C. C. Tappert

Cursive Script Recognition by Elastic Matching

Dynamic programming has been found useful for performing nonlinear time warping for matching patterns in automatic speech recognition. Here, this technique is applied to the problem of recognizing cursive script. The parameters used in the matching are derived from time sequences of x-y coordinate data of words handwritten on an electronic tablet. Chosen for their properties of invariance with respect to size and translation of the writing, these parameters are found particularly suitable for the elastic matching technique. A salient feature of the recognition system is the establishment, in a training procedure, of prototypes by each writer using the system. In this manner, the system is tailored to the user. Processing is performed on a word-by-word basis after the writing is separated into words. Using prototypes for each letter, the matching procedure allows any letter to follow any letter and finds the letter sequence which best fits the unknown word. A major advantage of this procedure is that it combines letter segmentation and recognition in one operation by, in essence, evaluating recognition at all possible segmentations, thus avoiding the usual segmentation-then-recognition philosophy. Results on cursive writing are presented where the alphabet is restricted to the lower-case letters. Letter recognition accuracy is over 95 percent for each of three writers.

1. Introduction

The advent in the early 1960s of electronic tablets capable of accurately capturing the *x-y* coordinate data of pen movement precipitated activity on cursive writing recognition [1, 2]. Except for an occasional thesis, activity in this area has decreased in recent years. Almost all efforts have been restricted to noncapital roman script [1, 2]. Two major approaches have been described in the literature. The approach undertaken at MIT by Eden and his students was basically one of "analysis by synthesis," where a model is created for the handwriting process and decoding is performed by fitting the model's production parameters [3]. Several more direct approaches were used by Harmon and his colleagues at Bell Laboratories, the most successful being an analysis on a letter-by-letter basis following explicit segmentation. Letter accuracy for this technique was first reported at 60 percent [4] and later, after improvements, at 90 percent [2] for carefully formed script.

However, with the increased interest in office systems, particularly those that reduce the principal's dependence on secretarial support, there is also a growing interest in communicating with machines in a person's natural modalities, such as speech and handwriting. For direct written input by principals, perhaps the most difficult technical problem is that of cursive script recognition.

The dynamic programming technique of elastic matching (dynamic time warping) was applied to speech recognition problems over a decade ago and has since become widespread [5–7]. Recently, elastic matching has been successfully applied to the recognition of discrete handwritten characters, where an input character is matched against each of a set of prototypes and assigned the name corresponding to the prototype yielding the best match [8]. Here, elastic matching is extended to the recognition of cursive writing. The procedure decodes handwritten words into estimated strings of letters. Operating on a word at a time, using letter prototypes, and allowing any letter to follow any letter, the decoder uses elastic matching to find the prototype sequence which best fits the unknown word, yielding the corresponding letter sequence as the estimated letter string. Elastic matching provides the decoder with the essential feature of being insensitive to minor perturbations of input letter shapes relative to prototype letter shapes. Thus, elastic matching is

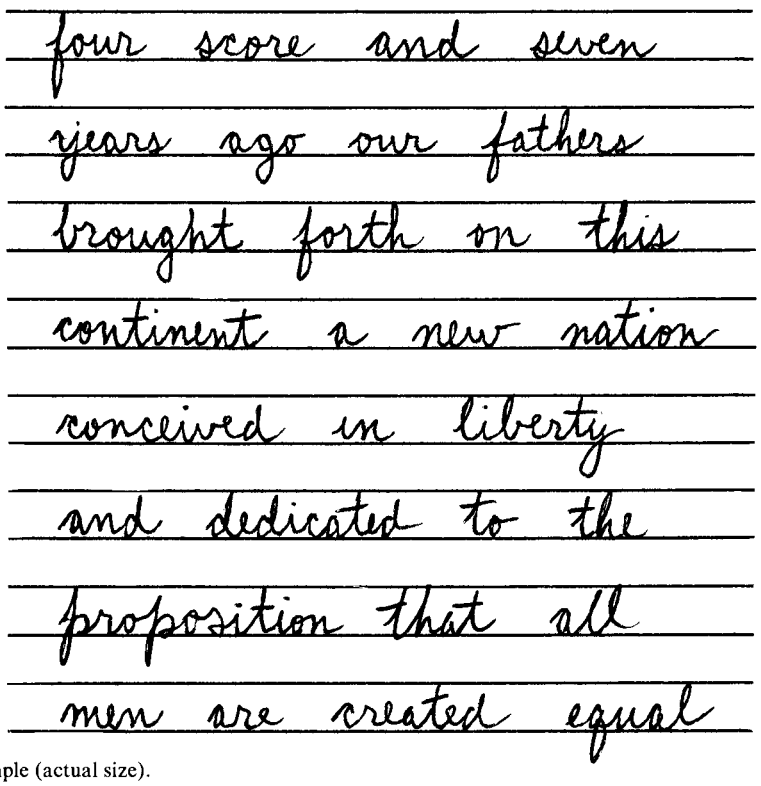

Figure 1 Handwriting sample (actual size).

