International Journal of Computer Vision, 5:3, 303-331 (1990)

Reading Cursive Handwriting by Alignment of Letter Prototypes

SHIMON EDELMAN AND TAMAR FLASH
Department of Applied Mathematics and Computer Science, The Weizmann Institute of Science, Rehovot 76100, Israel

SHIMON ULLMAN
Department of Applied Mathematics and Computer Science, The Weizmann Institute of Science, Rehovot 76100, Israel, and Department of Brain and Cognitive Sciences and the Artificial Intelligence Laboratory, MIT, Cambridge, MA 02139, USA

Abstract
We describe a new approach to the visual recognition of cursive handwriting. An effort is made to attain human-like performance by using a method based on pictorial alignment and on a model of the process of handwriting. The alignment approach permits recognition of character instances that appear embedded in connected strings. A system embodying this approach has been implemented and tested on five different word sets. The performance was stable both across words and across writers. The system exhibited a substantial ability to interpret cursive connected strings without recourse to lexical knowledge.

1 Introduction

The interpretation of cursive connected handwriting is considerably more difficult than the reading of printed text. This difficulty may be the reason for the relative lack of attention to the problem of reading cursive script within the field of computational vision. The present article describes progress made toward understanding and solving this problem.

We identify and discuss two main causes of the difficulties associated with handwriting recognition: uncertainty of segmentation of words into characters and variability of character shapes. We then extend a method that has been recently proposed for general object recognition, the alignment of pictorial descriptions, to handwriting recognition. A system based on the alignment of letter prototypes has been implemented and tested. Our results may indicate that the achievement of human-like performance in reading cursive handwriting is within the reach of the state of the art in computer vision.

1.1 Problems Specific to Script Recognition

The problem of character recognition inherits from general object recognition most of its difficulties (some of those, such as occlusion, are absent because characters are two-dimensional). For printed text, the difficulties can be largely overcome by present-day methods. For example, a recently developed character recognition system [Kahan et al. 1987] achieved better than 97% correct performance on mixtures of six dissimilar fonts. Even without reliance on lexical knowledge, the performance was well over 90%.

In comparison, the problem of cursive character recognition without recourse to a lexicon appeared so far to be forbiddingly difficult. Moreover, only a few attempts to attack the considerably simpler problem of recognizing handwritten words with a lexicon have been made until recently. A survey of the state of the art made in 1980 [Suen et al. 1980] contains no reference to a system that performs *off-line* recognition of connected handwritten words (most of the systems for cursive script recognition are *on-line*, that is, they rely on knowledge of the writing sequence obtained, for example, from a digitizing tablet, and not just on visual data). During the last decade, two systems that read cursive words have been implemented [Hayes 1980; Srihari and Bozinovic 1987]. These systems depend heavily on lexical knowledge, presumably because of the difficulties associated with reading cursive script. Two major sources of difficulty can be identified: ambiguity of segmentation of words into characters and variability of character shapes.

SU is partially supported by NSF grant IRI-8900267.

1.1.1 Segmentation Ambiguity. In cursive connected script, ligatures (pieces of contour connecting the characters) may constitute a substantial portion of the image, serving as a pool of contour fragments in which spurious letters may be detected. These letters, in turn, give rise to unwanted alternative interpretations that may be entirely plausible as far as the contour is concerned (if a lexicon is used, most of the alternatives can be ruled out). In other words, it is frequently possible to segment a cursive word into characters in a manner different from the original intention of the writer ([Eden 1961], see figure 1).

Segmentation is, therefore, ambiguous, especially locally, that is, when individual letters or short letter sequences are considered. At the entire string level, clashes between inconsistent local interpretations facilitate the emergence of a consistent, unambiguous global interpretation. For example, a set of local interpretations that covers the input string leaving out a substantial piece of its contour is likely to be suppressed in favor of a more comprehensive cover.

The problem of segmentation may be regarded as consisting of two subproblems: (1) finding the set of all potential segmentation points; (2) choosing from this set a subset that is, in effect, a guess about the intention of the writer. The problem is difficult because a priori any point on the contour may turn out to be a segmentation point. Thus, segmentation does not seem to be amenable to a solution by brute-force search.

In scene interpretation, segmentation (figure-ground separation) is also considered a difficult problem [Pavlidis 1977]. It is, however, less complex than the segmentation of cursive script, since natural objects, as opposed to letters, usually cannot be subdivided into parts that are, in turn, meaningful objects in the same basic category.

1.1.2 Character shape variability. The second problem that must be faced by any handwriting interpretation system is high variability of character shapes. This problem is especially severe when the text is not intended to be read by people unfamiliar with the handwriting

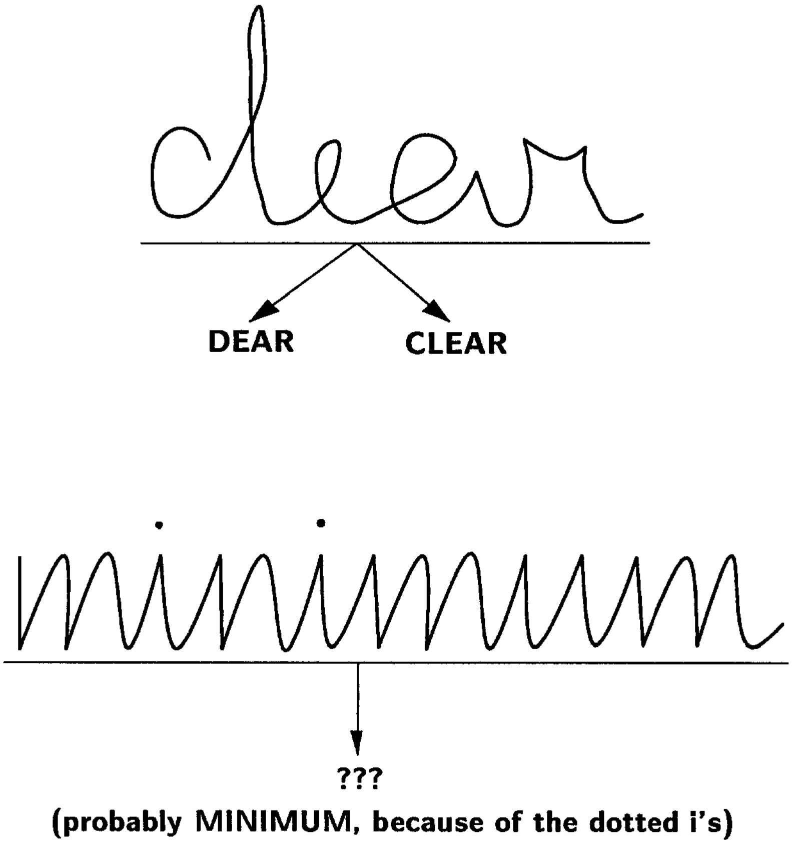


Fig. 1. The interpretation of cursive strings is often ambiguous.

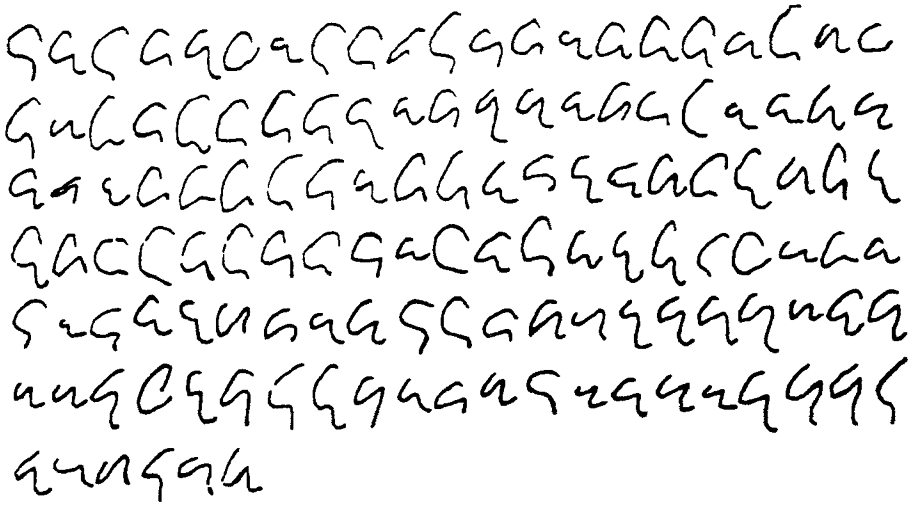


Fig. 2. 132 *a*'s taken from the notebook of P. Claudel, "Le voyage en Italie" (reproduced with permission from Pattern Recognition 11, Duvernoy & Charraut, *Stability and stationarity of cursive handwriting*, Copyright 1979, Pergamon Press plc).

Fig. 3. Different versions of the handwritten numeral 2. Both the geometry and the topology of the 2's change from sample to sample.

of the writer. An example may be seen in figure 2, reproduced from [Duvernoy and Charraut 1979]. This figure shows 132 samples of the letter *a*, taken from a notebook of the French poet, P. Claudel. If any regularity is to be found among these samples, it is hardly expressible as an isometry. Furthermore, one can easily think of examples where even the topology of the character changes from one instance to another (see the different versions of the numeral 2, some with a loop and others without a loop, in figure 3).

2 A Parallel with Three-Dimensional Object Recognition

2.1 Why Is Object Recognition Difficult

A similar problem of variability is encountered in the recognition of three-dimensional objects: an object's appearance may vary considerably depending on its pose relative to the observer. Finding regularities in the set of views that belong to a single object appears to be the only way to approach recognition, short of storing templates of all possible views of the object and comparing them with the actual view [Ullman 1986]. For simple geometrical shapes, such as triangles, regularity may be defined by specifying the set of transformations that a view of the shape may undergo. For the family of views representing a three-dimensional object, this set of allowable transformations cannot be defined easily.

2.2 Recognition by Prototype Alignment

One way to approach the problem of visual recognition is to search the space of all possible views of all stored object-models [Lowe 1986; Ullman 1986]. The purpose of visual recognition is to find a model whose match with the given object is optimal. If the viewed object is denoted by V, object models by $\{M_i\}$ and the set of allowable transformations of M_i by $\{T_{ij}\}$, then the goal of the search is to minimize some measure of distance D between the object and a model, that is, to find i, j that give $\min_{i,j} D(V, T_{ij}M_i)$.

Some machine vision systems utilize the search paradigm directly, usually in conjunction with various heuristics that reduce the necessary amount of search [Goad 1986; Grimson and Lozano-Perez 1987]. Ullman [1986] pointed out that it is possible to reduce the size of the search space considerably by computing for each model a unique transformation that aligns the model with the image in some optimal sense. The role of the aligning transformation is to normalize the problem in such a manner that the search needs to be performed over all the models, but not over their different views.

The basic idea of the alignment approach is to decompose the process of recognition into two stages. First, the transformations between the viewed object and each one of the models are determined. This is the alignment stage. Second, the model that minimizes the distance measure is found. Thus, the search is for $\min_i D(V, M_i')$, where the transformation T_{ij} that produces M_i' given V and M_i is computed in the alignment stage. The search is therefore conducted over the set of all models, but not over their different views.

The success of the alignment scheme depends on the possibility of computing the aligning transformation T_{ij}, given an image and a model M_i. The parameters of the transformation can be computed given the (3-D) coordinates of three points in the model and the (2-D) coordinates of three corresponding points in the image (the *anchor points*; see [Ullman 1986]). This scheme can compensate for transformations that include translation in the image plane (two parameters), rotation in space (three parameters), and scaling (one parameter) of model objects, followed by an orthographic imaging projection. Intuitively, the three pairs of corresponding coordinates supply the six equations that are necessary to determine the six parameters of the transformation.

