each prototype against the candidate, select the one which is, in some sense, the closest match, with the intent that the closest match is appropriately associated with the spoken input.

There are many algorithms that are currently used for matching a candidate and prototype. Some of the more successful techniques include network-based models [26], and hidden Markov models (HMM's) applied at both the phoneme [27] and at the word level [28], [29], [30], [31], [32]. However, DP remains the most widely used algorithm for real-time recognition systems [33]. In addition, DP has been considered to be sufficiently mature so that a number of custom VLSI solutions have been proposed or developed [34], [35], [36], [37], [38], [39], [40]. The VLSI chip described in the last reference is currently available to interested parties from AT&T.

There are many varieties of speech-recognition algorithms. For smaller vocabulary systems, a set of one or more prototypes is stored for each utterance in the vocabulary. This structure has been used both for talker-trained and talker-independent systems as well as for DUR and CSR. When the recognition task for a large vocabulary (>1000 utterances), or even a talker-independent medium-sized vocabulary (100-999 utterances) is considered, the use of a large set of prestored word- or utterance-level prototypes is, at best, cumbersome. For these systems, parsing to the syllabic or phonetic level is reasonable [41]. In this paper, dynamic-programming techniques are described which apply to small and medium vocabulary DUR and CSR. Most of the current large-vocabulary systems use stochastic modeling ideas and hidden-Markov models. The Viterbi algorithm [42], [43], which will be briefly discussed in Section 7, is the most widely-used algorithm for recognition on stochastic models. Given a set of predetermined transition and output statistics and a string of output events, it finds the path through the model which was most likely to have generated that string of output events. In similar fashion to the dynamic-programming algorithm, the main computation is in the forward direction and, at the end of the forward computation, the probability for producing that output string by traversing the model along that path is obtained. Also, if the state sequence is desired, a back-tracking stage is required. Thus, the Viterbi algorithm is considered a "stochastic" version of dynamic programming [44]. The development of the hidden-Markov modeling ideas for recognition is a broad subject and the reader is advised to read [45] for a fuller treatment.

## DP CONSTRAINTS AS APPLIED TO DUR

Necessary concepts for the application of DP to speech are most easily introduced in relation to DUR. These ideas were first presented by Nagata, Kato and Chiba in 1962 [46], and, somewhat later, the two most-often-referenced papers on the subject appeared [47], [48]. Motivation for using the somewhat costly DP algorithm is indicated in Figure 6, in which both the X and the Y axes represent the independent variable *time*. Consider, for example, that the interval along the X-axis is one in which the candidate word **speech** has been spoken in a

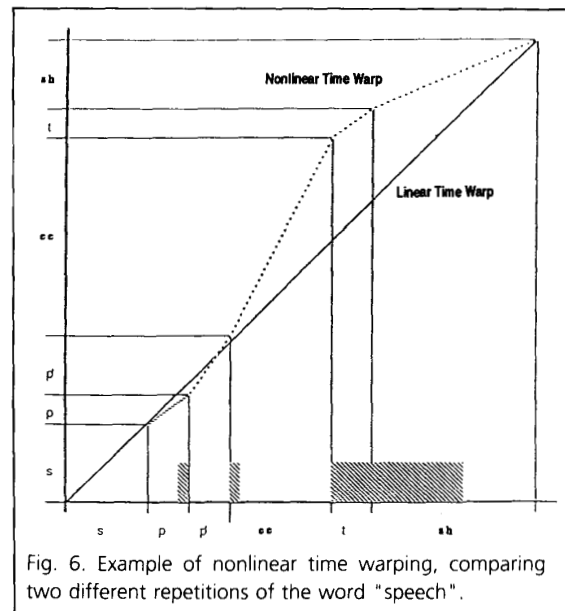


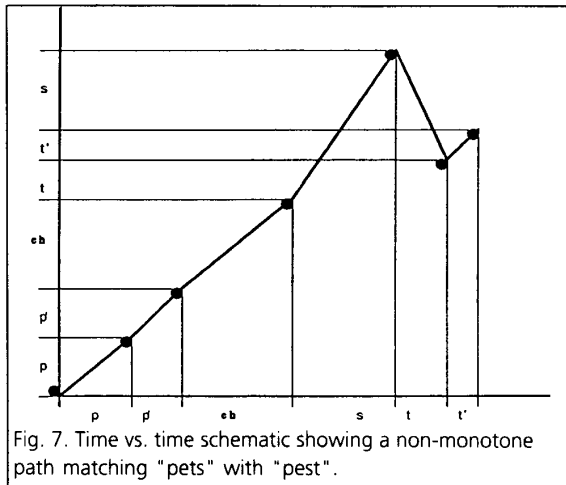
Fig. 6. Example of nonlinear time warping, comparing two different repetitions of the word "speech".

normal fashion; the Y-axis contains a time interval in which the prototype for the word **speech** has been recorded. It is drawn as if the /ee/ vowel in the prototype is somewhat longer than normal, and the /sh/ sound at the end of the word somewhat shorter. The fact that these phonemic differences in length occur in normal speech implies that a **nonlinear** alignment in time is essential when comparing a candidate to a stored prototype. The linear match shown would make severe errors by comparing different sounds in the time intervals cross-hatched at the bottom of the Figure.

Some important considerations must be addressed in order for the DP algorithm to serve as an algorithm for "optimizing" the time alignment between a candidate and a prototype.

These include:

- An imposition of the overall stretching or compression to be allowed. This, in some sense, is the major factor in the *global constraint*.
- An imposition (often determined directly by the *global constraint*) on an allowed set of local predecessors when computing the "optimal" cumulative values in the forward direction. This is called the *local constraint*. The *local constraint* also imposes the requirement for *monotonicity*, i.e., phonetic events will be considered in their natural order in time; for example, comparison of the words *pets* and *pest* by the non-monotonic warping path shown in Figure 7 will be disallowed. Finally, the "local constraint" also imposes some measure of continuity. This is done by limiting the scope of the predecessor search -- fewer potential predecessors implies fewer "skips" and thus a more continuous function.
- The imposition of a metric that may be computed to compare two sets of features -- one from the candidate and one from the prototype -- either context dependent or not. Often, this metric is the simple Euclidean ( $L_2$ ) or absolute value

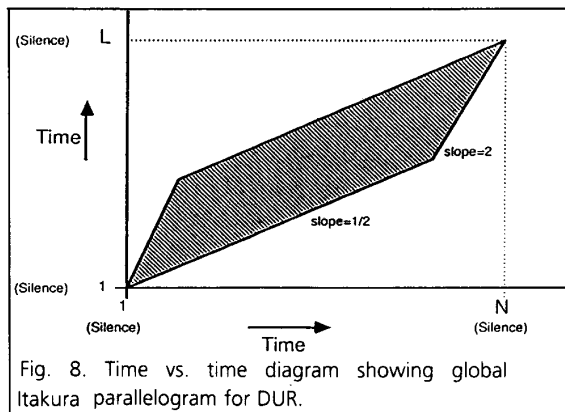


$(L_i)$  distance for a spectral feature vector. The number representing the quality of match, or a distortion measure between the  $j^{\text{th}}$  feature vector of the prototype (candidate) and the  $j^{\text{th}}$  feature vector of the candidate (prototype) for the  $k^{\text{th}}$  prototype will be defined.

$$d_k(i, j) \equiv \|C(i, *), P_k(j, *)\|. \quad (4.1)$$

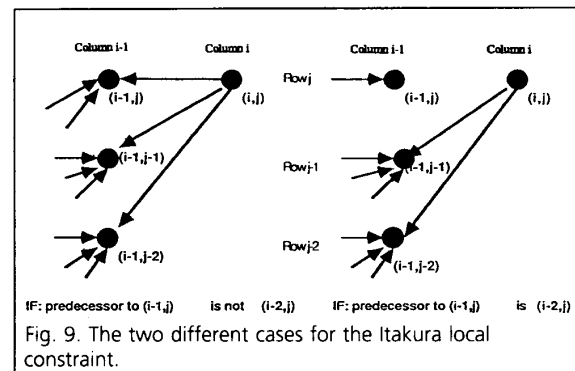
In this work  $d_k(i, j)$  shall be referred to as the local distortion. The  $*$  holds the place of the index on feature component which is the "variable of integration" for the measure. This metric is measure of distortion, rather than of similarity, in that higher values of the metric imply **less** similarity, i.e., **more** distortion. Numerous distortion measures have been proposed and this is still an active research area [49], [50], [51], [52], [53], [54]. The evaluation of these measures is beyond the scope of this work.

There are really two, fairly closely related paradigms for the constraints which apply both to DUR and CSR. The Itakura constraints [48] are easily developed from the imposition of the *global constraint* that a factor of two stretching and/or compression is the most allowed. If an additional assumption is made, such as having the utterance-detection algorithm guarantee that a single feature vector of silence both precedes and follows the utterance, then one can lock the match at either end as shown in Figure 8. The factor of two imposition



is also clear in the figure; extending lines from the origin  $((1, 1) -$  the match of the two initial silence feature sets) and back from the terminal point  $((N, L) -$  the match of the two final silence feature sets) creates the well-known parallelogram search space. One should note that if  $N$  is exactly twice (or half of)  $L$ , the allowed search space is simply the diagonal line. Finally, as one can see in Figure 8 (approximately), when the utterances being compared are about the same length  $N$ , only about  $N^2/3$  evaluations of the DP equation need to be done. This fact implies that for a vocabulary of size  $K$  with a wide variation in the length of the prototypes, that some utterances will be rejected on length alone and that the number of DP evaluations per candidate of length  $N$  will be significantly fewer than  $K \cdot N^2/3$ .

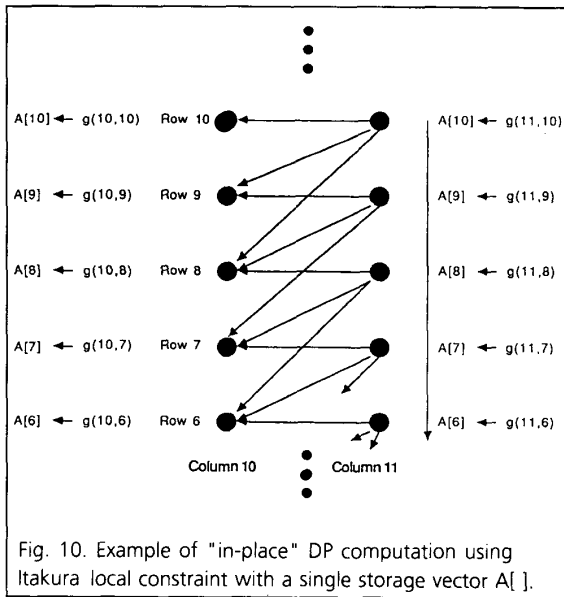
The Itakura local constraint is shown in Figure 9. The normal situation is shown on the left where three predecessor paths are allowed. As predecessors to point  $(i, j)$  all come from the previous column, horizontal skipping is disallowed by this constraint. The lowest path in the Figure indicates that a vertical feature vector may be skipped --  $(i, j)$  is optimally preceded by  $(i-1, j-2)$  -- and thus an overall slope of two could be achieved in the pathological limit. One may also see that if the optimal



predecessor to  $(i-1, j)$  were  $(i-2, j)$ , then if the horizontal path from  $(i-1, j)$  to  $(i, j)$  were allowed by the local constraint, a global horizontal path could result. Therefore, paths of global slope less than  $1/2$  are eliminated by the paradigm depicted on the right side of Figure 9, in which a horizontal path may not be extended by another horizontal path.