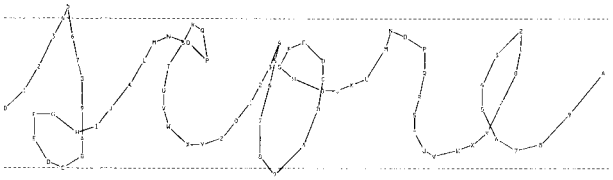

Figure 2 Machine representation of the word "score."

used to normalize expected differences in writing rate which cause repeated writings of the same letter to vary in the quantity of sample points and also cause nonlinear time variation in the shape of a letter. Explicit letter segmentation is not performed. Rather, elastic matching permits evaluation of recognition at all possible segmentations and simultaneously obtains the best combination of segmentation and recognition. The procedure used here is a one-pass, dynamic programming match similar to that previously applied in continuous speech recognition [9–11].

2. Data taking

An electronic tablet is used which detects and quantifies the motion of a pen to provide digital representations of the x and y coordinates of the pen tip. A pen up-down signal is also provided. The tablet has a resolution of 0.005 inch, and data are sampled at 70 samples/second. Program communication is via an IBM 3277 terminal. The writing sample is displayed on a Tektronics 611 direct-view storage display and, if accepted by the user, transmitted to an IBM System/370 computer for processing. The use of electronic tablets provides on-line source data of a nearly exact trace of the path of the tip of the writing instrument; these data give the number, order, and direction of the strokes used in writing. A stroke consists of the coordinate points from pen down to pen up. Lines of words to be recognized are written on ¼-inch lined paper registered on the tablet. An example of actual data entered is shown in Fig. 1. Figure 2 shows the machine representation of the word *score* of Fig. 1. In this representation straight lines connect coordinate points.

3. Procedures

This section begins with a description of several preliminary processing steps, the measurements derived from the sequence of coordinate data, and the metric used to compare such measurements of the input writing with those of letter prototypes. The elastic matching procedure is then described, going from the formulation of an optimization problem to its dynamic programming solution. Two enhancements which act as constraints on the model are also presented. Finally, the system's three modes of operation—training, recognition, and labeling—are described.

Several preliminary processing steps are performed on the data. First, a line of writing is automatically separated into words; it is assumed that the writer leaves sufficient space between words. Second, due to inaccuracies of pen up-down detection in the hardware, occasionally extra data are recorded having the appearance of a hook at the beginning or end of a stroke. Such hooks are eliminated by an algorithm which looks for big angle changes close to the ends of strokes and eliminates the points from the angle to the stroke end. Although this was a severe problem for the recognition of discrete handwritten characters [8], it is less so for cursive writing where there are fewer strokes per character. Third, as a form of "jitter" reduction, the data corresponding to a dot are reduced to a single point. Fourth, hesitation and pausing as well as other writer variability with regard to speed of writing are reduced by retaining those data points which are roughly equally spaced in distance.

A final processing step is performed on delayed strokes. A delayed stroke is one used to complete a character but which does not immediately follow the first portion of that character. For example, the word *city* is generally written with three strokes—the first is the main portion of the word, the second the dot of the *i* and the third the cross of the *t*. Here, the second and third strokes are delayed. However, because an unknown word is matched with a direct concatenation of letter prototypes, it must be possible to partition the time sequence of strokes into letters. Therefore, a special procedure moves delayed strokes to their appropriate places within a word. This procedure repeatedly examines the last stroke of a word and, if it is a dot or cross, cuts (if necessary) the underlying stroke into two strokes and moves the dot or cross by reordering the strokes so that it immediately follows the main part of the character *i*, *j*, *t*, or *x*. The resulting stroke sequence corresponds to that which would be obtained from writing in a manner such that each letter is completed before beginning the next.