The computational gain offered by the alignment approach may be estimated using a combinatorial search formulation of the recognition process, as follows [Huttenlocher and Ullman 1987]. In combinatorial terms, the goal of recognition may be defined as finding the largest pairing of model and image features for which there exists a single transformation that maps each model feature to its corresponding image feature. For i image features and m model features there are at most $p = i \times m$ pairs of image and model features. In principle, any subset of these p pairs could be the largest set of matched image and model points, making the number of matches that must be examined exponential in p. In contrast, if the search is structured as an alignment stage followed by a comparison stage, then the exponential problem of finding the largest consistent matching is reduced to the polynomial—$O(p^n)$—problem of finding the best n-tuple of matching image and model points (for the three-point alignment $n = 3$).

3 Adapting Alignment to Handwriting Recognition

3.1 Modeling Handwriting Generation Helps Recognition

It is not a priori clear whether an alignment approach could be applied to cursive script recognition. One prerequisite is that the information necessary for shape normalization should be available from the positions of several key points in the handwritten contour. Those points would then be used as anchors for computing the aligning transformation between letter prototypes and the actual image. Ideally, this transformation would remove most of the character shape variability.

A reason to believe that such key points exist was provided by studying the process of human arm movement [Hogan 1982, 1984; Flash 1983; Flash and Hogan 1985] and, in particular, of handwriting generation [Edelman and Flash 1987]. We will first present a brief summary of the generation model, and then point out its implication to the recognition problem.

- *Kinematics from shape.* The shape of a handwritten trajectory determines its kinematics (the dependence of position on time).
- *Strokes.* Cursive characters are represented and generated as concatenations of basic strokes. Although different sets of such strokes are conceivable, the repertoire appearing in figure 4 can account for the diversity of handwritten characters. The shape of a stroke is determined by the positons of just three control points (the two endpoints and a middle, *via*, point).
- *Dynamic optimization.* Stroke trajectories are planned with the minimization of a cost function as an objective. The form of the cost function reflects the characteristics of the desired trajectory. For example, smooth trajectories are produced if the cost function penalizes high value of position derivative with respect to time.

An empirical investigation [Edelman and Flash 1987] indicated that *snap* (the fourth derivative of position) has to be minimized to successfully simulate stroke trajectories recorded from subjects. Accordingly, the cost function is

$$C = \int_0^{t_f} \left(\left(\frac{d^4x}{dt^4} \right)^2 + \left(\frac{d^4y}{dt^4} \right)^2 \right) dt \qquad (1)$$

where $x(t)$ and $y(t)$ are the Cartesian coordinates of the tip of the pen. The cost function C and an appropriate set of boundary conditions (including the constraint of passing through the via point) define a variational problem, which can be solved, for example, using Pontryagin's maximum principle with equality constraints on internal points [Flash and Hogan 1985]. The resulting expression for $x(t)$ is

$$x(t) = \sum_{n=0}^{7} a_n t^n + p_x(t + t_1)_+^7 \qquad (2)$$

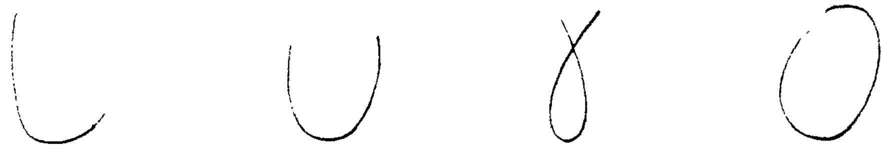


Fig. 4. The four basic stroke types—hook, cup, gamma, and oval. All cursive characters can be represented as combinations of strokes belonging to this set, with the addition of a straight-line stroke.

where a_n and p_x depend on the boundary conditions, on the positions of the control points, and on the movement duration t_f, with $(t - t_1)_+$ defined as follows:

$$(t - t_1)_+ = \begin{cases} t - t_1 & \text{if } t \geq t_1 \\ 0 & \text{otherwise} \end{cases}$$

Here t_1 is the time of passage through the via point, obtained together with a_n and p_x by solving the minimization problem. The expression for $y(t)$ is analogous to (2).

Equation (2) corresponds to a familiar result from spline theory, stating that a natural spline of degree $2m - 1$ minimizes the L_2 norm of $d^m x/dt^m$ [de Boor and Lynch 1966]. An important difference between dynamic optimization and spline interpolation is that the former specifies the time at the via point (or knot, using spline terminology). An analogous spline problem is therefore interpolation with variable knots. As opposed to the interpolation with regularly spaced knots, which is a problem of linear algebra, the variable-knot problem must be approached using the calculus of variations [Karlin 1969].

The optimization model of handwriting, called MS_1 (for minimum snap with one via point), has been evaluated experimentally, by simulating handwritten trajectories recorded from subjects [Edelman and Flash 1987]. A statistical evaluation indicated a good agreement between recorded and computed values of all kinematic characteristics of the trajectories. More important, stroke trajectories could be reliably computed from rotation and translation invariant geometrical descriptions: the relative positions of the start, via, and end points of the stroke.

3.1.1 Understanding Handwritten Shape Variability. An intuitive constraint on any model of handwriting is that similar shapes, such as the 2's appearing in figure 3, should be produced by similar motor programs. Within the MS_1 model, the apparent difference between the two 2's can be accounted for by the influence of small perturbations of control point locations on the trajectory, as follows. Consider a complex parametric curve of the form

$$z = \frac{z_0 + z_1 u + \ldots + z_n u}{m_0 + m_1 u + \ldots + m_n u} \tag{3}$$

where the m_i are real, the z_i are complex and u is a real parameter [Zwikker 1963]. Clearly, the trajectories generated by the MS_1 model are of this form (with the coefficients of $x(t)$ in (2) and of $y(t)$ in an analogous expression being $Re(z_i)$ and $Im(z_i)$, respectively). For such curves, a continuous change of the coefficients in (3) may cause the disappearance of an existing loop, or the appearance of a new one in place of a cusp (Zwikker [1963], p. 74). Now, the transformation from the control point locations to the polynomial expression for the trajectory is linear, therefore continuous. Consequently, continuous perturbations of control points may cause variability similar to the observed changes in the shape of the character "2."

Another way to regard this issue is through the formal notion of well-posedness. One of the conditions for a problem to be well-posed is continuous dependence on boundary conditions (e.g. [Torre and Poggio 1986]). Computation of the polynomial spline coefficients a_i and b_i given the control points is an instance of Hermite-Birkhoff interpolation, a well-posed problem. The continuous dependence of its solution on the locations of the control points is discussed, for example, by Ritter [1969].

3.1.2 Control Points as Alignment Anchors. The generation model described above suggests that recognition by alignment could be applied to cursive script by using the control points as anchor points for the alignment stage. The analysis of stroke recordings collected from subjects [Edelman and Flash 1987] indicates that control points (start, end, and via points of the generation model) are typically placed along the contour in positions that have simple descriptions in visual terms and therefore can be recovered from the shape of the characters. Specifically, they can be defined in terms of the local vertical axis orientation, or in terms of salient contour features such as terminators. Since the visual saliency of the control points is a prerequisite for their use in alignment, this situation is rather fortuitous. Were the control points to be found in random places along the contour, the integration between the generation model and the recognition paradigm would be less straightforward. We shall describe now the properties of the two major classes of anchor/control points that can be distinguished.

Primary Control Points. Control points that correspond to stroke start and end locations are typically situated at the vertical extrema of the contour and the line endings (including T-junctions). Their main property is stability with respect to moderate changes in the local vertical reference.

For the purpose of classification, let us define the *valency* of a contour point as the number of distinct places in which the contour crosses a small circle ascribed around the point. For primary points of valency 2, the localization of contour extrema is more stable if the curvature in the vicinity of the point is sharp rather than blunt (see figure 5). Points of valency 1 (i.e., endpoints), corresponding to an extreme case of a sharp contour, are naturally well localized. T-junctions, having valency 3, may be decomposed into an endpoint abutting a simple contour.

Note that X-crossings should not be used as anchor points because their location can change drastically with small deviations of control parameters. Indeed, a crossing may even disappear entirely, as it often does in a γ-like stroke that serves as the ascender or descender in letters such as *b*, *d*, *h*, *g*, or *y*. The account given by the generation model for this phenomenon appeared above.

Secondary Control Points. Although normally the middle (via) points of the strokes occupy primary locations along the contour, sometimes they may be found at the horizontal rather than vertical extrema. An example is provided by the via point of the ⊂-stroke that comprises the left part of an *a* (figure 6). The secondary importance attributed to a horizontal-extremum anchor point is due to its poor localization relative to the primary points. This asymmetry stems from the tendency of handwritten letters to have sharp tops and bottoms and somewhat more rounded left and right curves.

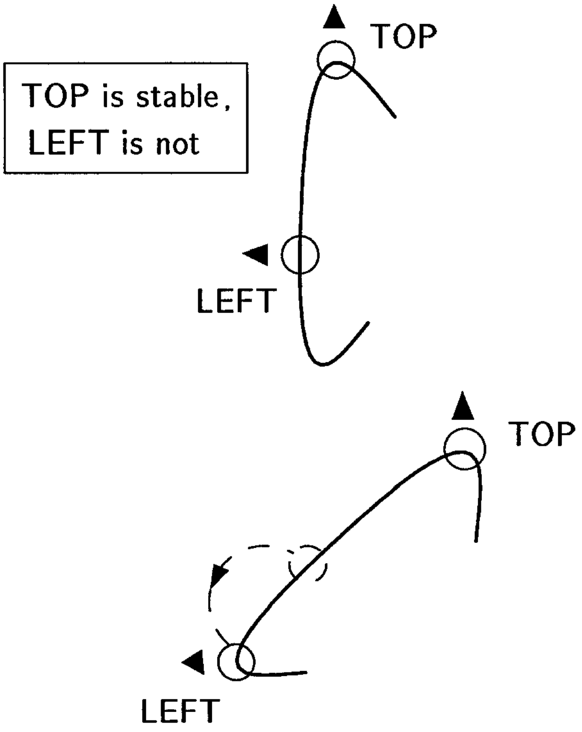


Fig. 5. Sharply bending contours allow better localization of control (anchor) points.

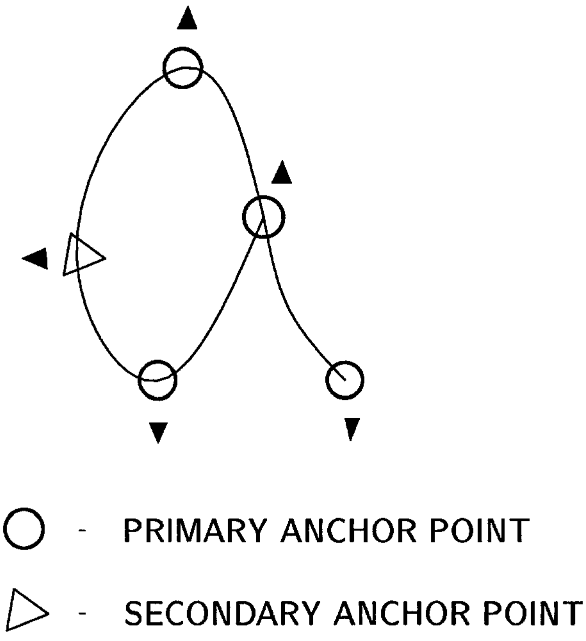


Fig. 6. Primary and secondary control (anchor) points for a particular instance of the letter *a*.

3.2 Practical Issues in Recognition by Alignment

The process of recognition involves comparison between a given shape and a set of exemplars or prototypes [Paivio 1978]. If shape variability remains even after the viewing conditons have been compensated for, for example, by alignment, then the task may be regarded as including elements of *classification*. The variability inherent in the writing process clearly makes the task of reading a matter of classification. The remainder of this section is devoted to three important issues pertaining to the application of alignment to the problem of recognition/classification of handwritten characters. The first issue has to do with representation of character prorotypes. The second deals with the transformations that the prototypes are allowed to undergo (and that are to be compensated for by alignment). The third issue is that of quantification of the discrepancies that remain after the transformations are carried out.