These ideas may be clearer in light of the need to satisfy two properties. *Monotonicity* implies that the predecessors in the forward DP computation should come only from points adjacent or below. *Continuity* requires that a "skip" of more than one is to be disallowed in any local DP calculation.

For ease in programming, the normal Itakura DP constraints are *asymmetric* in that, like the previous example problem, one is forced to match every feature set of the utterance on the horizontal axis. This is also beneficial for programming and storage in that predecessor results need only be retained for one full column of the DP global space, as shown in Figure 10. Say cumulative results for column 10 are stored in the vector  $G(j)$ , and the next step in the forward search is to compute column 11. None of the predecessors contained in  $G(j)$  are "above" the current point, i.e., predecessors to point  $(11, j)$  are the points  $(10, j)$ ,  $(10, j-1)$ , and  $(10, j-2)$ .



Thus, if computation is performed down the column, i.e.,  $j = \dots, 10, 9, 8, \dots$ , then the current accumulated result value may replace the past value in  $G(i)$  -- a genuine *in-place* DP algorithm, as indicated in Figure 10.

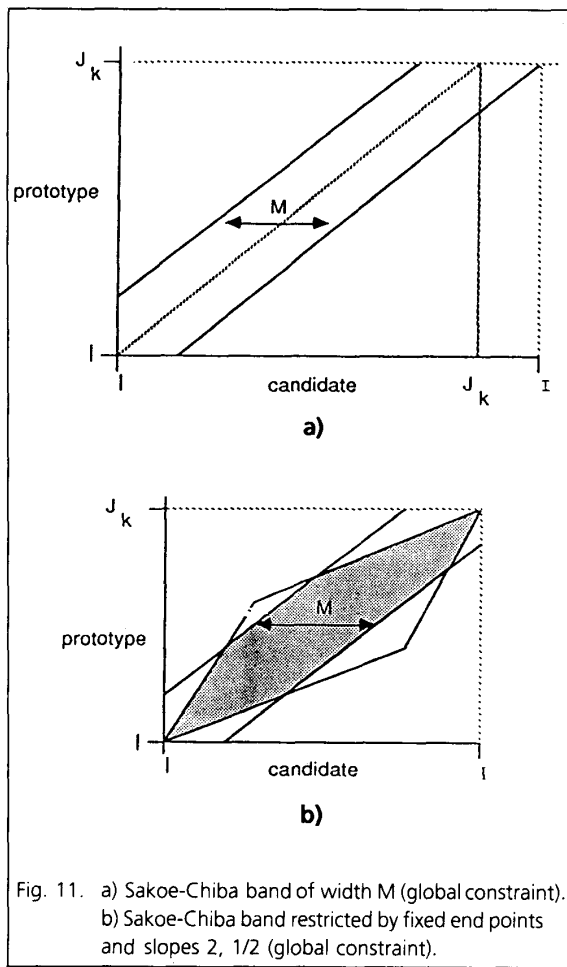
If  $g_k(i,j)$  is defined as the cumulative (optimal) error to match prototype  $k$  to the candidate from the origin (0,0) to point  $(i,j)$ , then the DP algorithm for the Itakura constraint, as shown in Figure 9, is given by,

$$g_k(i,j) = d_k(i,j) + \min \begin{cases} g_k(i-1,j) & A \text{ if } f \text{ predecessor not } A \\ g_k(i-1,j-1) & B \\ g_k(i-1,j-2) & C \end{cases} \quad (4.2)$$

The special constraint listed is necessary so that a "horizontal line" match is disallowed and the minimum slope is constrained to 1/2.

The Sakoe-Chiba family of constraints [49] generates either symmetric constraints -- both the utterance placed on the vertical axis and the one on the horizontal axis are treated the same -- or asymmetric constraints -- the horizontal axis is favored as in the case of the Itakura constraints. Typically, the global constraint is a band of width  $M$  rising at a 45° angle as shown in Figure 11a. For DUR, this band can be truncated as shown in Figure 11b (for the slope 2 or 1/2 constraint) to limit the DP computation space. The truncation is clearly dependent upon the local constraints selected.

A family of local constraints was introduced by Sakoe and Chiba. As an example, three local constraints are shown in Figure 12. In Figure 12a, three local predecessors are allowed, and no local constraint upon the slope is imposed. Thus, for this DP algorithm, long horizontal or vertical optimal matches are allowed, although the "band" global constraint will restrict these pathological cases. A slope constraint of 2 (or 1/2) is imposed in Figure 12b. An off-diagonal predecessor may be taken only through the upper and lower trajectories, thus



guaranteeing the slope constraint. Finally, it is easy to see that the slope constraint may be modified to 3 (or 1/3) as shown in Figure 12c, and that asymmetric combination versions may be adopted.

The algorithms shown in Figure 12 may be symmetric or asymmetric due to the weights assigned to the paths. A *symmetric constraint* weights advances along both axes equally while an *asymmetric constraint* weights advance along one axis only. Taking Figure 12b (three local trajectories,  $\pi_1, \pi_2, \pi_3$ ) as an example, one may write the DP algorithm as

$$\begin{aligned} f(\pi_1) &= g_k(i-2, j-1) + \nu_1 * d_k(i-1, j) + \nu_2 * d_k(i, j) \\ f(\pi_2) &= g_k(i-1, j-1) + \nu_3 * d_k(i, j) \\ f(\pi_3) &= g_k(i-1, j-2) + \nu_4 * d_k(i, j-1) + \nu_5 * d_k(i, j) \end{aligned} \quad (4.3)$$

where  $\nu_n, n \in [1,5]$  are weights dependent upon the symmetry and  $f(\pi_m)$  represents the cost of extension of a path to  $(i,j)$  through local trajectory  $\pi_m$ . The new cumulative score at point  $(i,j)$  would then be set to

$$g_k(i,j) = \min \begin{cases} f(\pi_1) \\ f(\pi_2) \\ f(\pi_3) \end{cases} \quad (4.4)$$



The weights for different symmetries for Equations 4.3 and 4.4 are listed in Table IV.

Constraint	$\nu_1$	$\nu_2$	$\nu_3$	$\nu_4$	$\nu_5$
Symmetric	2	1	2	2	1
Horiz. Asym.	1	1	1	0.5	0.5
Vert. Asym.	0.5	0.5	1	1	1

**Table IV.**  
Vertex Coefficients for Sakoe-Chiba Constraint of Figure 12b

The weights of Table IV arbitrarily correspond to a weight of one unit for each move counted. Thus, to compare paths of different lengths, i.e., compare the average cost for each step of a path, it is necessary to divide the accumulated result by the counted path length. For the symmetric algorithm, the counted path length will be the sum of the vertical and the horizontal span of a path. This implies that the counted path length must depend on the length of each prototype, and the division by the path length is usually performed. For, say, the horizontal asymmetric case, the path length is simply the horizontal span of the path, and, if the candidate were to be along the horizontal, one would always be dividing by the same number and the division would be unnecessary. This saving could be an important consideration for DUR systems.

The Sakoe-Chiba constraints are not selected for "in-place" calculation. For example, for the most-commonly used algorithm of Figure 12b, two predecessor columns must be kept and the computation must be performed bottom-to-top, i.e.,  $j = \dots, 5, 6, 7, 8, \dots$ . Thus three columns of local storage are the minimum required to perform the DP forward calculation. If  $M$ , the width of the band, is small, however, the above characteristic does not impose too great a hardship.

#### DETERMINISTIC DP AS APPLIED TO CSR

This section discusses the direct or deterministic application of DP to connected speech recognition. In CSR, a candidate consists of a **string** of words. It is, more specifically, some string of concatenated words from a limited vocabulary, uttered by a talker in a normal, connected fashion. CSR does not rely on any *a priori* knowledge of the input string, such as the number of words, or the beginning or endpoint of each. Early work on CSR centered on searching for ways to segment the string into discrete words, and then applying dynamic programming techniques developed for DUR [55]. For more recent CSR, which is the focus of this section, it is always assumed that the number of words in a string is unknown, the more general case. It is also assumed that the specific vocabulary is unknown, i.e., vocabulary-specific rules or algorithms are disallowed.

The words in the candidate are usually somewhat different from stored prototypes due to the allophonic effects at word boundaries; for example, the /z/ sound in "zero" really sounds like /z/ --voicing is on -- when preceded by "three", but appears

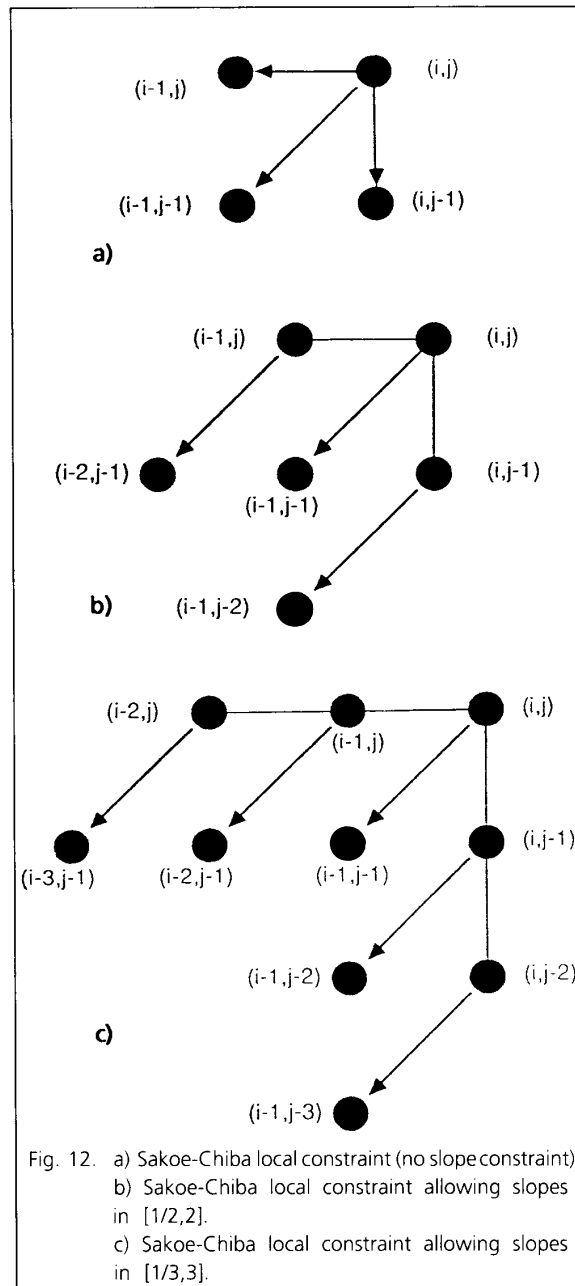


Fig. 12. a) Sakoe-Chiba local constraint (no slope constraint).  
b) Sakoe-Chiba local constraint allowing slopes in  $[1/2, 2]$ .  
c) Sakoe-Chiba local constraint allowing slopes in  $[1/3, 3]$ .

more like an /s/ when preceded by "six". In connected speech, there are instances in which adjacent words in the string share a sound, such as the /n/ between "seven" and "nine". There can also be small pauses or unique transitional events between certain words such as the /w/-like transition between the /oo/ and /ei/ sounds between "two" and "eight".