From the coordinate data remaining after the above processing, two measurements are extracted to characterize each coordinate point. These measurements or model parameters were chosen for their properties of invariance with respect to size and translation of the writing (Fig. 3). The first of these parameters is the slope angle, ϕ_i, of the tangent to the curve at an arbitrary point *i*. It is approximated simply by the slope angle of the line segment from one point to the next. This angle parameter is invariant to size and translation of the input symbol. The second parameter is the height of the point, y_i, as measured from the current baseline and normalized by the line spacing of the lined writing paper attached to the tablet. This parameter is invariant to *x*-translation and, because it is normalized, to size of the input symbol. Except for size and translation these two parameters can recreate the shape of a stroke and generally of a word as well. Following preprocessing and parameter derivation a word token is represented by a sequence of parameter vectors $S = V_1, V_2, \cdots, V_N$, where $V_i = (\phi_i, y_i)$ and N is its length.

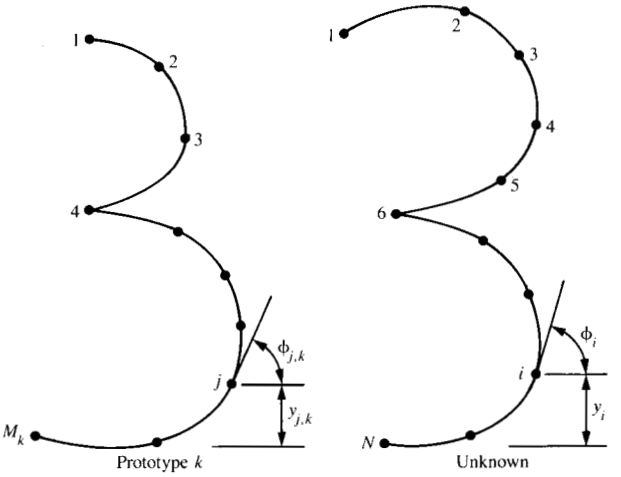

Figure 3 Model parameters: angle ϕ and height y.

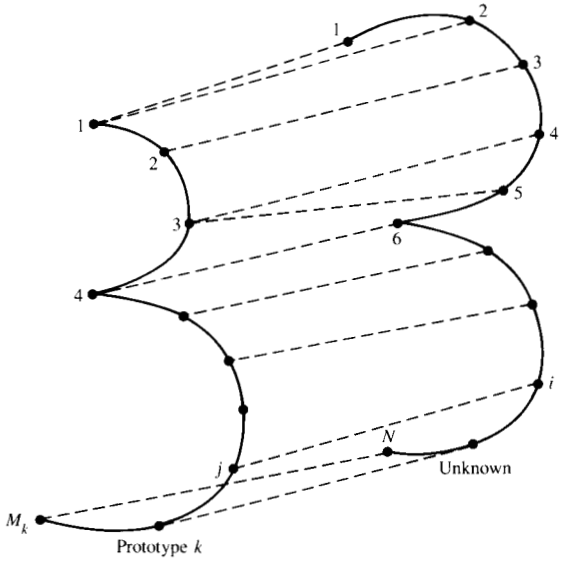

Figure 4 Elastic matching.

Letter prototypes are used to provide references against which the unknown letter sequence is matched. A letter can have more than one prototype. In order to perform this matching, a distance or metric is necessary in the comparison of an arbitary point *i* in the unknown with a point *j* in the *k*th prototype. The distance used is

$$d(i, j; \mathrm{k}) = \min\{|\phi_i - \phi_{j,k}|, 360 - |\phi_i - \phi_{j,k}|\} + |y_i - y_{j,k}|, \quad (1)$$