3.2.1 Stroke Representation. How should stroke prototypes be represented? From the computational point of view, the issue of representation is important, because it can greatly affect how easy it is to do different things with it [Marr 1982]. Consequently, if a stroke prototype is to be compared to the input image by alignment, a simple choice is to store it as a set of points and to carry out the comparison pictorially. An alternative is, for example, to store the Fourier decomposition of the stroke [Persoon and Fu 1977]. In that case, the image would have to be transformed too. Moreover, the transformation necessary for alignment would have to be figured out in terms of Fourier components rather than image elements such as points or other place tokens [Marr 1982].

An example of a pictorial representation of a stroke appears in figure 7. Characters such as *c* that contain a single stroke each are called *simple*. Most of the characters consist, however, of two strokes. These characters are called *compound*. The reasons behind the simple/compound distinction are discussed below.

3.2.2 Compound Characters. In order to be able to deal with flexible or articulated objects such as cats or compasses, the original three-point alignment scheme must be extended [Ullman 1986]. A possible extension proposed by Ullman involves the subdivision of the image into several regions by triangulation. Each of the triangular regions then undergoes independent alignment, with its vertexes serving as anchor points. In this manner, the two arms of a compass would be transformed independently, permitting it to be recognized irrespectively of the angle formed by its arms in a given image. Analogously, individual strokes that form clearly bipartite letters such as *h* or *y* should be transformed independently, because normally their parameters (e.g., size and, to a lesser extent, orientation) are free to vary independently of each other without affecting the identity of the entire letter.

Independent region transformation may not always be appropriate for nonrigid object alignment. First, it assumes the possibility of dividing the model naturally

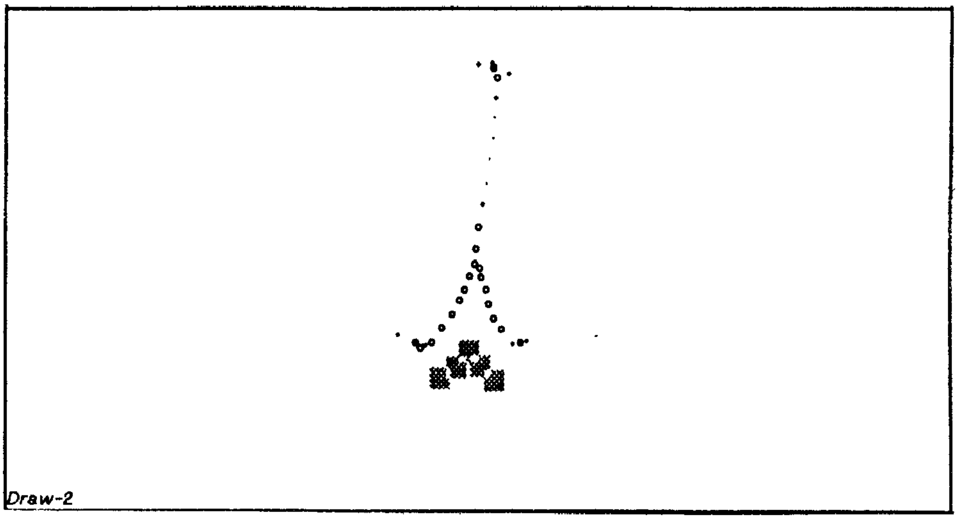


Fig. 7. The pictorial representation of a stroke prototype. This stroke may be, e.g., a part of the letter *d*. The dots and the circles signify the positions of the prototype points. The size of a circle reflects the weight given to its contribution to the prototype-image distance. The squares signify "forbidden" regions. Penalty is imposed on the stroke instance if parts of the image contour are found in these regions after the alignment.

into nonoverlapping regions (otherwise, the aligning transformation would have to be one-to-many). For some objects (e.g., a pair of scissors) such division is impossible. A second problem has to do with the boundary conditions along the region borders. In handwriting, for example, it is logical to demand that a continuous-slope contour that crosses a border should be mapped into another continuous-slope contour. The coupling between transformations of adjacent regions introduced by this constraint can complicate matters considerably.

A different solution to the problem of compound letters may be devised by combining stroke detection by alignment with letter recognition through a structural description. Note that this is different from structural recognition with sophisticated part detection [Biederman 1985], since no a priori segmentation is required and the parts need not be generic. Consider again the example of a pair of scissors. A standard structural scheme would look for the blades and the handles separately (among other reasons, because half a pair of scissors is too complicated an object to be detected directly, in a bottom-up fashion). The modified alignment method, on the other hand, would represent the scissors as consisting of two halves (with coinciding designated points), each of which is detected using model alignment.

An advantage of this approach is the simplicity of the spatial relations that must hold among object parts. For example, most compound characters can be divided into constituent strokes at cusp points, where the only required relation between parts is that of point coincidence. In addition, the decomposition into parts has psychological motivation (reviewed in [Edelman 1988]), and their number, in contrast with conventional structural descriptions, does not exceed two.

In practice, the combined stroke-structured approach may also have computational advantages. For the letter *m* (which has six natural anchor points) to be aligned with its prototype, a six-point correspondence must be established in the initial phase of the alignment. Consequently, if the points are unlabeled, and if all possible pairings are to be evaluated, then $O(n^6)$ correspondences must be tried, where n is the number of candidate anchor points detected in the image. If $n = 70$ (a typical number in our experience), the complexity is $\binom{n}{6} \approx 1.3 \cdot 10^8$. In comparison, if the letter *m* is represented by two strokes having each three anchor points, then the complexity is $2\binom{n}{3} + kn^2 = O(n^3)$, or approximately $1.1 \cdot 10^5$ for $n = 70$. The lower-order term in the above expression reflects the complexity of relationship verification. It is quadratic, since the number of detected strokes is typically linear in n. This term may be neglected, because verification of stroke relationships is less expensive than prototype transformation (i.e., k is sufficiently small). An additional advantage of the combined method is common to all structural approaches: once detected for a given word, stroke instances may be used in the recognition of several letters, resulting in yet larger savings in computation.

3.3 Transformations

The success of the alignment scheme depends on the possibility of computing the compensating transformation, given an image and a model. The choice of an appropriate set of allowable transformations for the first stage of recognition should be a compromise between the structure of the problem at hand and other considerations such as generality and computability. The structure of the problem is important because the role of the transformation stage is to cancel out image variability caused by object position, attitude and, for recognition of nonrigid objects, deformation.

The general problem of multiple anchor-point alignment is to find a warping function that agrees exactly at the anchor points while satisfying certain constraints on the transformation it represents. This problem was treated by Bookstein [1978], who studied biological shape and shape change using D'Arcy Thompson's Cartesian grid method. Bookstein developed a method of finding an optimal diffeomorphism $f: R^2 \rightarrow R^2$ that maps one shape onto another, given a set of corresponding anchor points (in his terminology, *landmarks*).

The optimality condition used by Bookstein states that the mapping f should be as smooth as possible. Specifically, it should minimize $\int |\nabla^2 f|^2$. He suggests as a physical model the deformation of a thin elastic sheet subjected to point displacements. Functions f that possess the required minimization property can be shown to satisfy the biharmonic equation $\nabla^2 \nabla^2 f = 0$. Grimson [19], working on the interpolation of sparse stereo data, arrived at an essentially similar optimization problem. We have experimented with his solution to this problem, which used the method of gradient projection, and found it too computationally expensive to be useful in the transformation stage of recognition by alignment. In some cases, however, modeling the aligning transformation as a general diffeomorphism seems inevitable. One example is the recognition of deformable objects [Foster 1975; Chen and Penna 1986].

If the problem is to recognize a rigid object whose image is formed by an orthographic projection from 3-D into 2-D (as in [Huttenlocher and Ullman 1987]), then the transformation should include six parameters: two for position, three for 3-D orientation and one for scaling. Huttenlocher and Ullman [1987] chose to represent the transformation as a composition of translation, rotation and scaling. For the special case of flat objects the result of these transformations can be conveniently represented as an affine mapping:

$$\begin{aligned} x' &= ax + by + m \\ y' &= cx + dy + n \end{aligned} \qquad (4)$$

The six parameters of this transformation may be computed from the correspondence of three pairs of anchor points detected in the image and in the model.

If orthographic projection is assumed, and if the characters are written on a flat surface, then the aligning transformation must be at least as powerful as (4) to be able to compensate for viewing conditions. On the other hand, according to the MS_1 model of character generation, it is possible that transformation (4) could compensate for character variability, because its six parameters match the six degrees of freedom of stroke shapes (corresponding to the position of the three control points).

Formally, a stroke trajectory described by MS_1 possesses sixteen rather than six degrees of freedom (see equation (2)). On the other hand, these sixteen parameters specify more than just the shape of the stroke—the time course of the generation (writing) process is determined too, in such a manner as to match real stroke trajectories produced by subjects. Thus, a sixteen-parameter model is needed to achieve parametric similarity between real and simulated strokes, whereas just six parameters suffice for geometric similarity. In other words, to model the shape of any instance of a stroke of a certain type, it is sufficient to apply the affine transform determined by control point correspondence to the image of the prototypical stroke of that type.

Note that the claim that an affine transformation plus a small set of prototypes suffice to describe stroke shapes is empirical rather than mathematical. To test the validity of this assumption we have matched stroke shapes recorded from subjects to affine transformations of prototypical strokes. We found that the affine transform indeed compensates adequately for character variability.[1] Figure 8, for example, illustrates how well strokes obtained from a *c*-like prototype by an affine transform match instances of that type that appear embedded in cursive strings taken from four subjects. A similar example for a different stroke type appears in figure 9.

Given the coordinates of three points in the image and three corresponding points in a model, the parameters of the affine transformation that would bring the two sets of points into a perfect register may be found by solving two systems of linear equations, one for *a*, *b*, and *m*, and the other for *c*, *d*, and *n* parameters:

$$\begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ m \end{bmatrix} = \begin{bmatrix} x'_1 \\ x'_2 \\ x'_3 \end{bmatrix}$$

$$\begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{bmatrix} \begin{bmatrix} c \\ d \\ n \end{bmatrix} = \begin{bmatrix} y'_1 \\ y'_2 \\ y'_3 \end{bmatrix}$$

or, in matrix form, $PA_x = X$ and $PA_y = Y$. The two systems are nonsingular and have unique solutions $A_x = P^{-1}X$ and $A_y = P^{-1}Y$ if $det(P) \neq 0$, that is, if the three anchor points of the model (x_1, y_1), (x_2, y_2), and (x_3, y_3) are not collinear (in order for the two systems to be well-conditioned, near collinearity of the anchor points should also be avoided).

For some prototypes, more than three anchor points may be needed to perform an adequate alignment. In that case the two systems are overdetermined, but affine transformation can still be used. Instead of the inverse P^{-1} of P, the pseudoinverse P^+ [Ben-Israel and Greville 1974] may be employed to compute the affine parameters. The alignment then is not perfect, but it is optimal in a least-squares sense (the sum of the squares of distances between pairs of corresponding points is minimized). If the matrix P^TP is of full rank, then $P^+ = (P^TP)^{-1}P^T$, otherwise a different method of computing P^+, such as singular value decompositon [Ben-Israel and Greville 1974] must be used.

Affine alignment of prototypes with just two anchor points (such as simple straight strokes) leads to underdetermined linear systems. One alternative then is to obtain a minimum norm solution using a generalized inverse. Another possibility is to use four parameters: rotation, scale, and translation, that are sufficient in this case to obtain perfect alignment.