In many connected speech applications it is important that word decisions be made as the talker is speaking and while processing is taking place. This type of left-to-right recognition is called *early-decision* [56]. Early-decision algorithms enable the microphone to be on continuously, minimizing problems associated with sensitivity to pauses in the utterance. The

alternative to early-decision may be called *deferred decision*. Here, no recognition output is forthcoming until the talker pauses; soon thereafter, decisions for the whole captured string are given. Theoretically, as more information is available for the deferred-decision recognizer, performance should be better. To date, no experimental work has validated this hypothesis. It is clear, however, that early-decision recognition algorithms must be developed with more careful attention to real-time concerns.

In CSR, one attempts to match prototypes to subintervals of the candidate. In the earliest algorithms, a strict concatenation of the candidate data was enforced. A more realistic formulation involves a "looser" definition of concatenation such that small overlaps and/or gaps are allowed. Let the operator  $\oplus$  denote this loose concatenation. A CSR candidate string having  $Q$  words can then be represented as

$$\text{Candidate}(Q) = c_1 \oplus c_2 \oplus \cdots \oplus c_q \oplus \cdots \oplus c_Q \quad (5.1)$$

In DUR, for a given prototype  $k$ , all paths started at a fixed starting point in the DP space, and ended at a fixed termination point. As may be seen in Figure 13, in CSR an "optimal" path to some point in the DP space  $(i, j)$  may be derived from *every* allowed starting point for that path. Thus, there is a small, countable number of "optimal" paths to each  $(i, j)$ ; these are differentiated by using  $m$  as the index in the path's starting

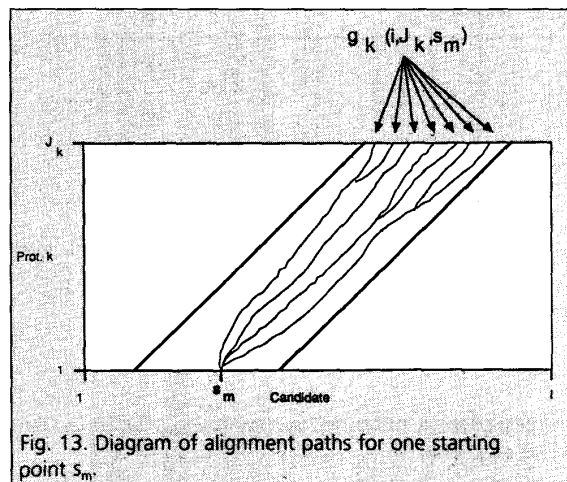


Fig. 13. Diagram of alignment paths for one starting point  $s_m$ .

point,  $s_m$ . One should note that  $s_m$  indicates a time index and that  $s_m \in [1, I]$  for a candidate of  $I$  samples. One may also see in Figure 13 that all paths terminate at the top of the DP space, and not at a single point. Therefore, it is possible that orders-of-magnitude more "score" data per prototype than in the DUR case, in which there is only one "score" per prototype, must be handled. In CSR, one can take the view that there are  $M$  "optimal" scores for each ending point, which implies  $I \cdot M$  "scores" for each prototype  $k$ .

The following are defined for CSR:

- $g_k(i, j, s_m) \equiv$  the cumulative score at point  $(i, j)$  for prototype  $k$  originating at (candidate) time sample  $s_m$ .
- $G_k(i, j, s_m) \equiv g_k(i, j, s_m) / (i - s_m + 1)$ , where the denominator is the horizontal path length.

The prototypes, always shown along the vertical axis in a graphic such as in Figure 13, have different overall lengths; in particular the length (the number of feature vectors) for the  $k^{\text{th}}$  prototype will be defined as  $J_k$ . As all paths end at the top in Figure 13, the values necessary for further consideration for CSR are  $g_k(i, j, s_m)$  or  $G_k(i, j, s_m)$ . Usually, the latter values are used because they allow a meaningful comparison to be made between paths of very different lengths and among different prototypes. However, for some vocabularies, one might save the expensive "division" operation and the former measure could suffice. In either case, these values are essentially three dimensional. They are a function of the index on prototype,  $k$ , path starting point  $(s_m, 1)$ , and path ending point  $(i, J_k)$ . If, somewhat parallel to the DUR case,  $M$  is defined as the number of allowed path starts for each path end, then

$$\text{Number of output values} = K \cdot I \cdot M \quad (5.2)$$

For example, with 100 prototypes ( $K=100$ ) and the typical case of allowing about 20 path starts for each path end ( $M=20$ ), then, for a candidate with 200 feature vectors ( $I=200$ ) (similar to a seven digit utterance spoken at a moderate rate), 400,000 values would result!

In the early days of CSR (1977-1985), the main reason for reduction of this data was the cost of the memory. This cost is less of a motivation today because of great advances in memory technology. However, one still needs an algorithm for extracting only a few items of information from this large volume of data.

Consider that, the connected speech recognizer is expected to determine:

- the **number of words** in the candidate,  $\hat{N}$ ,
- an **ending point**, referenced to the feature index, for each word  $\hat{i}_n$ ,  $\hat{i}_n \in [1, I]$ ,  $n \in [1, \hat{N}]$ , (Usually  $\hat{i}_N = I$ .)
- the **classification** of each word, referenced to a prototype number,  $\hat{k}_n$ ,  $\hat{k}_n \in [1, K]$ ,  $n \in [1, \hat{N}]$ .

For the values above, only 15 numbers specify the result for a seven-word input string. Sometimes it is useful to include a number for each word that is indicative of the error in a word match, so the connected speech recognizer must also produce the **word-match error** defined as  $\hat{e}_n$ ,  $n \in [1, \hat{N}]$ . In the notation above, the "hat" indicates final-decision information.

There are two schools of thought regarding how to reduce the resultant data from the forward stage of DP. As data are required for each **endpoint** of the paths, one may use a maximization process on one of the other two variables, i.e. the prototype index  $k$  or the starting point  $s_m$ , to compress the results by one or more orders of magnitude. This large reduction in data and memory requirement does incur the relatively small expense of doubling the size of the storage area needed for the remainder of the data so that the **argument** for the maximum value of the "missing" variable may be stored.

In what may be defined as **Fixed Memory (FM)** CSR DP, the following are performed:

$$(5.3) \quad G_{FM}(i, m) \equiv \min_{1 \leq k \leq K} g_k(i, J_k, s_m) / (i - s_m + 1) \quad (5.3)$$

$$\bar{k}(i, m) \equiv \operatorname{argmin}_{1 \leq k \leq K} g_k(i, J_k, s_m) / (i - s_m + 1) \quad (5.4)$$

Equation 5.4 uses the abbreviation *argmin* which operationally means that  $\bar{k}(i, m)$  is the particular value of  $k$  which minimizes  $g_k(i, J_k, s_m) / (i - s_m + 1)$ . One should note that the path-length normalized result, which is the one commonly used, is defined. In addition, the results are indexed on  $m$ , rather than on  $s_m$ , so that storage in a simple rectangular array is possible. If the longest candidate to be allowed is of length  $l_{max}$ , then two storage areas are needed, each of size  $l_{max} \cdot M$ . For the example above, only 4000 locations are required for each storage area. Furthermore, the size of this area is fixed for all vocabularies, i.e., it is not a function of  $K$ , the number of prototypes.

The memory size is not fixed for the other type of CSR DP defined as **Vocabulary-Dependent Memory (VDM)**. In VDM, at each point  $(i, J_k)$  for prototype  $k$ , the following are computed:

$$G_{VDM}(i, k) \equiv \min_{1 \leq m \leq M} g_k(i, J_k, s_m) / (i - s_m + 1) \quad (5.5)$$

$$\bar{s}(i, k) \equiv \operatorname{argmin}_{1 \leq m \leq M} g_k(i, J_k, s_m) / (i - s_m + 1) \quad (5.6)$$

Two storage areas are required, each of size  $l_{max} \cdot K$ . The dependence on  $K$  implies that more storage is needed as the number of prototypes grows. This is a disadvantage for VDM CSR, as, for example, for 100 prototypes in the above example, five times the storage area of FM CSR is needed. There is, however, a real advantage to VDM; the *principle of optimality* holds for the minimization process over the starting point index  $s_m$ . This is made clear using the example shown in Figure 14, where the simple Itakura local constraint has been used. "Optimal" paths for four different starting points,  $m = 1, 2, 3, 4$ , are shown emanating from the three allowed predecessors.

This idea may be derived from the unnormalized previous definition of Equation (5.5) and the DP forward iteration,

$$g_{VDM}(i, k) = \min_{1 \leq m \leq M} [d_k(i, J_k) + \min_{0 \leq j \leq 2} [g_k(i - 1, J_k - j, s_m)]] \quad (5.7)$$

As the local metric,  $d_k(i, J_k)$  is not a function of a minimized variable, and since the minimizations are independent,

$$g_{VDM}(i, k) = d_k(i, J_k) + \min_{0 \leq j \leq 2} [\min_{1 \leq m \leq M} [g_k(i - 1, J_k - j, s_m)]] \quad (5.8)$$

Thus,

$$\bar{g}_k(i, j) \equiv \min_{1 \leq m \leq M} [g_k(i, j, s_m)] \quad (5.9)$$

may be defined as well as,

$$\bar{s}_k(i, j) \equiv \operatorname{argmin}_{1 \leq m \leq M} [g_k(i, j, s_m)]$$

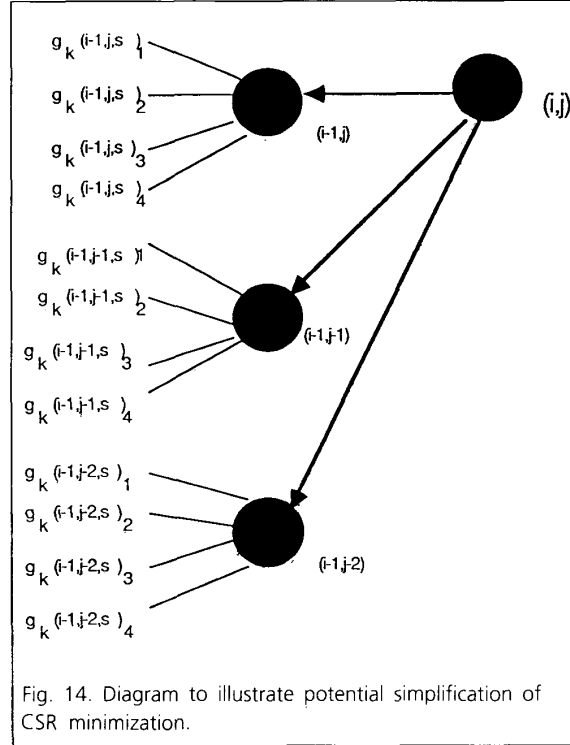


Fig. 14. Diagram to illustrate potential simplification of CSR minimization.

Initializing,

$$\bar{g}_k(i, 1) \equiv d_k(i, 1) \quad (5.10)$$

and

$$\bar{s}_k(i, 1) \equiv i,$$

the desired result is obtained by substituting  $\bar{g}$  for the last term in Equation (5.8).

$$\bar{g}_k(i, j) = d_k(i, J_k) + \min_{(0, 1) \leq n \leq 2} [\bar{g}_k(i - 1, j - n)] \quad (5.11)$$

$$\bar{s}_k(i, j) = \bar{s}_k(i - 1, j - \operatorname{argmin}_{(0, 1) \leq n \leq 2} [\bar{g}_k(i - 1, j - n)])$$

The notation (0,1) implies the normal Itakura local constraint. Essentially, the minimization over path starting point  $s_m$  is done locally and implicitly.