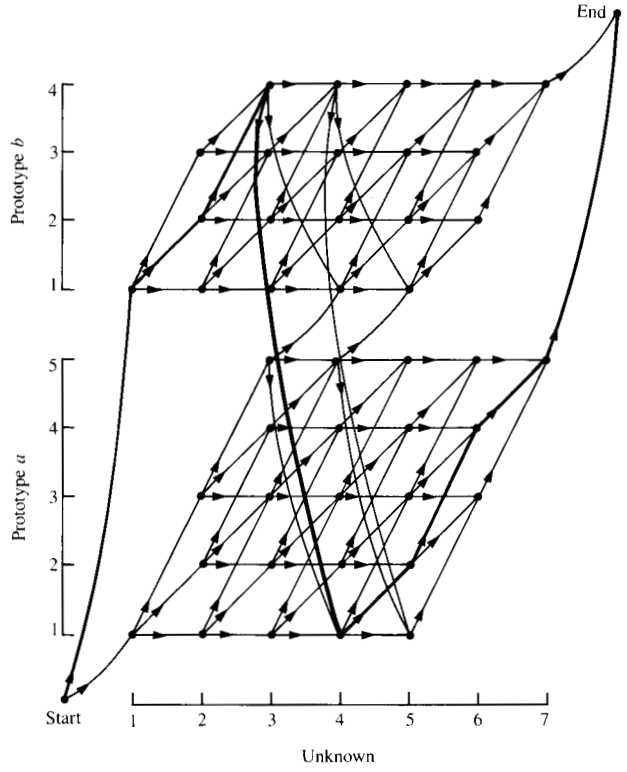

Figure 5 Lattice of simple, two-prototype model.

where the first term is the magnitude of the angle between the directed tangents and the second is the magnitude of the height difference (Fig. 3). Height is scaled to contribute the same weight as the angle parameter in the metric.

The decoder uses elastic matching at the letter level and, allowing any letter to follow any letter, finds the optimal letter sequence which uses up all the data points of the input word. Thus, the decoder finds the sequence of prototypes which best matches the input word. This is done by minimizing the expression

$$\min_{\substack{L,\ell_1,\ell_2,\cdots,\ell_L;\\ N_{\ell_1},N_{\ell_2},\cdots,N_{\ell_L}}} \left\{ \min_{\{w(i)\}} \sum_{i=1}^{N_{\ell_1}} d(i, w(i); \ell_1) + \min_{\{w(i)\}} \sum_{i=1}^{N_{\ell_2}} d(N_{\ell_1} + i, w(i); \ell_2) \right.$$
$$\vdots$$
$$\left. + \min_{\{w(i)\}} \sum_{i=1}^{N_{\ell_L}} d(N_{\ell_1} + N_{\ell_2} + \cdots + N_{\ell_{L-1}} + i, w(i); \ell_L) \right\}, \quad (2)$$

where L is the estimated number of letters in the word and, for the jth letter, ℓ_j is the best matching letter prototype and N_{ℓ_j} the corresponding number of points of the unknown word used to match that prototype. The N_{ℓ_j} must sum to N, the number of points in the unknown word. This optimization problem can be viewed as a global optimization of L local optimizations. For a local letter optimization, the warping function w maps the index of the unknown to the index of the prototype. The boundary conditions $w(1) = 1$ and $w(N_k) = M_k$, where M_k is the length of prototype k, ensure that the first and last points of a prototype are matched. The continuity condition $w(i + 1) - w(i) = 0, 1, 2$, operating within the scope of the prototype, provides the elasticity so that successive points in the unknown can be mapped to a single point, successive points, or points whose indices differ by two, skipping a point, in the prototype (Fig. 4).

The decoding procedure can be described with reference to the lattice of possible paths through the model. For purposes of illustration, the lattice of possible paths for a simple, two-prototype model is shown in Fig. 5. The lattice contains transitions to the first point of each prototype at the beginning of the unknown, from the last point of each prototype at the end of the unknown, from the last point of a prototype to the first of each prototype (where possible), and within each prototype according to the continuity condition. It is easily shown that the minimization of (2) is solved *efficiently and optimally* by dynamic programming using the recursion relation

$$D(i, j; k) = d(i, j; k) + \begin{cases} \min\{D(i-1, j; k), D(i-1, j-1; k), D(i-1, j-2; k)\} & \text{if } j > 2 \\ \min\{D(i-1, j; k), D(i-1, j-1; k)\} & \text{if } j = 2 \\ \min\{D(i-1, j; k), \min_m \{D(i-1, M_m; m)\}\} & \text{if } j = 1 \end{cases}, \quad (3)$$

where $D(i, j; k)$ is the cumulative distance to point i in the unknown and point j in prototype k. Starting with $D(1, 1; k) = d(1, 1; k)$ for all k and $D(i, j; k)$ very large (infinite) elsewhere, the cumulative distance recurrence relation is used to obtain, for all possible paths through the model, the path having the minimum distance. Computation of cumulative distance proceeds column by column through the lattice in one forward pass through the unknown input. The first line of Eq. (3) is the recursion relation which allows transitions according to the continuity condition. The second line is similar but, being only at the second point of the prototype ($j = 2$), skipping is not permitted because, according to the boundary conditions, the first prototype point cannot be skipped. The third line holds at the beginning

Table 1 Decoder accuracy results.