Even if the alignment is perfect, some values of the computed affine parameters may be considered implausible. This may happen, for example, when the transformation maps a rounded letter such as an *o* into an eccentric slanted ellipse, and the result is too distorted

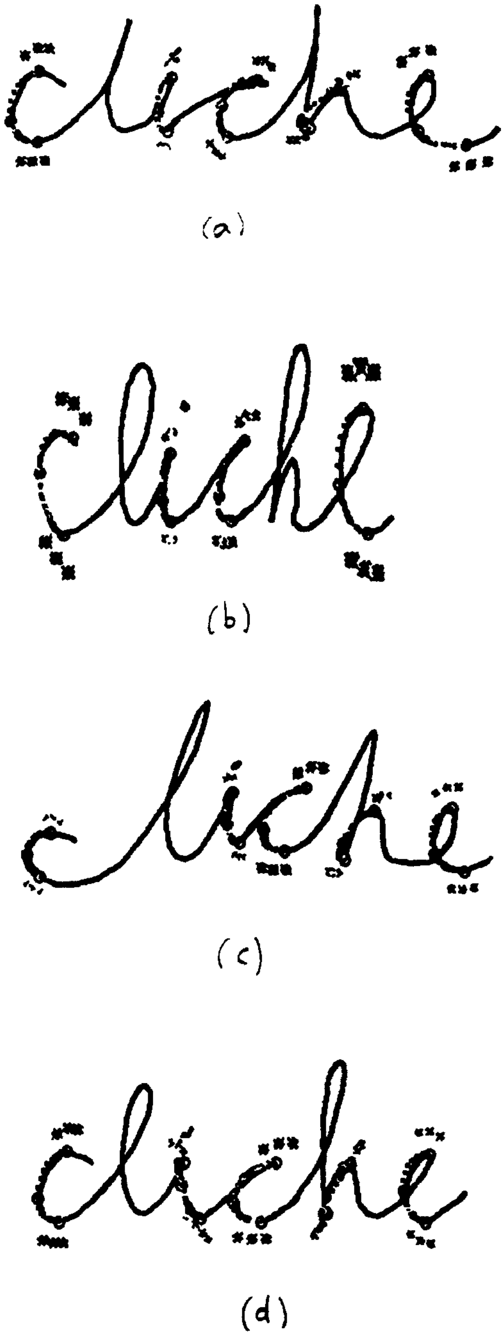


Fig. 8. Compensating for handwritten shape variability by an affine transform (see text). Here, the system looked for instances of the *c*-like stroke in four examples of the word *cliche*, written by four different subjects. The detected instances are marked by the dotted lines. The shaded areas correspond to the "forbidden" regions (see figure 7). Note that there is no objective way to determine whether a *c*-like stroke should be found in the input. It is easy to see, however, that out of the entire variety of *c*-like strokes that seem to be present in these images (20 instances) only one was missed (the borderline case that is a part of the *h* in (b)). The control points in the image were detected automatically in all cases.

to be accounted for either by personal writing style or by viewing slant. In order to attach less weight to correspondences that bring about such excessive distortion, and to save post-alignment distance computation time (by discarding implausible matches), one needs a method of estimating the distortion caused by a given affine transformation.

A possible method for estimating this distortion is computing a norm of the matrix A that represents the homogeneous part of (4) (clearly, the translation component is irrelevant). We use for this purpose the 2-norm subordinate to the Euclidean vector 2-norm $\|x\| = \sqrt{x \cdot x}$, where $(\cdot)$ designates inner product: $\|A\| = \max_{\|x\|=1} \|Ax\|$. The expression for $\|A\|$ in terms of the elements of A is obtained by the Lagrange multiplier method. Stroke instances that yield exceptionally large values of $\|A\|$ are omitted from further consideration.

3.4 Metrics

To perform the comparison between the image and a transformed prototype shape, a distance function on shapes must be defined. The choice of the distance function (specifically, its complexity) largely dictates the structure of the entire recognition process.

At one end of the complexity scale are distance functions that are sufficiently powerful to capture invariance under admissible transformations (or deformations). Powerful distance functions obviate the need for a normalization stage, such as alignment. Recognition methods that represent objects by feature vectors [Duda and Hart 1973] may be regarded as using such distance functions. Feature spaces with hundreds of dimensions are sometimes used.[2]

At the other end of the scale one finds simple methods such as template matching. The price of simplicity in this case is in the need for normalization prior to comparison. This is acceptable, as long as the source of shape variability is known and can be compensated for by an alignment-like process (as it turns out to be the case in handwriting).

If it were possible to presegment the image and to distinguish the contours belonging to the object to be recognized from those of other objects and the background, then a function that complies with the three metric axioms (non-negativity, symmetry, and triangle inequality) could be employed in the comparison stage of recognition. An example of such a function for point

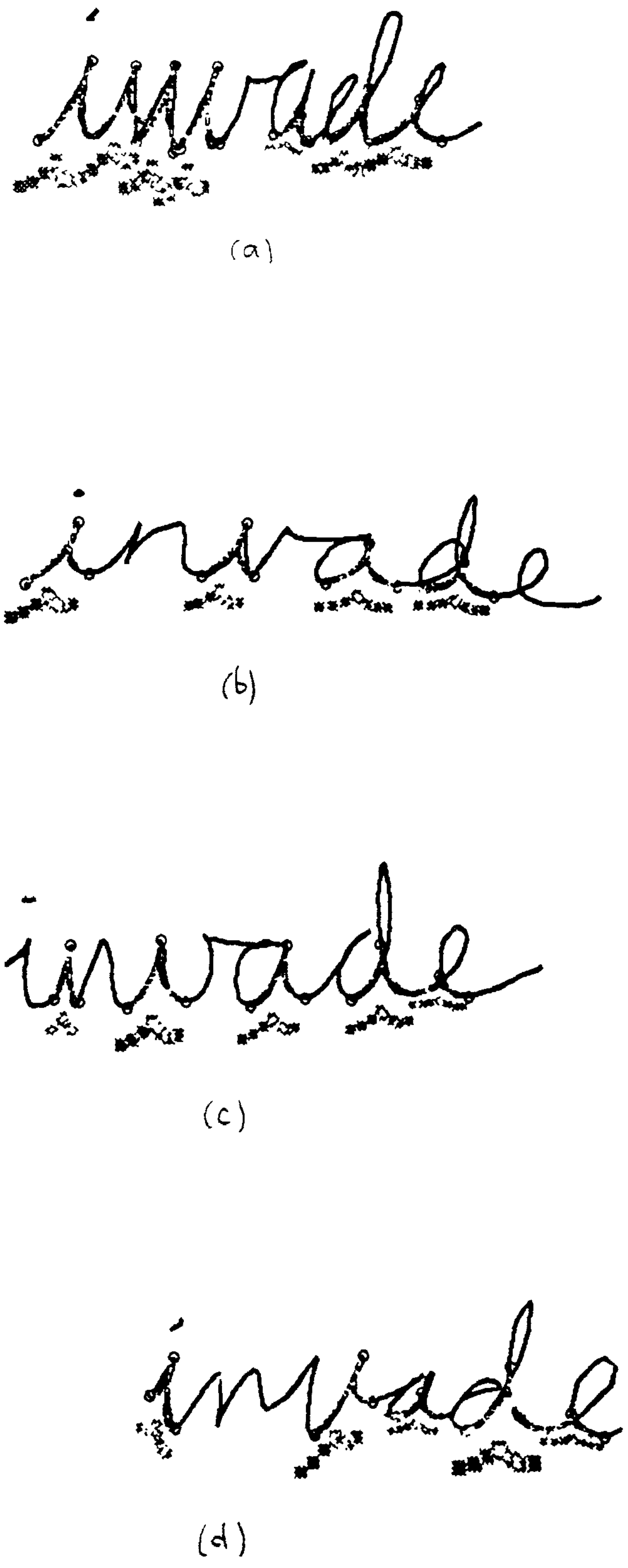


Fig. 9. Looking for instances of the *i*-like stroke in four examples of the word *invade*, written by four different subjects. All 21 of the undisputed instances were correctly detected.

sets is the Hausdorff metric (e.g., [Serra 1982]). The Hausdorff metric contains two components: an image to model distance contribution and a model to image one. To compute the first component, one must decide which part of the image corresponds to a single stroke. To avoid the problems associated with segmentation we have used an asymmetric model-to-image, distance.

For compound characters, close model-to-image fit of the individual components does not suffice to make the entire character acceptable (see figure 10). We apply two kinds of goodness-of-fit criteria for compound characters after the individual strokes have been recognized. The first relies on affine-invariant geometrical features such as ratios of distances along parallel lines. An example is the approximate equality constraint on the heights of the two components of a *w*. The second criterion has to do with the affine transformations performed on the components. The homogeneous parts of these transformations must be close for the entire character to be considered acceptable. The measure of closeness is supplied by a matrix metric. Suppose that the transformation matrixes of two stroke instances are A_1 and A_2. If the distance $||A_1 - A_2||$ is too large, the compound character formed by the two strokes is rejected.

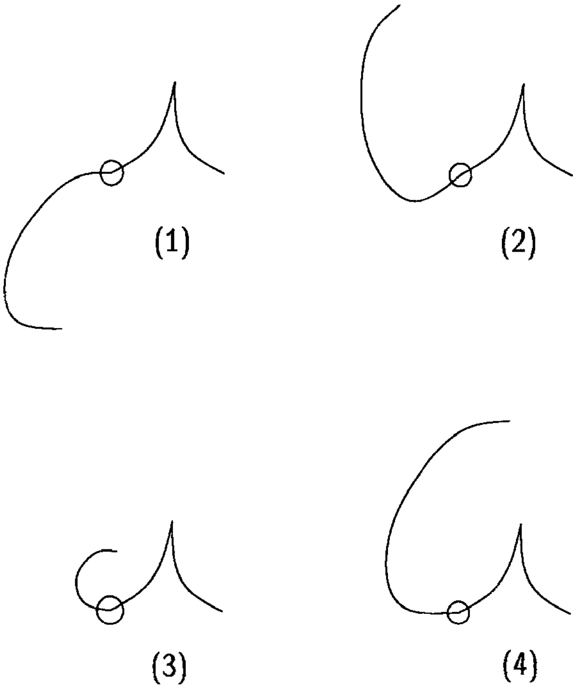


Fig. 10. Which of the four examples here would be judged as an instance of an *a*? It is not enough that the constituent strokes of a compound character are connected. The connection must be at the right place in each of the strokes, and their relative size and orientation must agree. Panels (1), (2), and (3) illustrate the possible problems, and (4) shows an admissible configuration of the two strokes.

For some letters it is desirable to penalize different components of the transformation distance by different amounts. For example, in a *d* the relative scale of the two strokes is less important than the difference in the rotations they have been subjected to, while in a *w* the opposite is true. To impose differential penalties, the matrix that defines the homogeneous part of the transformation is decomposed into a product of matrices, corresponding to the elementary transformations (i.e., scaling and rotation) that are to be discerned. We use singular value decompositon for this purpose [Ben-Israel and Greville 1974].

3.5 Summary of the Method

The main stages of the recognition process are as follows:

- *Anchor point extraction.* This is performed by tracing the image contours.
- *Stroke detection.* Strokes are recognized by prototype alignment, using affine transformations computed from anchor-point correspondences.
- *Letter hypothesization.* Potential instances of each of the 26 letters are detected. Every instance at this stage has a score that reflects its closeness to the part of the contour with which it is aligned.
- *Interpretation.* At this stage a best-first search is used to assemble the interpretation string out of the set of all detected letter instances.