As the forward DP equation is computed only once per point per prototype, there is a computational advantage to using VDM CSR over FM CSR, which offsets the **storage** advantage of the latter. In FM CSR, the DP equation is computed several times for each point  $(i, j)$  for prototype  $k$ . As a significant portion of this computation involves obtaining the metric  $d_k(i, j)$ , this part of the work may be pre-computed and stored. This reduces the computational advantage of VDM CSR somewhat, at the expense of increasing the required local memory.

### The NEC CSR Algorithm

The earliest DP CSR algorithm, that of Sakoe and Chiba of NEC, was developed for a real-world recognizer [57]. Because memory was an important consideration at that time, it was the FM CSR type. Sakoe, Chiba and Kato [58] concluded that the asymmetric version of the algorithm pictured in Figure 12b was "best" for connected speech. They used this with a band of 8 (18ms per feature vector) on each side, which implies  $M = 17$ . In their system, this allowed about 144ms of maximum local warping for any path. Their distance metric was a simple sum of the absolute values, and much of the computing was done in parallel using a bit-slice system for the DP equation. After obtaining the resulting fields of Equations 5.3 and 5.4 in a step they called *word-level matching*, they reapplied DP to the fields, searching backward to find the string and boundary information in a second step which they called *phrase-level matching*.

*Phrase-level matching*, in which  $N$  is considered unknown, is conceptually easier to understand if one considers the index for the **last** word in the string as 1 and for the first word as  $N$ . The initialization for the DP is then,

$$\hat{i}_1 = 1, \quad (5.12)$$

and the recursions are

$$e(\hat{i}_n) = \min_{1 \leq m \leq M} [G_{FM}(\hat{i}_n, m)] \quad (5.13)$$

$$m_n = \operatorname{argmin}_{1 \leq m \leq M} [G_{FM}(\hat{i}_n, m)]$$

$$\hat{k}_n = \hat{k}(\hat{i}_n, m_n)$$

$$\hat{i}_{n+1} = \hat{i}_n - J_{\hat{k}_n} + m_n - (M + 1)/2$$

where  $M$  is considered odd, and the indices are defined as shown in Figure 15. The recursion continues until, for word  $N+1$ ,  $\hat{i}_{N+1} \leq 1$ . This specifies  $N$ .

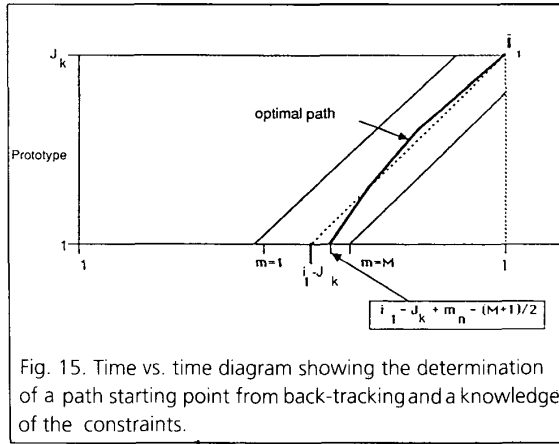


Fig. 15. Time vs. time diagram showing the determination of a path starting point from back-tracking and a knowledge of the constraints.

It is easy to see from Equation 5.13 that the second stage of DP requires very little computation. Therefore, at little expense, a third DP stage is sometimes added for better performance. This stage is really a DUR stage in which prototypes are developed by concatenating word prototypes into full-string prototypes. Typically, this is done for from two to ten optimal

or suboptimal paths as determined by small changes to the phrase-level DP algorithm. The string selected is that of the best DUR match.

The NEC algorithm was obviously developed for deferred decision. It used a spectral feature vector of size 16, and, to the authors' knowledge, used combinatorial logic for addition, shifting, and minimization, as shown in Figure 16, to compute

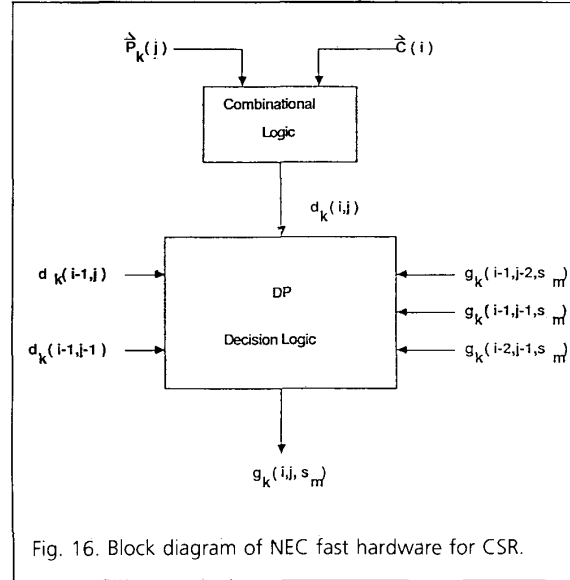


Fig. 16. Block diagram of NEC fast hardware for CSR.

a DP output in under 500ns. For this to be done in real-time, (i.e., a new feature vector handled in 18ms) as is required for an early-decision version,  $M \cdot J_{AVG}$  points must be computed for each new feature vector. Using an average prototype length  $J_{AVG} = 30$  (480ms) and  $M = 17$ , as above, implies the maximum number of prototypes to be,

$$K_{MAX} = \frac{18,000 \mu s / \text{vector}}{17 \cdot 30 \text{ vectors/prototype} \cdot 0.5 \mu \text{sec}} = 70 \text{ prototypes.}$$

A new second stage would also be necessary. However, the geometry of this algorithm allows it to be modified to run as an early-decision algorithm, in spite of a significant increase in complexity and memory requirements: In this case, no third DP stage is possible.

### The Level-Building Algorithm

Level-building is a fixed-memory algorithm which, in the deferred-decision case, attempts to streamline the NEC algorithm. It was originally formulated for the situation where the number of words was known *a priori*, but was soon extended so that  $N$  could be unknown [59], [60]. This DP algorithm is called level-building because it can be thought of as horizontally partitioning the ordinate (in DP space) into a series of  $N_L$  steps or levels.  $N_L$  may (or may not) be the actual number of words  $N$ , but is usually close. Level-building assumes

that the candidate string is partitioned into word chunks. Thus, if no grammatical restrictions are imposed, the algorithm must compute

$$L_{MAX} \equiv \lceil J/J_{MIN} \rceil \quad (5.14)$$

pieces, where  $J_{MIN}$  is the number of feature vectors in the prototype of minimum length, appropriately reduced to some minimum allowed match length, and  $\lceil \cdot \rceil$  implies the integer ceiling. For example, for the factor-of-two Itakura constraints,  $J_{MIN}$  would equal 1/2 the length of the smallest prototype and  $J$  would be the length of the candidate.

When all the prototypes are of the same length,  $J$ , one may get a picture of level-building from Figure 17. One should note

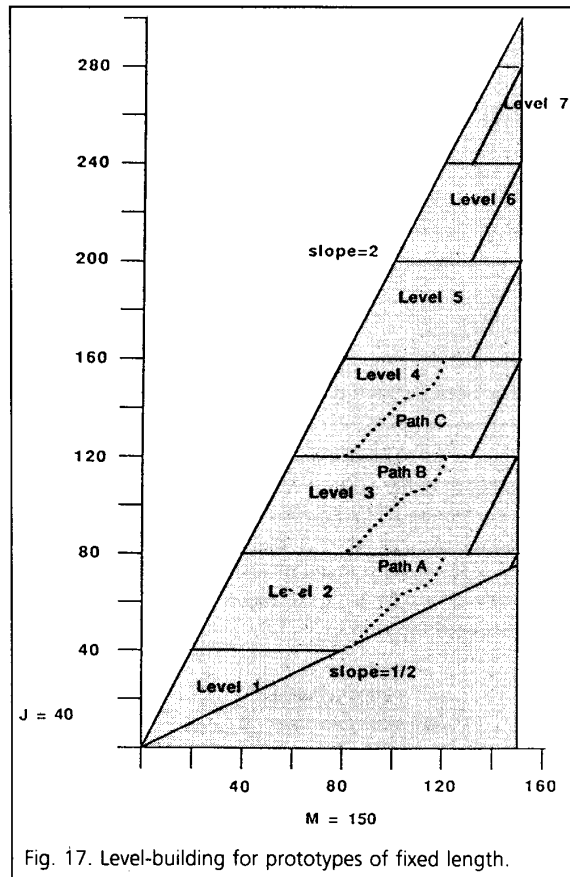


Fig. 17. Level-building for prototypes of fixed length.

the similarity to Itakura DUR. The concept is that, at each level, only certain paths need to be evaluated. For example, the first word in the string is forced to start at  $i=1$  and thus may end anywhere in the range  $J/2 \leq i \leq 2J$ . (In level-building, the Itakura constraint has normally been imposed.) The second level has paths which start in the latter range and may end at the top of the second level in  $J \leq i \leq 4J$ . This process continues for  $L_{MAX} = \lceil 2J/J \rceil$  levels, for, in general, starting points in the range defined as a function of level,  $l$  as,  $B(l) \leq i \leq E(l)$ . Here,  $B(l)$  is defined as the time index of the beginning of the range for level  $l$ , and  $E(l)$  is defined as the time index of the end of the range for level  $l$ . One should note  $B(0) = E(0) = 1$  and  $G^0(1) = 0$ , where  $G^l$  is defined below as the cumulative error.

Myers and Rabiner [60] noticed that the data needed for

storage at the end of a DP computation for a level were three one-dimensional arrays, the error  $G^l(i)$ , the path beginning  $s^l(i)$ , and the prototype number  $k^l(i)$ , where  $l$  is the index on level. Their main computation, for some level  $l$ , is initialized for each prototype  $k$  by

$$\begin{aligned} g^l(i, 1, k) &= G^{l-1}(i) & B(l-1) \leq i \leq E(l-1) \\ s^l(i, 1, k) &= i & B(l-1) \leq i \leq E(l-1) \end{aligned} \quad (5.15)$$

The DP recursion is

$$\begin{aligned} j^l &\equiv \underset{\text{Itakura Set } n(i, j-2, j)}{\operatorname{argmin}} g^l(i-1, n, k) \\ g^l(i, j, k) &= d_k(i, j) + g^l(i-1, j^l, k) \\ s^l(i, j, k) &= s^l(i-1, j^l, k) \end{aligned} \quad (5.16)$$

Note that both a cumulative error and a starting point are stored at each computation. Then, at the top of each level, the following minimization over the prototype index  $k$  is performed:

$$\begin{aligned} G^l(i) &= \min_{1 \leq k \leq K} [g^l(i, J_k, k)] & B(l) \leq i \leq E(l) \\ k^l(i) &= \underset{1 \leq k \leq K}{\operatorname{argmin}} [g^l(i, J_k, k)] & B(l) \leq i \leq E(l) \\ s^l(i) &= s^l(i, J_{k^l(i)}, k^l(i)) & B(l) \leq i \leq E(l) \end{aligned} \quad (5.17)$$

The efficiency of this part of the algorithm led to a saving of about an order of magnitude over the NEC algorithm. The amount of memory may be seen to be upper-bounded by  $3 \cdot l \cdot L_{MAX}$ . The level-building algorithm is not the most efficient, however. As may be seen in Figure 17, a DP path may be computed more than once; aside from an initial value, paths A, B, and C require precisely the same computations for each prototype  $k$ . However, for reasonable string and vocabulary sizes, the amount of **storage** required by the level-building algorithm is the smallest of the algorithms to be discussed.