Writer	*Accuracy*	*(Correct/All letters)*	*Errors*
A	98%	(228/233)	$u \rightarrow a, o \rightarrow a, w \rightarrow u, v \rightarrow n, u \rightarrow n$
B	95%	(168/176)	$y \rightarrow g, h \rightarrow k$ (2), $w \rightarrow u, o \rightarrow a, u \rightarrow a, o \rightarrow e, r \rightarrow c$
C	96%	(353/366)	$d \rightarrow cl$ (2), $a \rightarrow u$ (2), $v \rightarrow o$ (2), $i \rightarrow e$ (2), $i \rightarrow r, re \rightarrow w, e \rightarrow c, a \rightarrow ce$
All	97%	(749/775)	

of a prototype ($j = 1$) and gives the choice of repeating the prototype point or making a transition between prototypes, going from the end of one prototype to the beginning of another. In the transition between prototypes, the cumulative distance corresponding to the start of each prototype is obtained by examining the ends of the preceding prototypes and selecting the minimum cumulative distance. The number of point distances computed is limited by the size of the lattice which has less than $N\overline{M}n$ points, where $\overline{M}$ is the average prototype length and n is the number of prototypes. The letter sequence is estimated by storing pointers, which refer back to prior optimal states, while stepping through the recurrence relation, and then tracing them back from $(N, M_k; k)$ for that k which minimizes $D(N, M_k; k)$. Shown in Fig. 5 in dark lines is a possible path through the lattice corresponding to the letter sequence *ba*.

For improved performance this procedure was modified in two ways—limiting segmentation points and using digram (letter pair) statistics. Transition from one letter prototype to another (*i.e.*, segmentation) is prohibited at cusps and corners, points not headed in the appropriate angular direction (essentially up and to the right), points near ends of strokes except the last point of a stroke, and points closed above within the same stroke. The main intent of this technique is to inhibit segmentation at those points which are definitely not segmentation points with the aim of avoiding segmentation errors and speeding computation since the possibilities are more limited.

Digram frequency information in the form of digram weights (penalties) derived from digram transition probabilities is employed. On making the transition from one prototype to another the digram weight corresponding to the appropriate letter pair is added to the cumulative metric. A digram weight is proportional to the negative of the logarithm of the corresponding transition probability. The rationale for this definition is essentially that adding logarithms corresponds to multiplying probabilities; the relationship between probabilistic models and additive distance metrics can be found in [12]. The transition probabilities used were based on a text containing 10 000 letters; zero transition probabilities were modified to a small value so as not to prohibit any transition.

The program in which the procedure is implemented has three modes of operation—training, recognition, and labeling. In the training mode, an initial set of prototypes is created simply by storing parameters derived from script letter tokens which are written discretely. In the recognition mode, recognition is performed by matching the unknown word against the prototypes as described above and choosing that sequence of letters yielding the smallest overall distance. In the labeling mode, additional prototypes obtained from cursive writing are added to the set of prototypes. The decoder is forced to yield the letter sequence of a specified text. This is done for each word by rigging the digram weights so as to permit only those letter pairs which occur in the word. Since the decoder yields an estimate of segmentation boundaries as well as an estimated letter sequence, the obtained segments correspond to letters and are used to create new prototypes.

The procedure is implemented in Pascal/VS, an IBM version of Pascal, and runs on the IBM System/370 computer. The current total storage requirement is 200K bytes, and processing speed in the recognition mode is 3 letters/second using 52 prototypes. Computation time is proportional to the number of prototypes used in the matching process.

4. Preliminary experimental results

Experimental results were obtained for carefully produced writing samples from three writers. The writers were instructed to include all ligatures; this point is discussed further in the following section. For each writer, a set of 165 letter prototypes was established by adding 113 prototypes from the cursive writing of a specified text to 52 prototypes (two of each letter) from discretely written cursive characters. The decoder was then run on new samples of cursive writing from each writer using prototypes obtained from the writer. These new writing samples were based on texts of entirely different material than that used for obtaining prototypes.

For each of the three writers, letter accuracy and a list of the letter errors are shown in Table 1. One of the writing samples from writer C is shown in Fig. 1. The corresponding machine output of the decoder is shown in Fig. 6. For this

four score and seven
yccers ago our futhers
brought forth on this
continent a new nation
conceived in liberty
and declicated to the
proposition that all
men are created equal

Figure 6 Decoded output.