We have implemented a complete word recognition system based on control point alignment. The next section describes it in some detail. We also describe several choices and techniques that are not an integral part of the basic scheme, but were found useful in the practical implementation.

4 Implementation

4.1 Extracting Alignment Tokens

In the first stage, the system traces the word contour to extract the primary and secondary tokens (anchor points) on which the subsequent stroke alignment is based. The tracing starts from the leftmost black pixel and proceeds by steps. In each step the continuation points are found by collecting all the black points that lie along the perimeter of a square centered at the current point (we set the size of the square at approximately twice the width of the contour, but any value between 1.0 and 5.0 times the width appears to work well). The continuation point that yields the lowest trace curvature is then selected. The tracing tends therefore to proceed straight wherever possible. When a continuation point is selected, the entire current square is "painted" (marked by 1's in an array congruent with the original image). Painted points are not allowed to provide continuation.

The coordinates of the centers of the squares are saved at each step. They constitute the coarse representation of the contour that is the output of the tracing. Only the local order of points matters: no attempt is made to reproduce the original writing sequence.

Two events can interrupt the tracing process. The first corresponds to a large value of the turn angle, defined in terms of three contiguous sample points. The second event is signaled by an empty continuation point set. Thus, the tracing stops when it falls off a termination of the contour, or makes too sharp a turn, or runs into an already painted area (as may happen at an intersection). When this happens, a new starting point is sought. If none is found (i.e., the entire contour of the word has been painted) the tracing process terminates, returning the set of contour pieces—lists of sample points.

Pieces of contour returned by the tracer are further processed in order to extract primary and secondary tokens: contour extrema in the vertical and the horizontal directions. First, the local orientations of the contour (stored in the sample point data structure) are smoothed by a convolution with a Gaussian mask. Consecutive members of the smoothed orientation list are then examined to detect the required extrema. A positive to negative zero-crossing signifies, for example, a maximum in the vertical direction (*top*), a transition from $\alpha > \pi/2$ to a $\alpha < \pi/2$—a local leftmost point (*left*), and so on.

Symbolic labels are assigned to the tokens in order to reduce the amount of search in the alignment stage. The classification is according to the type of the extremum: each token is labeled as *top*, *bottom*, *left*, or *right*. In addition, the tokens have property lists that include the following information:

1. *Writing zone.* Possible values of this slot are *upper*, *middle*, and *lower* (the zone is determined by a method described by Božinović and Srihari [1985]).
2. *Sub-zone.* Middle-zone tokens are further classified into *upper-middle* and *lower-middle* ones.
3. *Close tokens.* This slot points to a list of tokens that are within a threshold distance from the present token.

Figure 11 shows the primary tokens extracted from an image of the word *ornate*. Approximately 50 primary tokens were found. There were about half as many secondary tokens (not shown). At those places where the curvature of the contour around a vertical extremum is small, the localization of the tokens is poor (see the previous section). To tolerate this imprecise localization, the system is programmed to return two additional tokens, one on each side of the true one.

4.2 Finding Character Hypotheses

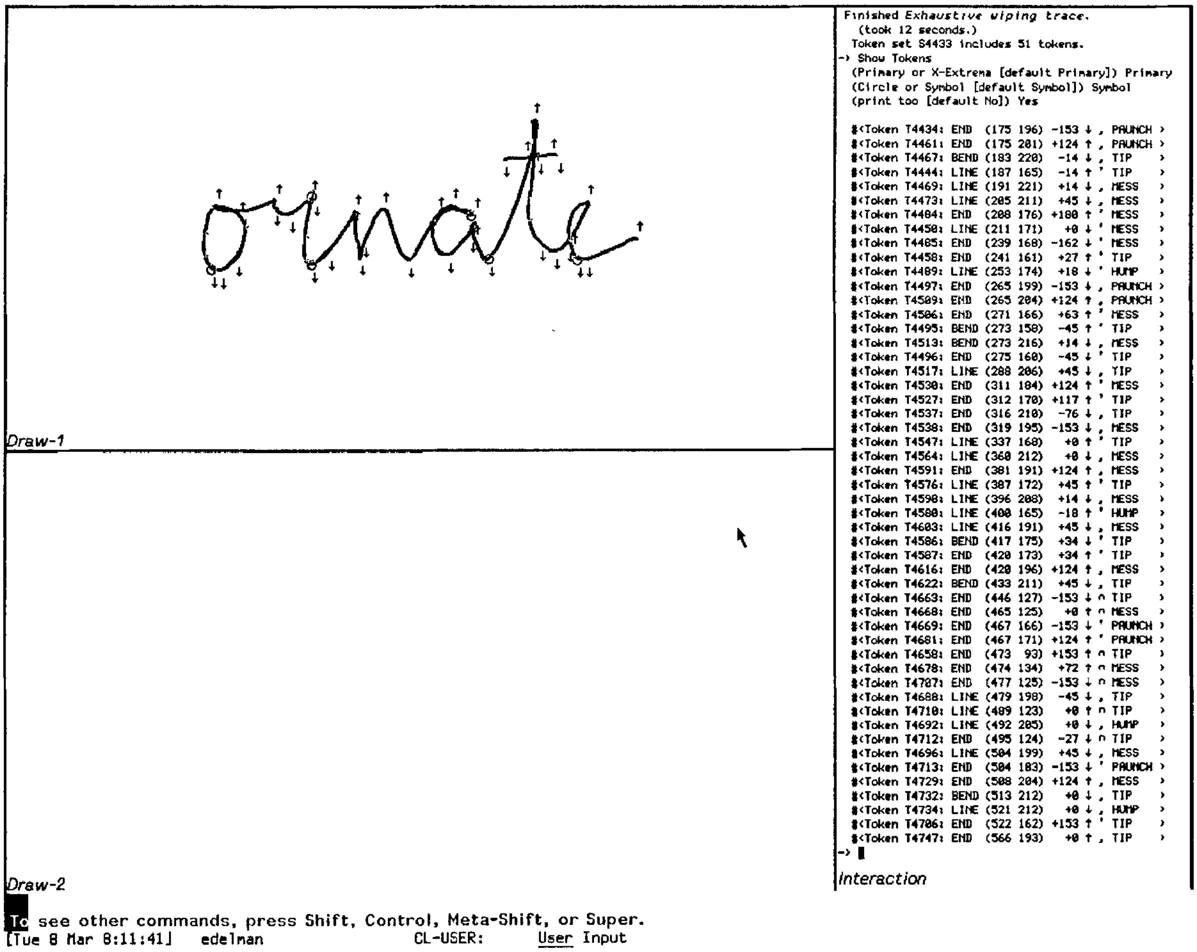
Having extracted and classified the tokens, the system proceeds to the detection of individual characters. Instances of each of the 26 characters (and a special character "&" that designates the ligatures) are detected in turn.

Correspondence and Alignment. The first step in character detection is looking up its prototypes in a special table (multiple entries correspond to different versions of the same character, such as the left and the right-facing *r*). An entry includes the names of the character's components and the name of the character *expert*: the function that is invoked to determine whether a particular combination of components is legal and to score the result if it is.

With several exceptions, the components of compound characters are simple strokes. The exceptions are *a*, *b*, *d*, *g*, *p*, and *q* which may include the letter *o*, and *m* which may include an *n*. After the system has the entire definition in terms of strokes, the prototypes of these are looked up in the prototype table. If instances of a stroke have already been detected for the current input word, they are reused. Otherwise, the strokes are found using alignment, and the result is remembered, to be used in the detection of a subsequent compound character.

To compute the aligning transformation, the system must establish correspondence between the set of prototype tokens and a subset of the tokens extracted from the image. The evaluation of potential matches is expensive: it involves computing and applying the aligning transformation and estimating the degree of fit of the result.

When token classification is used, the number of plausible correspondences is considerably smaller than



```
Finished Exhaustive wiping trace.
  (took 12 seconds.)
 Token set S4433 includes 51 tokens.
-> Show Tokens
 (Primary or X-Extrema [default Primary]) Primary
 (Circle or Symbol [default Symbol]) Symbol
 (print too [default No]) Yes

 #<Token T4434: END  (175 196) -153 ↓ , PAUNCH >
 #<Token T4461: END  (175 201) +124 ↑ , PAUNCH >
 #<Token T4467: BEND (183 220)  -14 ↓ , TIP    >
 #<Token T4444: LINE (187 165)  -14 ↑ ' TIP    >
 #<Token T4469: LINE (191 221)  +14 ↓ , MESS   >
 #<Token T4473: LINE (205 211)  +45 ↓ , MESS   >
 #<Token T4484: END  (208 176) +180 ↑ ' MESS   >
 #<Token T4450: LINE (211 171)   +0 ↓ ' MESS   >
 #<Token T4485: END  (239 168) -162 ↓ ' MESS   >
 #<Token T4458: END  (241 161)  +27 ↑ ' TIP    >
 #<Token T4489: LINE (253 174)  +18 ↓ ' HUMP   >
 #<Token T4497: END  (265 199) -153 ↓ , PAUNCH >
 #<Token T4509: END  (265 204) +124 ↑ , PAUNCH >
 #<Token T4506: END  (271 166)  +63 ↑ ' MESS   >
 #<Token T4495: BEND (273 150)  -45 ↑ ' TIP    >
 #<Token T4513: BEND (273 216)  +14 ↓ , MESS   >
 #<Token T4496: END  (275 160)  -45 ↓ ' TIP    >
 #<Token T4517: LINE (288 206)  +45 ↓ , TIP    >
 #<Token T4530: END  (311 184) +124 ↑ ' MESS   >
 #<Token T4527: END  (312 170) +117 ↑ ' TIP    >
 #<Token T4537: END  (316 210)  -76 ↓ , TIP    >
 #<Token T4538: END  (319 195) -153 ↓ , MESS   >
 #<Token T4547: LINE (337 168)   +0 ↑ ' TIP    >
 #<Token T4564: LINE (360 212)   +0 ↓ , MESS   >
 #<Token T4591: END  (381 191) +124 ↑ , MESS   >
 #<Token T4576: LINE (387 172)  +45 ↑ ' TIP    >
 #<Token T4598: LINE (396 208)  +14 ↓ , MESS   >
 #<Token T4580: LINE (400 165)  -18 ↑ ' HUMP   >
 #<Token T4603: LINE (416 191)  +45 ↓ , MESS   >
 #<Token T4586: BEND (417 175)  +34 ↓ ' TIP    >
 #<Token T4587: END  (420 173)  +34 ↑ ' TIP    >
 #<Token T4616: END  (420 196) +124 ↑ , MESS   >
 #<Token T4622: BEND (433 211)  +45 ↓ , TIP    >
 #<Token T4663: END  (446 127) -153 ↓ ∩ TIP    >
 #<Token T4668: END  (465 125)   +0 ↑ ∩ MESS   >
 #<Token T4669: END  (467 166) -153 ↓ ' PAUNCH >
 #<Token T4681: END  (467 171) +124 ↑ ' PAUNCH >
 #<Token T4658: END  (473  93) +153 ↑ ∩ TIP    >
 #<Token T4678: END  (474 134)  +72 ↑ ∩ MESS   >
 #<Token T4707: END  (477 125) -153 ↓ ∩ MESS   >
 #<Token T4688: LINE (479 198)  -45 ↓ , TIP    >
 #<Token T4710: LINE (489 123)   +0 ↑ ∩ TIP    >
 #<Token T4692: LINE (492 205)   +0 ↓ , HUMP   >
 #<Token T4712: END  (495 124)  -27 ↓ ∩ TIP    >
 #<Token T4696: LINE (504 199)  +45 ↓ , MESS   >
 #<Token T4713: END  (504 183) -153 ↓ ' PAUNCH >
 #<Token T4729: END  (508 204) +124 ↑ , MESS   >
 #<Token T4732: BEND (513 212)   +0 ↓ , TIP    >
 #<Token T4734: LINE (521 212)   +0 ↓ , HUMP   >
 #<Token T4706: END  (522 162) +153 ↑ ' TIP    >
 #<Token T4747: END  (566 193)   +0 ↑ , TIP    >
-> ▮
```

```
To see other commands, press Shift, Control, Meta-Shift, or Super.
[Tue 8 Mar 8:11:41]   edelman              CL-USER:      User Input
```

Fig. 11. Primary tokens extracted from the image of the word *ornate*. The printed representations of the tokens (ın the panel labeled "Interaction") ınclude type (*token*), identify (a unique symbol), class (*line*, *end*, or *bend*), coordinates, contour slope, extremum type (↑ for a *top*, ↓ for a *bottom*), zone (∩ for *upper* and appropriately placed, commas for *upper-middle* and *lower-middle*), and valency (see text).

the worst case. A collection of experts, one per prototype, filters the correspondences, retaining only the plausible ones. The rules employed by the experts are divided into general and prototype-specific ones. An example of a general rule states that *top* and *bottom* tokens do not match. Another example is a rule that is specific to the letter *c*, stating that the token that is matched to its top should be above the token that is matched to its bottom.[3]

Once the correspondence is established, the system computes the affine transformation that aligns the prototype with the image. Most of the prototypes have three anchor points, and some have two. The appropriate methods for computing the transformations, described in the previous section, are used in each case.