Level building is slightly more complicated (and more difficult to picture) when the lengths of the prototypes are different. This is illustrated in Figure 18, where, even for two different

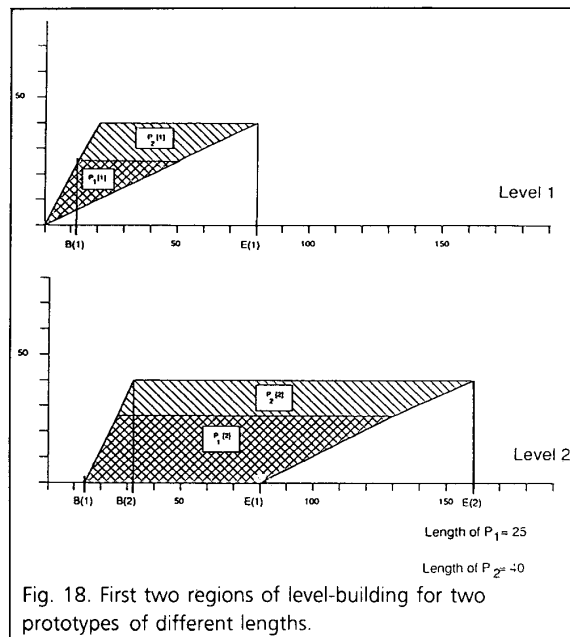


Fig. 18. First two regions of level-building for two prototypes of different lengths.

prototypes of length 25 and 40 respectively, the disparity in computation region is clear. The allowed starting point range for level 2 is a composite shown as  $B(1) \leq i \leq E(1)$ , which is different from the simple starting lines shown in Figure 17. It was shown that for an average prototype length of  $J$ , the average number of DP computations per level per prototype is  $J \cdot I/3$ , and thus the expected number of DP computations is  $L_{MAX} \cdot K \cdot J \cdot I/3$ .

The backtracking from the data of Equations (5.17) is about the same as in the NEC case; it requires only a few computations. Recently, level-building was extended so that early-decision is now feasible [61], [62]. Here, the algorithm has been modified to map to an equivalent network-based approach which permits early-decision or one-pass recognition.

#### The Bridle et al. algorithm

This algorithm is VDM CSR and typically applies the Itakura local constraint [63], [64]. As in the previous cases, nearly all the computation is performed in the first stage. The algorithm has essentially been derived in Equations (5.5) - (5.10). This algorithm performs  $J_k \cdot I$  DP computations which is the smallest over the cited algorithms, given no further global constraints. As expected in a VDM system, memory requirements are slightly larger than in the other cases, although as few as  $3 \cdot I$  locations could be used if the backtracker were only to use the data for the minimum over prototype and starting point; i.e., store

$$G(i) = \min_{1 \leq k \leq K} G_{VDM}(i, k) \quad (5.18)$$

$$k(i) = \underset{1 \leq k \leq K}{\operatorname{argmin}} G_{VDM}(i, k)$$

$$s(i) = \bar{s}(i, k(i)).$$

In the previous algorithms, the cumulative score at the start point,  $g_k(i, 1, i)$ , was initialized to  $d_k(i, 1)$ . Here links are introduced between the results already computed and new feature vectors, i.e.,

$$g_k(i, 1) = g(i) + d_k(i, 1) \quad (5.19)$$

$$\bar{k}(i, 1) = k(i)$$

$$\bar{s}_k(i, 1) = s(i).$$

where  $g(1) \equiv 0$ ,  $k(1) \equiv (\text{say } -1)$ , and  $s(1) \equiv 1$ . The best match is thus obtained for the lowest score at the end of the candidate input string, where the back-pointers indicate the prototypes chosen. The simplicity of this algorithm makes it easy to program. The algorithm's performance is independent of any *a priori* knowledge of string length, and it can be extended to allow early-decision recognition by simply emitting a "best score" at appropriate times, although the determination of these times may not be trivial.

A variant of Bridle's VDM CSR has been used at Brown University for several years [65]. The differences in the forward DP are as follows:

- The local constraint is a modified Itakura formulation; prototype "shrinkage" is allowed to be no more than 50% by disallowing the Itakura "skip" path to be taken twice in a row.
- For each feature vector in the candidate,  $G_k(i)$  and  $s_k(i)$  are saved for the decision-making phase.

- An **expert system** determines the time for a word-recognition decision and inspects the above data of the recent past. Potential overlaps (in time) due to coarticulation are understood, as are gaps due to short pauses.

The result is an early-decision system which uses a minimal number of forward DP computations and a relatively small amount of memory.

#### Computational Efficiency of CSR DP Algorithms

In Table V the computational requirements for the three DP algorithms are summarized. These numbers are consistent with those tabulated by Ney [66]. To review:  $I$  is the candidate length,  $K$  the number of prototypes,  $J_k$  the length of the  $k^{\text{th}}$  prototype,  $M$  is the band width for the NEC DP algorithm, and  $L_{MAX}$  is the number of levels in the level-building algorithm.

DP Method	Memory Requirements		Computation	
	Global	Local	Metrics	DP Searches
Sakoe Chiba	$2IM$	$3J_{max}$	$A. I \sum_k J_k$	$I M \sum_k J_k$
Level Building	$3L_{MAX} I$	$2J_{max}$	$A. I \sum_k J_k$	$L_{MAX} I/3 \sum_k J_k$
Bridle et al	$3I \text{ or } 2KI$	$2J_{max}$	$B. I \sum_k J_k$	$I \sum_k J_k$

A. Assumes metrics are computed *a priori*.  
B. No need for *a priori* storage.

**Table V.**  
**Comparison of different DP algorithms.**

While Table V indicates the number of metrics and searches to be computed, it should be understood that there is some underlying complexity for each of these computations. In particular, the metric computation generally involves multiply/accumulate operations, while DP searches generally are slowed by address computations. Therefore, efficient address generation and fast multiplication/accumulation are both important for implementing CSR DP efficiently. In current VLSI microprocessors, these two properties usually do not coexist. Digital-signal-processing microprocessors typically perform multiplication/accumulation at the maximum clock speed, but are substantially slowed when complex address computations and memory fetches are required. General-purpose microprocessors generally have a multiplicity of complex addressing modes and can store and retrieve from memory in single instructions although multiplication/accumulation may take many clock cycles. It does appear that the new RISC processors, such as the Motorola 88000, offer some very attractive trade-offs for DP implementation. In summary, one is forewarned that fast implementation of DP is affected equally by the non-arithmetic path extension and the computation of the metrics.

#### DP FOR CSR USING SYNCHRONOUS PROCESSING

*Asynchronous processing* refers to analyzing the input speech signal so that each feature vector represents a fixed-time interval. *Synchronous processing* uses variable-time intervals to analyze the input signal, producing feature vectors that are

synchronized to speech events but nonlinear in time. Processing of speech is called *event synchronous* if all feature vectors are based on time data which is representative of only one speech event [67]. For example, the 20-40ms windows used asynchronously often mix the burst of a stop (e.g. /p/, /t/, /k/) and the following vowel, or they include the end of a fricative (e.g. /s/, /sh/) with the prevoicing of a voiced stop (e.g. /b/, /d/, /g/). For voiced events, by extracting the pitch -- the rate at which the vocal folds open -- one obtains a spectrum driven by an impulse-like input rather than a periodic input, and thus harmonics of the pitch period are not in evidence in the resultant spectrum. It is a popular tenet that if talker-independent event-synchronous processing could be achieved with virtually no "parsing" errors, then recognition performance based on the synchronous processing would be better than that for the asynchronous case.

When the feature-vector index is not linearly related to time, the constraints set-up for the DP algorithms already given no longer apply. This implies that either

- the candidate and prototype data must be linearized in some fashion prior to using the conventional DP algorithms described in the previous section, or
- the synchronous data could be used directly in a DP algorithm with some new constraints which account for the nonlinearity.

These ideas will be discussed briefly in the next two subsections.

### Linearizing Synchronous Data

The strategy here is to pre-warp the feature vectors prior to the DP algorithm. The asynchronous DP algorithm would then assume that all features were to be treated equally (with respect to time). Two principal methods of data compression (and expansion) should be mentioned, linear time-scale warping and trace segmentation [68], [69], [70], [71]. The scope of the problem is made clear using some numbers for speech sampled at 12kHz. Each feature vector can represent an advance ranging from 35 to more than 180 time samples, corresponding to pitch frequencies of 342 to 66 Hz respectively.

The first method, *linear time-scale warping*, is a simple method in which a feature vector is selected (or interpolated) at some pre-selected interval (13.3 ms has been used for the 12kHz example). The obvious problem with this method is that the feature vectors which are not selected, and are discarded, might be very important. One should note that the selection of the feature vector nearest (in time) to the next, equispaced, time target, is to be preferred over interpolation so that the mixing of different events does not occur.

*Trace segmentation* is a compression/interpolation algorithm which employs some "measure" to compute the trace of a function in *L-space*. The simplest mechanism is to use the Euclidean or absolute-value distance between adjacent feature vectors in *L-space*. The sum of these distances for some input utterance represents the length of a "path" or "trajectory" in

*L-space*. The next step is to divide up the trajectory into some *M* equal parts and either select that feature vector closest to each equispaced point along the trajectory or use the exact point along the trajectory as a feature vector in the *M*-long result. This is illustrated by Figure 19 for an utterance composed of ten, two-dimensional vectors.

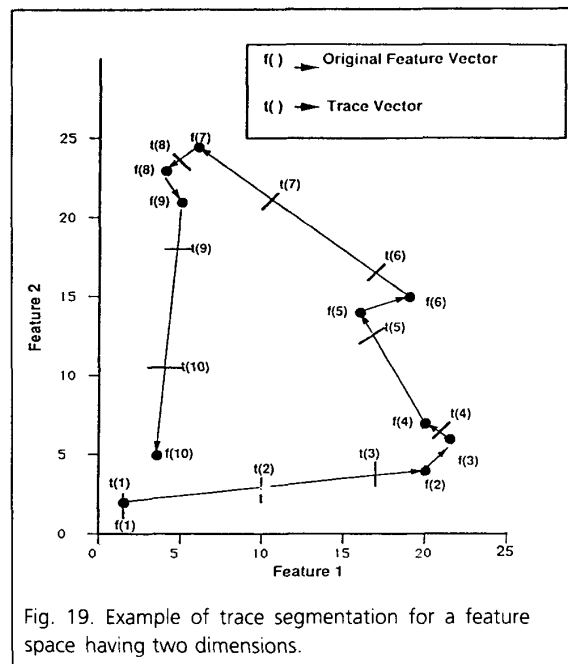


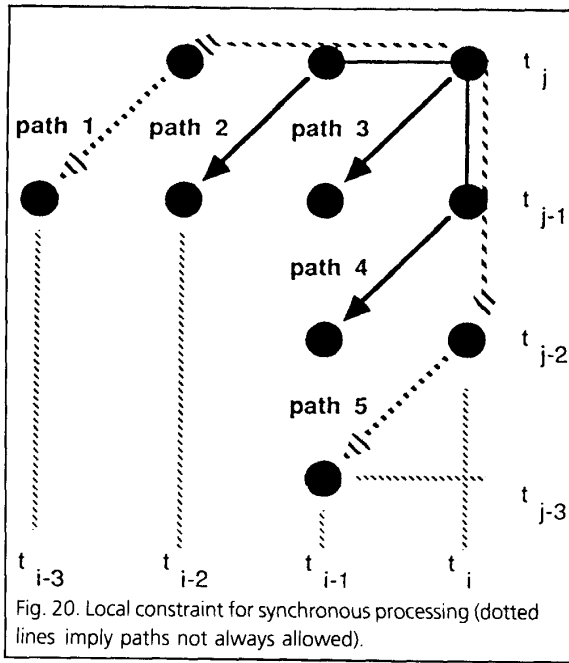
Fig. 19. Example of trace segmentation for a feature space having two dimensions.