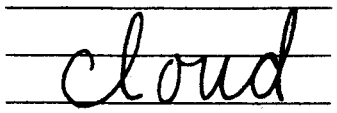

Figure 7 Example of missing ligatures.

writing sample there were four letters incorrectly decoded, $e \rightarrow c$, $a \rightarrow ce$, $a \rightarrow u$, and $d \rightarrow cl$.

5. Discussion

The above results, though based on small samples of writing from three writers, indicate that elastic matching is a promising technique for the recognition of cursive writing. Since these are early results of a new application of elastic matching, improvements can be expected with refinements of this approach.

Many of the recognition errors are the result of the decoder's inability to detect closure sufficiently. For example, the essential difference in discriminating between the pairs *a-u*, *g-y*, *o-v*, *n-u*, *a-ce*, and *d-cl* is that the first member of the pair is more closed on top. Closure is a global property and is not well characterized by the local angle and height measurements. Recognition accuracy should be substantially improved by incorporating measurements that adequately detect degree of closure. The two $i \rightarrow e$ errors arose primarily because the writer forgot to dot these *i*'s. In addition, the decoder does not adequately detect the difference between loops and cusps.

In cursive writing ligatures connect letters and, in English, these ligatures are normally high (about midline height) following *b*, *o*, *v*, *w* and low (baseline height) otherwise. As illustrated in Fig. 7, ligatures can be omitted. Here, the initial ligature of the first letter, the initial ligature of a medial letter, and the final ligature of the final letter are omitted. In order to limit the number of prototypes required in this feasibility study, the writers were instructed to produce initial and final ligatures for all letters. For cursive writing by writers who use the minimal number of pen lifts, which accounts for most, this means that care should be taken to ensure that the first letter of a word has an initial ligature and the last a final one. Note that all ligatures are included in the writing sample of Fig. 1. Now, since any letter can follow a letter of the set *b*, *o*, *v*, *w*, the procedure should account for initial high or low ligature for each letter. Therefore, the prescribed text from which additional prototypes were generated was designed to be concise and to include high and low initial ligature tokens of all letters except for a few low frequency ones.

Future studies should be concerned with permitting the no initial or final ligature cases in appropriate situations. Three possible ways of handling this are to add prototypes without ligatures, to add additional entry and exit points from prototypes with ligatures, and to add ligatures where they are omitted in the writing to be recognized. Also, in conjunction with a prototype set the decoder has a model which specifies what subset of prototypes, in terms of ligature types, can follow another. At present, any prototype can follow any other. A more sophisticated model could restrict the sequence of types as follows. The no initial ligature case can only occur at the beginning of a stroke—that is, at the beginning of a word or at the beginning of a new letter within a word when that letter begins with a new stroke. Initial high ligature can only occur at the beginning of a stroke or following the letters *b*, *o*, *v*, *w*. Initial low ligature can occur only at the beginning of a stroke or following a letter other than *b*, *o*, *v*, *w*. Finally, the no final ligature case can only occur at the end of a stroke—that is, at the end of a word or the end of a letter within a word when that letter ends a stroke.

Best results with the elastic matching technique are achieved when a writer creates his own prototypes, requiring time for the user to train the recognition system to his writing. For the system to be viable it is considered vital that it be easily trained to the writing style of an individual and that initial training should require only one or two examples of each letter so as not to burden the user. This precludes the use of an elaborate statistical approach requiring extensive user data. In addition, the system should have an adaptive capability in the sense that accuracy improves with use.

The present idea in establishing prototypes is to create progressively better prototype sets by a bootstrapping method. Beginning with a set of prototypes obtained from discretely written letters, the labeling (forced matching) procedure is used to add additional prototypes from cursive writing. This might be considered a second phase of training. These prototypes, in turn, can be used to obtain additional prototypes from other cursive writings. In order to maintain reasonable processing speed and provide sufficient storage for adding new prototypes in another labeling session, methods of deleting poor prototypes might be investigated. For example, prototypes which rarely provide the best match or are often involved in misrecognitions could be deleted since they could be considered poor representatives of their respective classes. Further study is needed of procedures for establishing a set of prototypes for a writer and perhaps also for modifying a prototype set over a period of time. Establishing prototypes is a critical procedure for a decoder of this type.