Evaluating the Match. The central assumption of the alignment method is that the entire contour of the prototype matches the image, once the corresponding anchor points have been brought into register by the aligning transformation. The goodness of the match is now computed by applying a prototype-to-image distance function, as described in the previous section.

For strokes and simple characters, the asymmetric nearest-neighbor function is applied by default. Other functions are available and may be selected through a menu. The nearest-neighbor distance is the most important but not the only component of a stroke's score: three additional factors are taken into account. The first is the amount of distortion undergone by the stroke's prototype. The distortion here is defined as $||A - kI||$, where A is the homogeneous part of the affine transformation applied to the prototype, and k is the average scaling factor, computed independently.

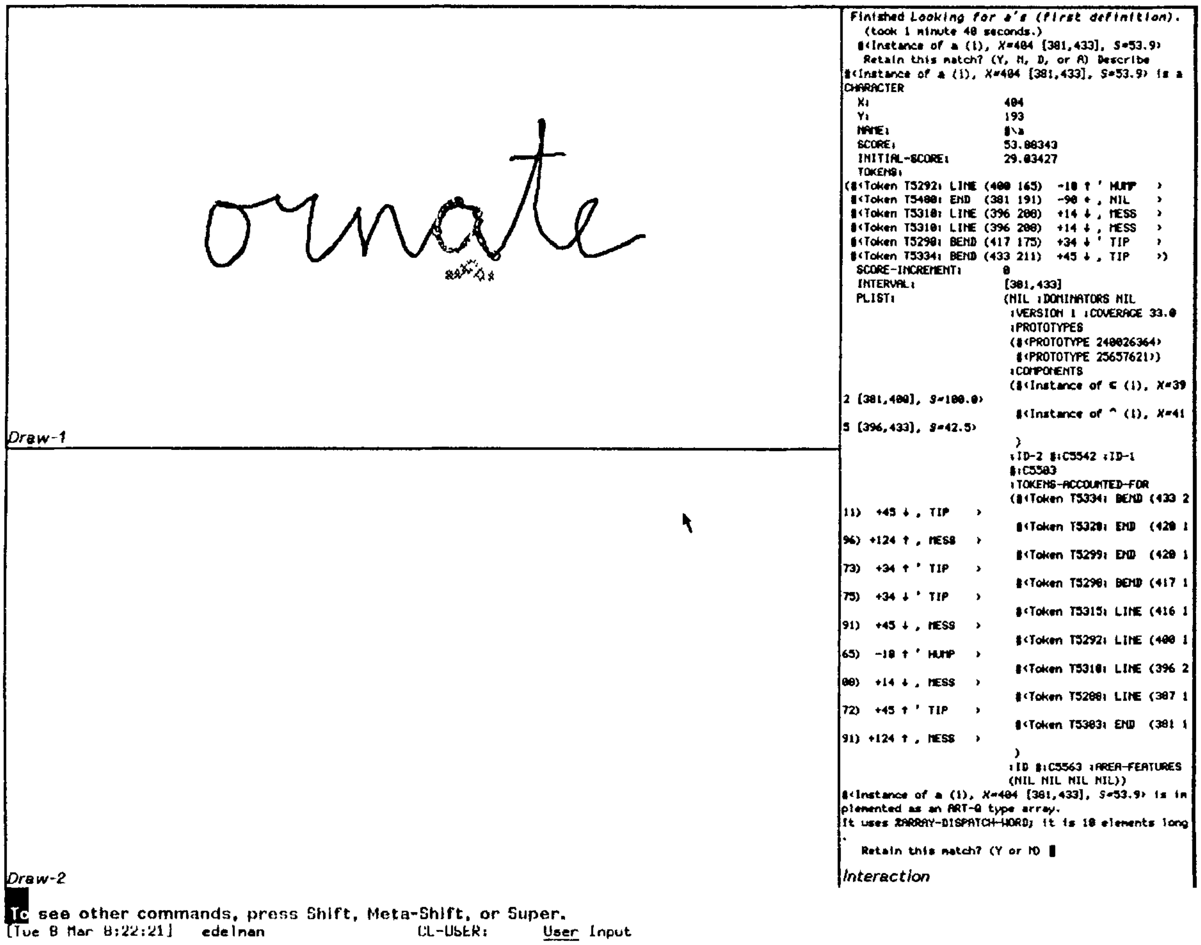
The second additional factor is prototype-specific. For an *i* it includes, for example, a bonus that depends on the location of the dot (if found) relative to the tip of the letter, for an *l*—a penalty proportional to the distance between its bottom and the lower border of the middle zone.

The third factor penalizes intrusion of image contour into forbidden zones that are also prototype-specific. For example, the top part of a *c* must have a clearance above and to the right of it, in order to reduce the score of a *c* that is embedded in a larger letter such as *d*.

The role of compound character experts mentioned above is to test combinations of strokes for legal configurations, and to evaluate the goodness of such configurations, using the method described in the previous section. The tests for legality rely mostly on computationally inexpensive, local operations, such as detecting coincidence or proximity of two tokens. For a small number of characters the tests involve, in addition, simple spatial relations (e.g., in a *t* the crossbar must at least be close to the vertical stroke).

The basis for the computation of a compound character's score is the mean of the scores of its two components. A penalty that depends on affine-invariant geometric measurements, and on the distance between the components' transformation matrices, is imposed on the basic score. The penalty computation is character-specific and is part of the character expert.

Both simple and compound character instances are represented by structures that include the following information (see figure 12):



```
Finished Looking for a's (first definition).
  (took 1 minute 40 seconds.)
 #<Instance of a (1), X=404 [381,433], S=53.9>
  Retain this match? (Y, N, D, or A) Describe
#<Instance of a (1), X=404 [381,433], S=53.9> is a
CHARACTER
 X:                     404
 Y:                     193
 NAME:                  #\a
 SCORE:                 53.08343
 INITIAL-SCORE:         29.03427
 TOKENS:
(#<Token T5292: LINE (400 165)  -10 ↑ ' HUMP   >
 #<Token T5400: END  (381 191)  -90 ← , NIL    >
 #<Token T5310: LINE (396 200)  +14 ↓ , MESS   >
 #<Token T5310: LINE (396 200)  +14 ↓ , MESS   >
 #<Token T5290: BEND (417 175)  +34 ↓ ' TIP    >
 #<Token T5334: BEND (433 211)  +45 ↓ , TIP    >)
 SCORE-INCREMENT:       0
 INTERVAL:              [381,433]
 PLIST:                 (NIL :DOMINATORS NIL
                         :VERSION 1 :COVERAGE 33.0
                         :PROTOTYPES
                         (#<PROTOTYPE 240026364>
                          #<PROTOTYPE 25657621>)
                         :COMPONENTS
                         (#<Instance of c (1), X=39
2 [381,400], S=100.0>
                          #<Instance of ^ (1), X=41
5 [396,433], S=42.5>
                          )
                         :ID-2 #:C5542 :ID-1
                         #:C5503
                         :TOKENS-ACCOUNTED-FOR
                         (#<Token T5334: BEND (433 2
11)  +45 ↓ , TIP    >
                          #<Token T5320: END  (420 1
96) +124 ↑ , MESS   >
                          #<Token T5299: END  (420 1
73)  +34 ↑ ' TIP    >
                          #<Token T5290: BEND (417 1
75)  +34 ↓ ' TIP    >
                          #<Token T5315: LINE (416 1
91)  +45 ↓ , MESS   >
                          #<Token T5292: LINE (400 1
65)  -10 ↑ ' HUMP   >
                          #<Token T5310: LINE (396 2
00)  +14 ↓ , MESS   >
                          #<Token T5288: LINE (387 1
72)  +45 ↑ ' TIP    >
                          #<Token T5303: END  (381 1
91) +124 ↑ , MESS   >
                          )
                         :ID #:C5563 :AREA-FEATURES
                         (NIL NIL NIL NIL))
#<Instance of a (1), X=404 [381,433], S=53.9> is im
plemented as an ART-Q type array.
It uses %ARRAY-DISPATCH-WORD; it is 10 elements long
.
   Retain this match? (Y or N) █
Interaction

To see other commands, press Shift, Meta-Shift, or Super.
[Tue 8 Mar 8:22:21]    edelman               CL-USER:        User Input
```

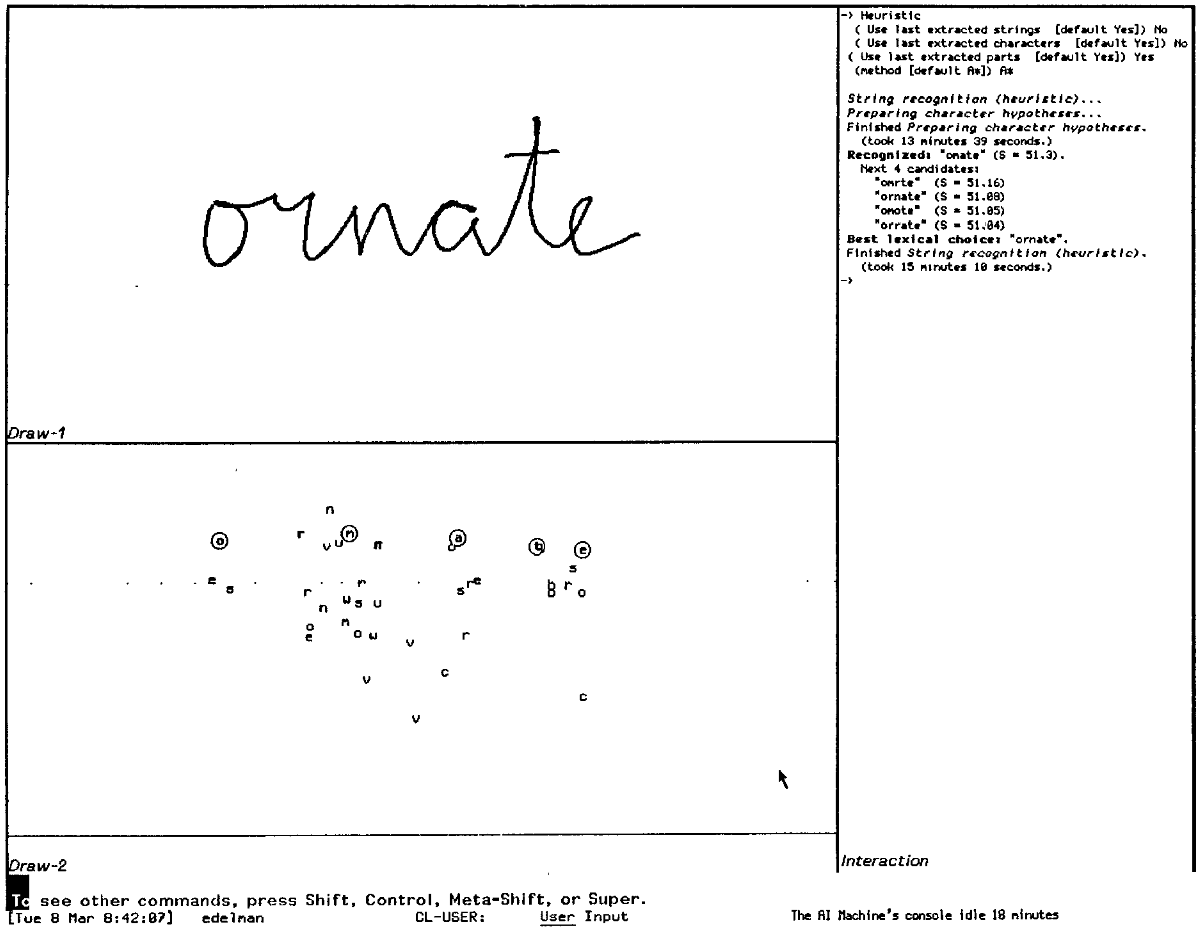
Fig. 12. An instance of the letter *a*, detected in a cursive strıng. Note the descrıptıon of the ınstance in the rıght panel.