One should note that a "measure" between two adjacent spectral feature vectors is simply a *spectral change* measure. Thus, the trace segmentation method puts a strong emphasis on transitional feature vectors and reduces the contribution from long, steady-state events. Therefore, this method is attractive to overcome some of the undue importance placed on long, steady-state events by conventional DP. However, one of the disadvantages of this algorithm, in the strict sense, is that it cannot be implemented in an early-decision fashion.

### A DP Algorithm for a Nonlinear Time Index

A DP algorithm which may be applied to synchronous data has been described in [14]. Normal DP may not be able to come close to a solution; take the worst-case scenario when a talker, on two different occasions, might utter the same voiced word at two significantly different pitch frequencies, e.g. 100 and 200 Hz. The number of feature vectors in the second utterance would be twice that of the first, an unlikely situation for normal speech but possible in stressed situations [72]. Although this may seem unrealistic, suppose the talker also said the word more slowly, while at a higher pitch. The longer warping paths cannot be accommodated by the previous local constraints.

However, another local constraint geometry proposed by Sakoe and Chiba [48] is shown in Figure 20. Two predecessor paths with longer "jumps" are augmented to the previous



Sakoe-Chiba constraint. In [14], experiments were performed which showed that this new constraint performed better for synchronously processed speech and that the new paths were selected between 5 and 15 percent of the time. However, while the additional paths allow skipping, when needed to align for differences in pitch etc., a blind application of the constraint also allows the potential skipping of an important event. For example, a short, important burst could be completely ignored. Thus the constraint should be made somewhat more intelligent. The following variables are now defined:

$n_c(i) \equiv$  time samples represented by a candidate feature vector  $i$ .

$n_{pk}(j) \equiv$  time samples represented by feature vector  $j$  of prototype  $k$ .

$$N_{pk} \equiv \sum_{j=0}^{j_k-1} n_{pk}(j)$$

$$N_c(i) \equiv \sum_{l=0}^i n_c(l)$$

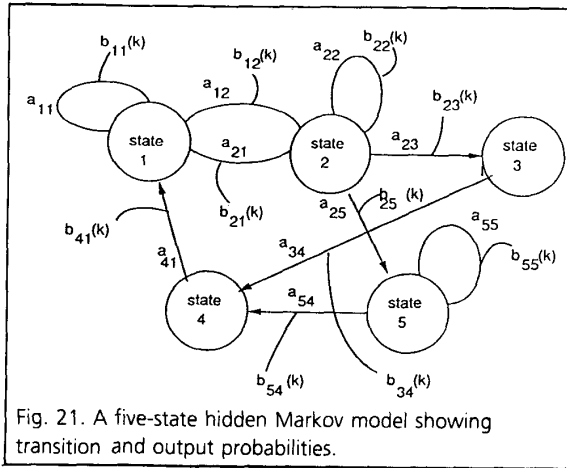
For example, if prototype  $k$  consisted of seven feature vectors having  $n_{pk}(0)$  through  $n_{pk}(6)$  equal to [160, 63, 67, 69, 67, 64, 160], then  $N_{pk} = 650$ . Suppose a candidate with  $n_c(0)$ ,  $n_c(1)$  and  $n_c(2)$  all equal to 40 was being compared to a prototype with  $n_{pk}(0)$ ,  $n_{pk}(1)$  and  $n_{pk}(2)$  all equal to 60. Would it be appropriate to choose *path 5* in the local constraint shown in Figure 20? This would account for a movement of 180 points in the prototype direction and only 40 points in the direction of the candidate. Obviously not, so the following should be added to the constraint:

- If  $n_{pk}(j) + n_{pk}(j-1) + n_{pk}(j-2) \leq (2 * n_c(i))$   
allow *path 1*.
- If  $n_c(i) + n_c(i-1) + n_c(i-2) \leq (2 * n_{pk}(j))$   
allow *path 5*.

Note that either *path 1* or *path 5* is allowed and they are never allowed simultaneously. One might note that if this constraint system were to be allowed for asynchronous processing, where, for example, the  $n_c(i)$ 's and  $n_{pk}(j)$ 's were equal to 160, neither *path 1* nor *path 5* would be selected, and the local constraint would simplify to the normal Sakoe-Chiba. Hence this constraint system could be construed as a unified approach for both synchronous and asynchronous processing. A complete treatment of this method, including some important material relative to normalization, is given in [14].

## THE VITERBI ALGORITHM

In addition to direct use of DP, speech researchers are using DP as a general tool with neural networks [73]. However, in general today, many CSR systems are based on hidden Markov models and do not use the deterministic forms of dynamic programming described in the previous sections [74], [75], [76], [77], [78], [79], [80], [81], [82], [83], [84], [85], [86], [87], [27], [31], [32]. However, nearly all the systems use one form or another of the Viterbi algorithm [42], [43] to compute the best path through the stochastic state model. In this section, the basics for a simple hidden Markov model will be presented. For further detail one is advised to refer to some of the specifics in the papers just cited.



Let the Markov model, shown in Figure 21, be used as an example. Here, the number of states,  $N$  is five. Given some insight into the speech system for which this is to be a model, it is usually the case that not all connections between the states are allowed. It is assumed that, given some set of parameters, the model can generate a string of labelled or quantified events (or feature vectors) in some sequential fashion. As before, let  $i$  be the index on feature vector in the input speech,  $i = 1, 2, 3, \dots, I$ .

The mechanism for generating each element  $i$  of this string of events from the model is the transversal of a link between states. Thus, a suitable set of constant parameters which characterize the Markov system are:

- $a_{mn}$  -- the probability that, at event  $i$ , there will be a state transition from state  $m$  to state  $n$ .



- $b_{mn}(k)$  -- the probability that for a state transition from  $m$  to  $n$  the output  $k$  will be produced.

- $s_m$  -- the probability of being in state  $m$  at the start.

Note that for the parameters to be true probabilities,

$$\sum_{n=1}^N a_{mn} = 1 \quad \text{for each } m \in [1, N] \quad (7.1)$$

$$\sum_{k=1}^K b_{mn}(k) = 1 \quad \text{for all } n, m \in [1, N]. \quad (7.2)$$

$$\sum_{m=1}^N s_m = 1. \quad (7.3)$$

where  $K$  is the number of possible outputs. In the above development, there is a discrete probability distribution on output for each link or transition. In some formulations, this distribution is forced to depend only on the state from which a transition occurs. In this case, only  $N$  output distributions are needed and  $b$  is no longer a function of  $n$ .

In speech, a typical scenario is to consider the set of outputs to represent different sounds; e.g., have  $K$  of dimension 40 to 60 to have a reasonable mapping to phoneme classes. Then, one might develop a state model such that taking certain paths through the model would indicate a specific word in the vocabulary to be recognized. One could use a single large model into which all the words are embedded, or several disjoint models, say one for each word. It really makes little difference since high probability for a path through some set of states would indicate high probability for a particular member of the vocabulary to have been uttered.

For a speech recognizer using normal asynchronous processing, the input signal for the Markov model would be (for the simple, discrete case considered here) a phoneme-class label,  $q_i$ , where  $q_i \in [1, K]$ , for each feature vector. The question then to be addressed is "What is the most likely path through the states to have produced the given string of input  $q_i$ ,  $i \in [1, I]$ ?" The fact that the generating state process is unknown, or is *hidden* makes this a *hidden Markov model*. The Viterbi algorithm, which is a stochastic version of dynamic programming, is a method for finding this most-likely state sequence.

Consider the probability of having produced the string  $q_i$  for  $i \in [1, I]$ , by means of a particular path through a given hidden Markov model, as a measure. If one wants to determine the path with the highest probability for producing string  $q_i$ , one can develop a measure  $C$ . Locally, this measure will be a function of the current state  $m$  as well as the current "time"  $i$ . Thus,  $C_m(i)$  will be defined as the *maximum* probability for a state sequence, ending in state  $m$ , to generate the first  $i$  observations of  $q_i$ . Clearly,

$$C_m(0) = s_m \quad m \in [1, N] \quad (7.4)$$

For  $i=1$ , the joint probability for each state may be computed by multiplying each initial state probability by the probability of the appropriate state transition and by the probability that out-

put  $q_1$  was produced by that transition. The maximum over all allowed transitions into state  $n$  at time 1 would be the "optimal"  $C_n(1)$  for each state  $n$ , i.e.,

$$C_n(1) = \max_{1 \leq m \leq N} [C_m(0) \cdot a_{mn} \cdot b_{mn}(q_1)] \quad (7.5)$$

It should now be evident that this process generalizes to

$$C_n(i) = \max_{1 \leq m \leq N} [C_m(i-1) \cdot a_{mn} \cdot b_{mn}(q_i)] \quad (7.6)$$

and that for potential backtracking one should keep,

$$s_n(i) = \operatorname{argmax}_{1 \leq m \leq N} [C_m(i-1) \cdot a_{mn} \cdot b_{mn}(q_i)] \quad (7.7)$$

A result may be computed by analyzing the values of  $C_n(i)$ , perhaps for a maximum, and backtracking using  $s_n(i)$  to determine the "optimal" state sequence.

To this point, the Viterbi algorithm looks like the deterministic algorithms except it uses multiplications rather than additions in its recursion. Not only are multiplications (for fixed-point arithmetic) more expensive computationally, but, even for relatively small values of  $i$ , the products would soon underflow--disastrously! Therefore, the logarithm is usually taken for the recursion. As a logarithmic transformation is monotonic, this does not affect the maximization computation. We, therefore, define

$$D_n(i) \equiv \log C_n(i) \quad (7.8)$$

and the recursion becomes,

$$D_n(i) = \max_{1 \leq m \leq N} [D_m(i-1) + \log[a_{mn}] + \log[b_{mn}(q_i)]] \quad (7.9)$$

$$s_n(i) = \operatorname{argmax}_{1 \leq m \leq N} [D_m(i-1) + \log[a_{mn}] + \log[b_{mn}(q_i)]] \quad (7.10)$$

It is clear that  $\log[a_{mn}]$  and  $\log[b_{mn}(k)]$  can be stored in lookup tables. Furthermore, their values will usually lie between -1 and -20, so they can be suitably quantized and used with fixed-point arithmetic. Integer overflow is easily managed. One can conclude that recognition using hidden Markov models can be quite efficient due to DP in the form of the Viterbi algorithm.

## CONCLUSION

Dynamic programming, both in its deterministic and stochastic forms, is the most widely used algorithm in speech recognition. This is because the speech recognition problem is easily formulated as one in which the best sequential path is desired. DP is a very simple idea which has profound impact in many areas; in fact, any optimization problem for which one can write a recursion lends itself to solution by DP. By mastering this simple idea, one gains the power to find optimal, i.e., intelligent, solutions to many problems.