The use of digram statistics is, of course, debatable. It gives the decoder an *a priori* statistical bias and, although this bias tends to improve decoding on the average, by about three percent on a portion of our data, it can also introduce errors. Therefore, it can be argued that it is better to leave the decoder unbiased. Using digram statistics is an attempt to include language information. Perhaps a better method of doing this would be to use a dictionary to limit the decoder's choice of letter sequences to dictionary entries, since most decoding errors result in letter sequences not found in a dictionary. Further investigation is required in this area. The other modification which limits segmentation might be better omitted. An accuracy increase of one to two percent was obtained over a portion of our data, but it is easy to foresee problems with this procedure, particularly with less careful writing. Furthermore, one of the main reasons for limiting segmentation, such as not permitting segmentation at a corner, was to handle special ligature situations. For example, if segmentation is permitted at a corner, *bo* could easily be decoded as *lo* because the decoder is allowed to follow an *l* prototype with one for *o* with a high initial ligature. With improved ligature handling, as proposed above, such confusions could not easily occur and the mechanism for limiting segmentation would not be as necessary.

Finally, it should be mentioned that future work should also deal with increasing the alphabet to include upper-case letters, punctuation symbols, and numerals. The potential advantage of using a dictionary to aid decoding was mentioned above. Eventually, especially for less carefully produced writing, the use of syntax and perhaps even semantics will probably also be necessary.

Acknowledgments

It is a pleasure to thank E. C. Greanias and E. F. Yhap for their support, S. K. Das, A. S. Fox, J. M. Kurtzberg, and L. Tan for stimulating discussions, and M. Karnaugh and J. H. Morrissey for encouragement to work on this problem.

References

1. N. Lindgren, "Machine Recognition of Human Language, Part III—Cursive Script Recognition," *IEEE Spectrum* **2,** 104–116 (May 1965).
2. L. D. Harmon, "Automatic Recognition of Print and Script," *Proc. IEEE* **60,** 1165–1176 (October 1972).
3. M. Eden, "Handwriting Generation and Recognition," *Recognizing Patterns,* M.I.T. Press, Cambridge, MA, 1968.
4. L. S. Frishkopf and L. D. Harmon, "Machine Reading of Cursive Script," *Information Theory* (4th London Symp.), C. Cherry, Ed., Butterworths, London, England, 1961, pp. 300–316.
5. V. M. Velichko and N. G. Zagoruyko, "Automatic Recognition of 200 Words," *Int. J. Man-Machine Studies* **2,** 223–234 (1970).
6. H. Sakoe and S. Chiba, "A Dynamic Programming Approach to Continuous Speech Recognition," *Proc. 7th Int. Congr. Acoust.,* Paper 20, C13, August 1971.
7. F. Itakura, "Minimum Prediction Residual Principle Applied to Speech Recognition," *IEEE Trans. Acoust. Speech Signal Processing* **ASSP-23,** 67–72 (February 1975).
8. C. C. Tappert and J. M. Kurtzberg, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, unpublished results.
9. J. K. Baker, "The DRAGON System—An Overview," *IEEE Trans. Acoust. Speech Signal Processing* **ASSP-23,** 24–29 (February 1975).
10. R. Bakis, in F. Jelinek, "Continuous Speech Recognition by Statistical Methods," *Proc. IEEE* **64,** 548–550 (April 1976).
11. C. C. Tappert, "A Markov Model Acoustic Phonetic Component for Automatic Speech Recognition," *Int. J. Man-Machine Studies* **9,** 363–373 (1977).
12. C. C. Tappert and S. K. Das, "Memory and Time Improvements in a Dynamic Programming Algorithm for Matching Speech Patterns," *IEEE Trans. Acoust. Speech Signal Processing* **ASSP-26,** 583–586 (December 1978).

Received March 12, 1982; revised June 9, 1982

Charles C. Tappert *IBM Research Division, P.O. Box 218, Yorktown Heights, New York 10598.* Dr. Tappert joined IBM in 1967 and since 1972 has been a research staff member at the Thomas J. Watson Research Center. His research has generally been in the area of pattern recognition. He worked on automatic speech recognition and speech coding for about ten years. Since 1978 he has been working on tablet-oriented terminals for the future office environment, where his main interest has been in the recognition of handwriting—cursive writing as well as discretely written characters. Dr. Tappert received a B.S. in 1960 from Swarthmore College, Pennsylvania, and an M.S. in 1962 and a Ph.D. in 1967 from Cornell University, Ithaca, New York.