— Location: x and y coordinates of the character's centroid
— Score
— Tokens used in the alignment
— The leftmost and the rightmost x coordinates of the interval occupied by the instance
— Version: most characters have more than one legal version or configuration
— Prototype: the transformed prototype (for compound characters, prototypes) that represents this instance
— Coverage: a list of tokens that are accounted for by this instance (i.e., are close enough to one of the points of the transformed prototype). A numerical coverage index is also stored. It is formed by assigning each covered token a contribution according to its type and location.

In this stage of the recognition process, more tokens are generated than justified by the image (for example, at a junction there are usually several tokens). All these tokens are kept, making the system more robust at the expense of an increase in the computation load. Consequently, several overlapping instances of the same character are frequently detected. Now, for every character, the set of all detected instances is filtered, to discard those that are clearly superfluous. The filtering is based on the concept of *domination.* A character instance is said to dominate another one if it has a higher score and the overlap between the two is sufficiently large.

The domination relation induces a partial order on the instance set. In the resulting partially ordered set, only those instances at least one of whose dominators is not in turn dominated may be safely discarded. The filtering is repeated in this manner, until no change happens.

The system's representation of the input after instances of all the characters have been detected and filtered is shown in figure 13. The instances are stored in a list,



```
-> Heuristic
  ( Use last extracted strings  [default Yes]) No
  ( Use last extracted characters  [default Yes]) No
 ( Use last extracted parts  [default Yes]) Yes
  (method [default A*]) A*

 String recognition (heuristic)...
 Preparing character hypotheses...
 Finished Preparing character hypotheses.
   (took 13 minutes 39 seconds.)
 Recognized: "onate" (S = 51.3).
   Next 4 candidates:
     "onrte"  (S = 51.16)
     "ornate" (S = 51.08)
     "onote"  (S = 51.05)
     "orrate" (S = 51.04)
 Best lexical choice: "ornate".
 Finished String recognition (heuristic).
   (took 15 minutes 10 seconds.)
->
```

```
To see other commands, press Shift, Control, Meta-Shift, or Super.
[Tue 8 Mar 8:42:07]    edelman              CL-USER:      User Input              The AI Machine's console idle 18 minutes
```

Fig. 13. All the letter instances detected in a string. The lower panel contains a representation of the list of instances, each placed at its proper x coordinate. The y coordinate is proportional to the logarithm of the corresponding instance's score.

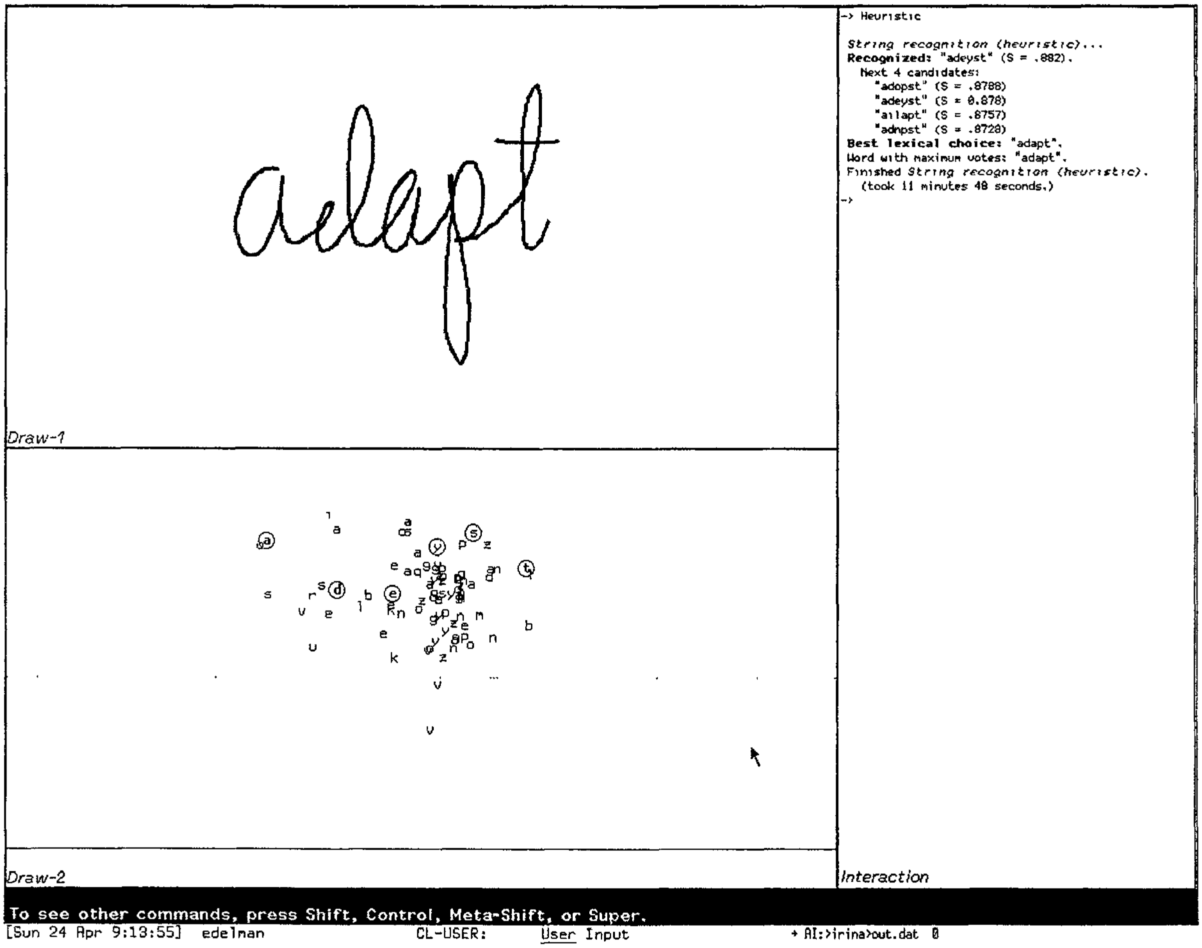
sorted by the x coordinate. A graphic representation may be seen in the lower panel of figure 13, where a letter designating each instance appears at its correct x coordinate. The y coordinate is proportional to the logarithm of the instance's score. The dotted line across the panel represents the cutoff score that can be varied to sieve out weak instances.

4.3 Choosing the Best-String Interpretation

The output of the letter recognition stage may consist of as many as 80 letter hypotheses for a 6-letter word (see figure 14). Among all these, the string recognition module must find the one that best fits the image. In addition, the next best 100 or so candidates are found (these are used by the lexical post-processor, described below). All candidates must be valid strings in the following sense: their constituent letters should preserve the original order along the x axis and should not overlap in space. This is a problem of constrained combinatorial optimization, aggravated by the need to find more than just the best result.

The problem may be solved by searching the space of all possible ordered subsets of the set L of the detected instances. The size of this space is $\binom{n}{n}$, where $n = |L|$. If only those subsets of L whose length is between 3 and 6 are considered, the size is reduced to $\Sigma^{6}_{i=3}\binom{n}{i} = O(n^6)$. The above validity requirement further reduces this figure. The remaining complexity is, however, sufficiently high to preclude the application of exhaustive search.

Our first attempt to solve the optimization problem was by a relaxation process. This process used "lateral



```
-> Heuristic

 String recognition (heuristic)...
 Recognized: "adeyst" (S = .882).
   Next 4 candidates:
     "adopst" (S = .8788)
     "adeyst" (S = 0.878)
     "ailapt" (S = .8757)
     "adnpst" (S = .8728)
 Best lexical choice: "adapt".
 Word with maximum votes: "adapt".
 Finished String recognition (heuristic).
   (took 11 minutes 48 seconds.)
->
```

```
To see other commands, press Shift, Control, Meta-Shift, or Super.
[Sun 24 Apr 9:13:55]  edelman            CL-USER:     User Input            • AI:>irina>out.dat 0
```

Fig. 14. A particularly difficult example of the combinatorial nature of the interpretation problem. Without using the lexicon, the best outcome was *adeyst*. This makes sense if one scrutinizes the image while trying to disregard the solution imposed by the lexicon. The result involving the lexicon was *adapt*, as it should be.

inhibition" among character instances in an iterative fashion. The hope was that after a number of iterations the instances that constituted a high-scoring, consistent solution would be strengthened at the expense of the weaker, conflicting instances.

In order to encourage the emergence of consistent solutions, the inhibition exerted by one instance upon another decreased with the distance between the two. It was highest when the characters actually overlapped. The scores of individual instances were taken into account by making the inhibition asymmetric (weaker instances did not inhibit stronger ones). An important role was attached to the notion of character inclusion (such as that of an *a* within a *d*). The "included" or dominated characters did not inhibit their dominators.

The lateral inhibition mechanism may be regarded a single-layer special case of an interactive activation network [McClelland and Rumelhart 1981]. As such, it could be extended in a straightforward manner to use lexical knowledge. However, our attempts to find a combination of parameters leading to an acceptable performance have failed, apparently for the following reasons. First, the domination rules that guide character-level inhibition properties that emerge at the next higher level: in groups of characters. In order to accommodate coalitions of characters into the scheme, all potential coalitions must be considered. This leads one back to the original combinatorial problem.

A second obstacle is due to the *transitivity* problem. Consider three character instances c_1, c_2, and c_3. Suppose that c_1 dominates c_2 but not c_3, and that c_2 in turn dominates c_3. The survivors of the lateral inhibition in this case should be c_1 and c_3. However, in many cases it turns out that c_2 suppresses c_3 (drives it below the extinction threshold) before c_1 can prevent it by suppressing c_2.