## REFERENCES

- [1] Eric V. Denardo, *Dynamic Programming*, Prentice-Hall, Englewood Cliffs, NJ, 1982.
- [2] P. Masse, *Les Reserves et la Regulation de l'avenir la vie Economique*, Hermann, Paris, 1946.
- [3] A. Wald, *Sequential Analysis*, Wiley, New York, 1947.
- [4] K.J. Arrow, D. Blackwell, and M.A. Girshick, "Bayes and Minimax Solutions of Sequential Decision Problems," *Econometrica*, 17, 1949, pp. 214-244.
- [5] K.J. Arrow, T.E. Harris, and J. Marschak, "Optimal Inventory Policy," *Econometrica*, 19, 1951, pp. 250-272.
- [6] A. Dvoretzky, Kiefer, and J. Wolfowitz, "The Inventory Problem: I. Case of known Distributions of Demand," *Econometrica*, 20, 1952, pp. 187-222.
- [7] A. Dvoretzky, Kiefer, and J. Wolfowitz, "The Inventory Problem: II. Case of Unknown Distributions of Demand," *Econometrica*, 20, 1952, pp. 451-466.
- [8] R.E. Bellman, "On the Theory of Dynamic Programming," in *Proc. of the National Academy of Sciences*, 38, 1952, pp. 716-719.
- [9] R. Isaacs, "Games of Pursuit," Rand Corporation Report P-257, Santa Monica, CA, 1951.
- [10] R.E. Bellman, *Dynamic Programming*, Princeton University Press, Princeton, NJ, 1957.
- [11] R.E. Bellman and S.E. Dreyfus, *Applied Dynamic Programming*, Princeton University Press, Princeton, NJ, 1962.
- [12] R.E. Larson and J.L. Casti, *Principles of Dynamic Programming*, Control and Systems Theory Series, Volume 7, Marcel Dekker, Inc., New York 1978.
- [13] H.F. Silverman and N.R. Dixon, "A Parametrically-Controlled Spectral Analysis System for Speech," *IEEE Transactions on Acoustics, Speech and Signal Processing*, Vol. ASSP-22, pp. 362-81, October 1974.
- [14] D.P. Morgan, "Event-Synchronous Analysis for Connected-Speech Recognition," Brown University PhD Thesis, May 1988.
- [15] "TMS32010 User's Guide," Texas Instruments, Inc., 1983.
- [16] "ADSP-2100 User's Manual," Analog Devices, Inc., 1988.
- [17] "WE-DSP32 Digital Signal Processor Information Manual," AT&T Documentation Management Organization, 1988.
- [18] "DSP56000 Digital Signal Processor User's Manual," Motorola, Inc., 1986.
- [19] "Summary Description-Voice recognition Module," Interstate Electronics Corp., Anaheim, CA, 1979.
- [20] B.A. Dautrich, L.R. Rabiner, T.B. Martin, "On the Effects of Varying Filter Bank Parameters on Isolated Word Recognition," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol. ASSP-31, No. 4, Aug. 1983, pp. 793-807.
- [21] N.R. Dixon and H.F. Silverman, "The 1976 Modular Acoustic Processor (MAP)," *IEEE Trans. on ASSP*, Vol ASSP-25, No. 5, Oct. 1977, pp. 367-379.
- [22] R.F. Lyon and C.A. Mead, "A CMOS VLSI Cochlea," in *Proceedings 1988 ICASSP*, New York, April 1988, pp. 2172-2175.
- [23] J.R. Cohen, "Application of an Auditory Model to Speech Recognition," Final Program and Paper Summaries for the 1986 DSP Workshop, October 20-22, 1986, Chatham, MA, pp.6.2.1-6.2.2.
- [24] J.G. Ackenhusen and Y.H. Oh, "Single Chip Implementation of Feature Measurement for LPC-Based Speech Recognition," in *Proc. ICASSP-85*, Tampa, FL, March 1985, pp. 1445-1448.
- [25] S.E. Blumstein and K.N. Stevens, "Acoustic Invariance in Speech Production: Evidence from Measurements of the Spectral Characteristics of Stop Consonants," *Journal of the Acoustical Society of America*, No. 66, 1979, pp. 1001-1018.
- [26] M.A. Bush and G.E. Kopec, "Network-based Connected Digit Recognition," *IEEE Trans. on ASSP*, vol. ASSP-35, no. 10, Oct. 1987, pp. 1401-1413.
- [27] R.M. Schwartz, Y.L. Chow, S. Roucos and M.A. Krasner, "Improved Hidden-Markov Modeling of Phonemes for Continuous Speech Recognition," in *Proc. 1984 ICASSP*, San Diego, CA, March 1984, pp. 35.6.1-35.6.4.
- [28] L.R. Rabiner, S.E. Levinson, and M.M. Sondhi, "On the Application of Vector Quantization and Hidden Markov Models to Speaker-Independent Isolated Word Recognition," *Bell System Technical Journal*, vol. 62, no. 4, Part 1, April 1983, pp. 1075-1106.
- [29] B.H. Juang, L.R. Rabiner, S.E. Levinson, and M.M. Sondhi, "Recent Developments in the Application of Hidden Markov Models to Speaker-Independent Isolated Word Recognition," in *Proc. 1985 ICASSP*, Tampa, FL, March 1985, pp. 9-12.
- [30] B. H. Juang and L. R. Rabiner, "Mixture Autoregressive Hidden Markov Models for Speaker-Independent Isolated Word Recognition," in *Proc. 1986 ICASSP*, Tokyo, Japan, April 1986, pp. 41-44.
- [31] L.R. Bahl, P.F. Brown, P.V. de Souza, and R.L. Mercer, "Estimating Hidden Markov Model Parameters so as to Maximize Speech Recognition Accuracy," IBM Research Report, RC-13121, no. 58589, September 1987.
- [32] L.R. Bahl, P.F. Brown, P.V. de Souza, R.L. Mercer, and M. A. Picheny, "A Method for the Construction of Acoustic Markov Models for Words," IBM Research Report, RC-13099, no. 58580, September 1987.
- [33] A. L. Gorin and R. Shively, "The Aspen Parallel Computer, Speech Recognition and Parallel Dynamic Programming," in *Proc. 1987 ICASSP*, Dallas, TX, April 1987, pp. 976-979.
- [34] J.G. Ackenhusen, "The CDTWP: A Programmable Processor for Connected Word Recognition," in *Proc. 1984 ICASSP*, San Diego, CA, March 1984, pp. 35.9.1-35.9.4.
- [35] B.J. Burr, B. D. Ackland and N. Weste, "Array Configurations for Dynamic Time Warping," in *IEEE Trans. on ASSP*, vol. ASSP-32, no. 1, February 1984, pp. 119-128.
- [36] F.C. Jutand, G. Chollet, and N. Demassieux, "VLSI Architectures for Dynamic Time Warping using Systolic Arrays," in *Proc. 1984 ICASSP*, San Diego, CA, March 1984, pp. 34A.5.1-34A.5.4.
- [37] J.R. Mann, and F.M. Rhodes, "A Wafer-scale DTW Multiprocessor," in *Proc. 1986 ICASSP*, Tokyo, Japan, April 1986, pp. 1557-1560.
- [38] G. Quenot, J.L. Gauvain, J.J. Gangolf, and J. Mariani, "A Dynamic Time Warp VLSI Processor for Continuous Speech Recognition," in *Proc. 1986 ICASSP*, Tokyo, Japan, April 1986, pp. 1549-1552.
- [39] J.I. Takahashi, S. Hattori, T. Kimura, and A. Iwata, "A Ring Array Processor Architecture for Highly Parallel Dynamic Time Warping," in *IEEE Trans. on ASSP*, vol. ASSP-34, no. 5, October 1986, pp. 1301-1309.
- [40] S. Glineski, T.M. Lalumia, D. Cassidy, T. Koh, C. Gerveshi, G. Wilson, and J. Kumar, "The Graph Search Machine (GSM): A Programmable Processor for Connected Word Speech Recognition and Other Applications," in *Proc. 1987 ICASSP*, Dallas, TX, April 1987, pp. 519-522.
- [41] L.R. Bahl, S.K. Das, P.V. De Souza, F. Jelinek, S. Katz, R.L. Mercer, and M.A. Picheny, "Some Experiments with Large-Vocabulary Isolated Word Sentence Recognition," in *Proc. 1984 ICASSP*, San Diego, CA, March 1984, pp. 26.5.1-26.5.4.
- [42] A.J. Viterbi, "Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm," in *IEEE Transactions on Information Theory*, Vol. IT-13, No. 2, April 1967 pp. 260-269.
- [43] A.J. Viterbi and J.K. Omura, *Principles of Digital Communication and Coding*, McGraw-Hill Book Co, New York, 1979.
- [44] P.B. Brown, "The Acoustic-Modeling Problem in Automatic Speech Recognition," IBM Research report, RC-12750, May 1987.
- [45] A.B. Poritz, "Hidden Markov Models: A Guided Tour," in *Proc. ICASSP-88*, New York, April 1988, pp. 7-13.
- [46] K. Nagata, Y. Kato, S. Chiba, "Spoken Digit Recognizer for Japanese language," in *Proc. 4th ICA*, August, 1962.
- [47] F. Itakura, "Minimum Prediction Residual Principle Applied to Speech Recognition," *IEEE Trans. on ASSP*, vol. ASSP-23, no. 1, Feb. 1975, pp. 67-72.
- [48] H. Sakoe and S. Chiba, "Dynamic Programming Algorithms Optimization for Spoken Word Recognition," *IEEE Trans. on ASSP*, vol.

ASSP-26, no. 1, Feb. 1978, pp. 43-49.

- [49] A.H. Gray, and J. D. Markel, "Distance Measures for Speech Processing," *IEEE Trans. on ASSP*, vol. ASSP-24, no. 5, October 1976, pp. 380-391.
- [50] H.F. Silverman and N.R. Dixon, "A Comparison of Several Speech-Spectra Classification Methods," *IEEE Trans. on ASSP*, vol. ASSP-24, no. 4, August 1976, pp. 289-295.
- [51] R.M. Gray, A. Buzo, A.H. Gray, and Y. Matsuyama, "Distortion Measures for Speech Processing," *IEEE Trans. on ASSP*, vol. ASSP-28, no. 4, August 1980, pp. 367-376.
- [52] N.R. Dixon, and H.F. Silverman, "What are the Significant Variables in Dynamic Programming for Discrete Utterance Recognition?", in *Proc. ICASSP*, Atlanta, GA, March 1981, pp. 728-731.
- [53] N. Nocerino, F.K. Soong, L.R. Rabiner and D.H. Klatt, "Comparative Study of Several Distortion Measures for Speech Recognition," in *Proc. ICASSP*, Tampa, FL, March 1985, pp. 25-28.
- [54] E.L. Boccheri and G.R. Doddington, "Speaker Independent Digit Recognition with Reference Frame-Specific Distance Measures," in *Proc. ICASSP*, Tokyo, Japan, April 1986, pp. 2699-2702.
- [55] M.R. Sambur and L.R. Rabiner, "A Statistical Decision Approach to the Recognition of Connected Digits," *IEEE Trans. on ASSP*, vol. ASSP-24, no. 6, December 1976, pp. 550-558.
- [56] H.F. Silverman, D.P. Morgan, and S.M. Miller, "An Early-Decision, Real-Time, Connected-Speech Recognizer," in *Proc. ICASSP-87*, Dallas, TX, April 1987, pp. 97-100.
- [57] Y. Kato, "Words into Action III: A Commercial System," *IEEE Spectrum*, Vol. 17, No. 6, June 1980, p. 29.
- [58] Y. Kato, "NEC Connected Speech Recognition System," Nippon Electric Co., Central Research Laboratories, Research Report, 1979.
- [59] C.S. Myers and L. R. Rabiner, "A Level Building Dynamic Time Warping Algorithm for Connected Word Recognition," *IEEE Trans. on ASSP*, vol. ASSP-29, no. 2, April 1981, pp. 284-296.
- [60] C.S. Myers and L.R. Rabiner, "Connected Digit Recognition using a Level-Building DTW Algorithm," *IEEE Trans. on ASSP*, vol. ASSP-29, no. 3, June 1981, pp. 351-363.
- [61] C.H. Lee and L.R. Rabiner, "A Frame-Synchronous Level Building Algorithm for Connected Word Recognition," in *Proc. ICASSP*, New York, NY, April 1988, pp. 410-413.
- [62] C.H. Lee and L.R. Rabiner, "Connected Word Recognition using a Network-Based Frame-Synchronous Level Building algorithm, to appear in *IEEE Trans. on ASSP*.
- [63] J.S. Bridle and M.D. Brown, "Connected Word Recognition using Whole Word Templates," in *Proc. Inst. Acoustics*, Autumn Conf., pp. 25-28, November 1979.
- [64] J.S. Bridle, R.M. Chamberlain, and M.D. Brown, "An Algorithm for Connected Word Recognition," in *Proc. ICASSP*, Paris, France, May 1982, pp. 899-902.
- [65] S.M. Miller, D.P. Morgan, H.F. Silverman, N.R. Dixon, and M.N. Karam, "Real-Time Evaluation System for a Real-Time Connected-Speech Recognizer," in *Proc. ICASSP-87*, Dallas, TX, April 1987, pp. 801-804.
- [66] H. Ney, "The Use of a One-Stage Dynamic Programming Algorithm for Connected Word Recognition," *IEEE Trans. on ASSP*, vol. ASSP-32, no. 2, April 1984, pp. 263-271.
- [67] H.F. Silverman and N.R. Dixon, "Transfer Characteristic Estimation for Speech via Multirate Evaluation," in *Proc. EASCON '75*, Washington, DC, September 1975, pp. 181A-G.
- [68] H.F. Silverman and N.R. Dixon, "State Constrained Dynamic Programming (SCDP) for Discrete Utterance Recognition," in *Proc. ICASSP-80*, Denver, CO, April 1980, pp. 169-172.
- [69] M.J. Kuhn, H. Tomaszewski, and H. Ney, "Fast Nonlinear Time Alignment for Isolated Word Recognition," *Proc. ICASSP-81*, Atlanta, GA, March 1981, pp. 736-740.
- [70] J.L. Gauvain, J. Mariani, and J.S. Lienard, "On the use of Time Compression for Word-Based Recognition," in *Proc. ICASSP-83*, Boston, MA, April 1983, pp. 1029-1032.
- [71] A. Waibel and B. Yegnanarayana, "Comparative Study of Nonlinear Time Warping Techniques in Isolated Word Speech Recognition Systems," *IEEE Trans. on ASSP*, vol. ASSP-31, no. 6, December 1983, pp. 1582-1586.
- [72] D.B. Paul, R.P. Lippman, Y. Chen, and C.J. Weinstein, "Robust HMM-based Techniques for Recognition of Speech under Stress and in Noise," *Speech Tech.*, New York, NY, April 28-30, 1986.
- [73] H. Sakoe, R. Isotani, K. Yoshida, K. Iso, and T. Watanabe, "Speaker-Independent Word Recognition using Dynamic Programming Neural Networks," in *Proc. 1989 ICASSP*, Glasgow, May 1989, pp. 29-32.
- [74] F. Jelinek, "Continuous Speech Recognition by Statistical Methods," in *Proceedings of IEEE*, Vol. 64, No. 4, April 1976, pp. 532-556.
- [75] L.R. Bahl, R. Bakis, P.S. Cohen, A.G. Cole, F. Jelinek, B.L. Lewis, and R.L. Mercer, "Further Results on the Recognition of a Continuously Read Natural Corpus," in *Proc. 1980 ICASSP*, Denver, April 1980, pp. 872-875.
- [76] L.R. Bahl, R. Bakis, P.S. Cohen, A.G. Cole, F. Jelinek, B.L. Lewis, and R.L. Mercer, "Continuous Parameter Acoustic Processing for Recognition of a Natural Speech Corpus," in *Proc. of 1981 ICASSP*, Atlanta, March 1981, pp. 1149-1152.
- [77] Y.L. Chow, M.D. Dunham, O.A. Kimball, M.A. Krasner, G.F. Kubala, J. Makhoul, P.J. Price, S. Roucos, and R.M. Schwartz, "BYBLOS: The BBN Continuous Speech Recognition System," in *Proceedings of 1987 ICASSP*, Dallas, April 1987, pp. 89-92.
- [78] S.E. Levinson, "Continuous Speech Recognition by Means of Acoustic/Phonetic Classification Obtained from a Hidden Markov Model," in *Proceedings of 1987 ICASSP*, Dallas, April 1987, pp. 93-96.
- [79] A. Rudnicki, E. Lehmann, K.H. Degraaf, and L.K. Baumeister, "The Lexical Access Component of the CMU Continuous Speech Recognition System" in *Proceedings of 1987 ICASSP*, Dallas, April 1987, pp. 376-379.
- [80] L.R. Rabiner, J.G. Wilpon, and B.H. Juang, "A Continuous Training Procedure for Connected Digit Recognition," in *Proc. ICASSP-86*, Tokyo, April 1986, pp. 1065-1068.
- [81] S. Roucos, and M.O. Dunham, "A Stochastic Segment Model for Phoneme-Based Continuous Speech Recognition," in *Proc. of 1987 ICASSP*, Dallas, TX, April 1987, pp. 73-76.
- [82] K.-F. Lee, and H.-W. Hon, "Large-Vocabulary Speaker-Independent Continuous Speech Recognition Using HMM," in *Proc. 1988 ICASSP*, New York, April 1988, pp. 123-126.
- [83] K. Ganesan and P. Mehta, "An Efficient Algorithm for Combining Vector Quantization and Stochastic Modeling for Speaker-Independent Speech Recognition," in *Proc. 1986 ICASSP*, Tokyo, April 1986, pp. 1069-1072.
- [84] K.-F. Lee, and H.-W. Hon, "Speaker-Independent Phone Recognition Using Hidden Markov Models," *IEEE Trans. on ASSP*, Vol. 37, No. 11, November 1989, pp. 1641-1648.
- [85] C.-H. Lee and L.R. Rabiner, "A Frame-Synchronous Network Search Algorithm for Connected Word Recognition," *IEEE Trans. on ASSP*, Vol. 37, No. 11, November 1989, pp. 1649-1658.
- [86] T. Takara, "Isolated Word Recognition using Continuous State Transition Probability and DP Matching," in *Proc. 1989 ICASSP*, Glasgow, May 1989, pp. 274-277.
- [87] J. Schroeter and M.M. Sondhi, "Dynamic Programming Search of Articulatory Code-books," in *Proc. 1989 ICASSP*, Glasgow, May 1989, pp. 588-591.

\*This work principally supported by NSF Grants MIP-8504277 and MIP-8809742.



**Harvey F. Silverman** (S'68-M'71-SM'79) received the B.S. and B.S.E.E. degrees from Trinity College in Hartford in 1965 and 1966, and the Sc.M. and Ph.D. degrees from Brown University, Providence, RI, in 1968 and 1971, respectively. He worked for IBM at the Thomas J. Watson Research Center, Yorktown Heights, NY, from June 1970 to August 1980, working in the areas of digital image processing, computer performance analysis, and speech recognition. He was the Manager of the Speech Terminal Project from 1976 until 1980. In 1980 he was appointed Professor of Engineering at Brown University, where his interests are in the areas of parallel architectures for signal processing and speech recognition, recognition and signal processing algorithms, and hardware implementation issues. He is the Director of the Laboratory for Engineering Man/Machine Systems in the Division of Engineering at Brown and is currently the Director of Undergraduate Programs for the Division of Engineering at Brown. Dr. Silverman was a member of the IEEE Acoustics, Speech, and Signal Processing Technical Committee on Digital Signal Processing from 1972 until 1983

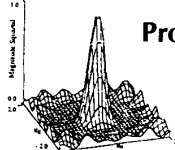
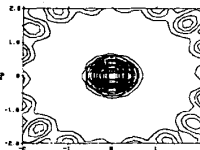
and was its Chairman from 1979 until 1983. He was the General Chairman of the 1977 ICASSP in Hartford.



**David P. Morgan** received his B.S.E.E. from the Catholic University of America in Washington, D.C. in 1983. In 1985 and 1988 respectively, he received his Sc.M. and Ph.D. from Brown University, where he studied under Dr. Harvey F. Silverman. Dr. Morgan is a member of the IEEE, Tau Beta Pi, and Sigma Xi. He has worked at the Department of Nuclear Medicine at the National Institutes of Health in Bethesda, MD; the Digital Signal Processing Division at Analog Devices in Norwood, MA; and is currently associated with the Voice Communications Initiative at Lockheed Sanders, Nashua, NH. Dr. Morgan's research activities include keyword spotting, voice synthesis, artificial neural networks, language identification and speaker identification. He has just completed a book entitled *Neural Networks and Speech Processing* for Kluwer Academic Publishers, which was co-authored with Dr. Christopher L. Scofield of Nestor, Inc., Providence, RI.

# sigx

## A General Purpose Signal Processing Package

NEW Extended Version of sig

*New Menu interface for occasional users,  
On-line HELP, and Command mode for experienced users.*

**Signal Processing Operations include:**  
Windowing, convolution, Fourier transforms, interpolation, decimation, digital filtering, linear systems, simulation, correlation, ensemble operations, spectral estimation, coherence, parametric and adaptive processing, deconvolution, identification and more...

**Graphics include:**  
Plotting, family plots, multiple viewport plots, cursor zoom, many graphic devices supported.

**New features include:**  
Surface (3D) plots, contour plots, multichannel signal and complex matrix operations, 2D Fourier transforms, optional MATLAB+ matrix package for easy algorithm prototyping with SIGX interface, additional algorithms (adaptive lattice, MLM, Burg spectral estimation, etc.), optional SIGXtutor, SIGtools and more...

**Available on:** SUN, DECstation, VAX (VMS, UNIX, ULTRIX), HP9000

**Techni-Soft** • P.O. Box 2525 • Livermore, CA 94550 • (415) 443-7213



# Status symbol

**Discover the single most vital source of technical information and professional support available to you throughout your working career... IEEE. Join us.**

**FOR A FREE MEMBERSHIP INFORMATION KIT USE THIS COUPON.**

Name \_\_\_\_\_

Title \_\_\_\_\_ ( ) Phone \_\_\_\_\_


Firm \_\_\_\_\_

Address \_\_\_\_\_

City \_\_\_\_\_

State/Country \_\_\_\_\_

Zip \_\_\_\_\_



**MAIL TO: IEEE MEMBERSHIP DEVELOPMENT**  
The Institute of Electrical and Electronics Engineers, Inc.  
445 Hoes Lane, P.O. Box 1331  
Piscataway, N.J. 08855-1331, USA (201) 562-5524