When the inadequacy of the lateral-inhibition approach in the present context became apparent, we replaced in a by a best-first heuristic search. In the best-first approach the tree-structured state space is searched by expanding at each step the most promising node among all those situated along the current expansion frontier (see figure 15). The search algorithm employed by the system is a variant of A^* [Pearl 1984]. We called it

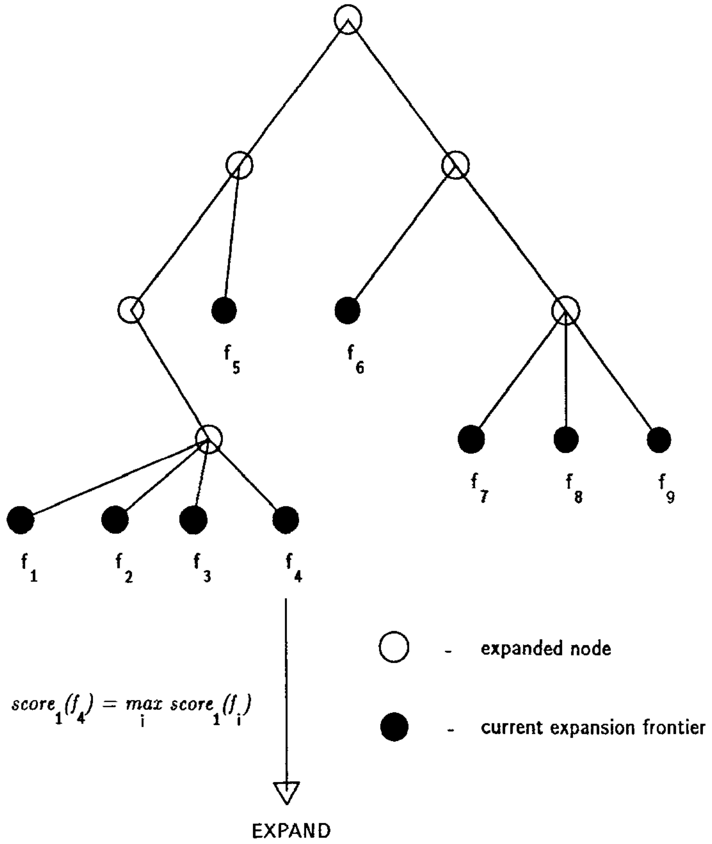


Fig. 15. Algorithm A^*, applied to the search for an optimal instance combination. At each iteration, the best candidate at the current search frontier (marked) is expanded and evaluated.

BCS for Best-Candidate Strings. It uses a heap [Aho et al. 1974] ordered by string score to hold the partial strings that belong to the current expansion frontier. This permits insertion and deletion of frontier set elements in time proportional to the logarithm of its size. BCS conducts the search from left to right, expanding at each iteration the top element of the heap, which is the best string encountered so far.

The score of a string at this stage (computed by a function, *score*$_1$) must be independent of its length, otherwise the best-first search degenerates quickly into a depth-first one. The independence is achieved by computing a basic letter-cumulative score and dividing it by the length of the string. The basic score has two main components: the sum of the scores of constituent characters, and the sum of their token coverage indices.[4] The two components are combined into a weighted sum (normally, the token coverage and the score are given equal weight). The result is multiplied by a penalty factor that depends mainly on the style consistency of the string.

The expansion of the best candidate is performed by the function *expand* which takes a string and returns all its one-letter continuations to the right, subject to two conditions: (1) validity of the result and (2) closeness of the continuation letter to the rightmost letter of the original string (this heuristic uses the fact that no gap wider than about twice the height of an average letter usually exists between two successive letters of a word). When expansion to the right is no longer possible, the string is removed from the heap and saved in an accumulator, and the entire cycle is repeated.

When a preset number of iterations is exceeded, the accumulator is sorted using a different scoring function, *score*$_2$. In contrast with *score*$_1$, this function gives preference to longer strings. Specifically, the token coverage of individual characters enters it in a cumulative fashion, while their scores are still averaged. This serves to discourage character fragmentation (e.g., breaking down a *w* into a *c* and a *u*).

4.3.1 Algorithm BCS

```
1. Initialize the heap $H$ and an accumulator $R$ to $\emptyset$ (the top of an empty heap is the null string $\epsilon$).
2. Do $h \leftarrow top(H)$; $E \leftarrow expand(h)$.
3. If $E \neq \emptyset$ then remove $top(H)$ and for all $e \in E$ insert $(e, score_1(e))$ into $H$; else remove $top(H)$ and add the result $h$ to $R$.
4. If $H = \emptyset$ or the number of iterations $N$ is exceeded, return the first 100 elements of $R$, after sorting it by $score_2$; else, go to step 2.
```

The maximum allowed number of iterations *N* was adjusted empirically to a value that prevented the algorithm from returning prematurely for all the test words. Provided that *N* is high enough, the following claims can be made about the performance of BCS. Since the path from the root of the search tree to a solution is irrelevant for the present problem, BCS satisfies the condition for optimality ([Pearl 1984] p. 62). It also trivially satisfies the completeness condition because the state space is finite (the string cannot be expanded beyond the rightmost element of *L*). The condition for admissibility is satisfied for an infinite *N*. BCS is therefore admissible on short enough inputs.

The output of the heuristic search is a list of strings, ordered according to a set of criteria that rely solely on *visual* information. For a considerable proportion of inputs, the top five strings on this list include correct interpretation (we shall describe the statistics shortly).[5] Even when they do not, the top interpretation can often be intuitively justified (see figure 14). If desired, lexical knowledge can now be applied to this raw output, as described below.

4.4 Application of Lexical Knowledge

In the present system, the application of lexical knowledge is no optional operation, and is independent of the visual recognition module, described above. Because of the independence requirement, contextual post-processing [Hanson et al. 1976] or error correction is used, rather than an integrated approach such as that of Srihari and Božinović [1987].

A contextual post-processor assumes that the letter string that is the outcome of the visual recognition stage is a garbled version of a legal word. It then uses the letter context of the string to produce the most likely word from which it could have originated. The degree of likelihood may be computed for each word in the lexicon directly, for example, using the Levenshtein metric on strings [Okuda et al. 1976]. Another possibility is to use n-gram statistics to decide which of the letters are erroneously detected [Hanson et al. 1976].

The only input to a post-processor of the above type is the best string supplied by the visual module: the less successful candidate strings are disregarded. This approach is unwarrranted in cursive script recognition, where the high inherent ambiguity of the image interpretation causes the spread of the useful information over the first several candidates. When the visual goodness-of-fit alone is considered, the best string often is an alternative parsing (interpretation) of the image. This parsing may have nothing to do with the legal word that is the intended interpretation.

In choosing the method for lexical post-processing, we also considered the requirement to simulate human ability to interpret (recognize the letters of) nonwords. In particular, the facilitation of letter recognition in pseudowords (such as pronounceable nonwords) had to be accounted for. An approach that incorporates this feature is interactive activation [McClelland and Rumelhart 1981]. However, as discussed above, it proved to be inadequate for cursive connected script.

The advantage of pseudowords over random strings may be attributed to the reader's tendency to look for meaning (Englishness) even where there is none. This makes the concept of statistical Englishness, as defined by Travers and Olivier [1978], useful in the simulation of the word (and pseudoword) superiority effect.

The lexical module relies on the synthesis of the concepts of lexical neighborhood and statistical Englishness (see [Edelman 1988] for a discussion). It uses the fast spell-checking function available in the Symbolics Lisp environment. The Englishness of a string S is estimated as

$$E(S) = \begin{cases} E_{\max} & \text{if } S \text{ is a legal word} \\ \frac{1}{|P|} \sum_{s \in P} \sum_{c \in C(s)} \text{length } (c) & \text{otherwise} \end{cases} \qquad (5)$$

where $P = \{\cup S_i \mid (S_i \subseteq S) \;\&\; (|S_i| \geq 3)\}$ is the bag (set with possible repetitions) of all substrings of S of length 3 or more, $C(s)$ is the set of *all corrections* of the string s, returned by the spell-checking function,[6] and $E_{\max}$—an empirically determined constant. The division by $|P|$ makes possible the comparison of $E(S)$ for short and long S's alike.

The E measure takes into account n-gram statistics of the language in an indirect manner, through a procedure that may be interpreted as activation of word units, with a subsequent excitation feedback to the letter unit level. The amount of excitation depends on the likelihood of letter groups. The grouping is encoded by letter positions relative to each other rather than by absolute positions. This approach incorporates the positive features of the interactive activation scheme, such as its natural account of word superiority, without suffering from its main shortcoming: the dependency on absolute positioning of letter units.

5 Performance

5.1 Evaluation Paradigm

The evaluation of the system's performance proceeded in four stages. The first stage amounted to a feasibility study, in which the best possible performance on a predetermined set of word images was sought. In the second stage, the system has been manually trained, using the same set of images. In the third stage, the system was tested on several image sets, produced by different writers. Finally, the influence of some of the system's parameters on its performance was estimated.

Six sets of *word images* (generated from four different *word* sets: see table 1) were used in this study. Four of these were produced by the first author. The remaining two sets were generated by two other writers. The six sets of images are described in table 2. For easy reference, they are designated by a capital letter (the initial of the writer), subscripted by the number of the word set.

Table 1. Word sets used in the experiments. Several *images* of some of these sets were produced (see text). The pseudowords in set #3 were obtained from legal words by substitution of one of the letters. All these pseudowords were pronounceable.

Word Set #	1	2	3	4
Description	43 words	40 words	40 pseudowords	32 words

Table 2. Image sets produced by the three writers. The set B_1' was derived from B_1 by erasing the ligatures between letters (see text).

Writer	Image Sets
EDE	E_1, E_2, E_3, E_4
BEN	B_1, B_1'
CHE	C_1

All words used in the experiments contained between three and six letters. The histogram of letter occurrences in word set number one (from which image sets E_1, B_1, and C_1 were generated) appears in figure 16. For

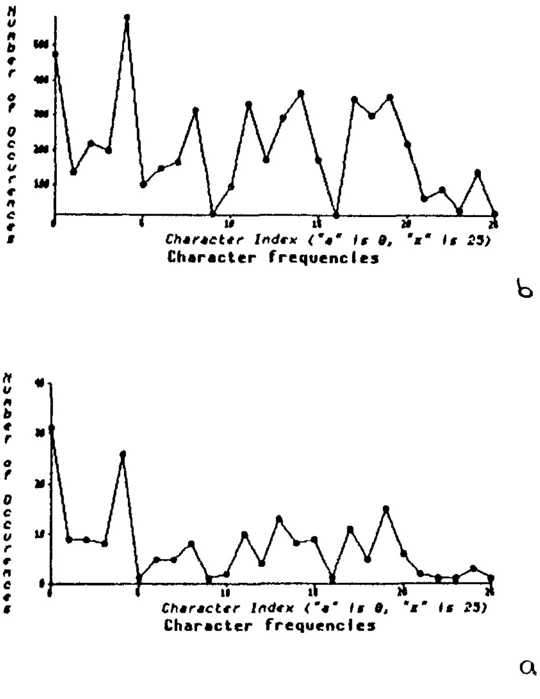


Fig. 16. (a) The histogram of letter occurrences in word set number one (from which image sets E_1, B_1, and C_1 were generated). (b) A similar histogram for a randomly chosen set of 1000 words. The two histograms are similar.

comparison, a similar histogram for a randomly chosen set of 1000 words is also shown. The images were produced by a digitizing tablet [Edelman 1988]. Ruled paper and an ordinary ballpoint refill, built into the tablet's stylus, were used. The writers were asked to write along the rulings. No other constraints on the writing style were imposed (see figure 17 for an example of an image set, C_1).

5.2 Evaluation Results

5.2.1 Feasibility Study. The purpose of the feasibility study was to find out the best performance on a fixed image set, E_1, that could be reached with a moderate investment of programming effort. Such a study was necessary, because the initial definitions of prototype characters, as well as the initial versions of character experts (see the previous section), were empirical, guided by our intuition about the problem. The first run of the program on E_1 resulted in 77% top-five correct rate.[7] This figure has been brought subsequently to 95%, with the best lexical choice being correct in 100% of the cases. When tested on E_2 and E_3, the system achieved an average correct rate of 51% without the lexicon.

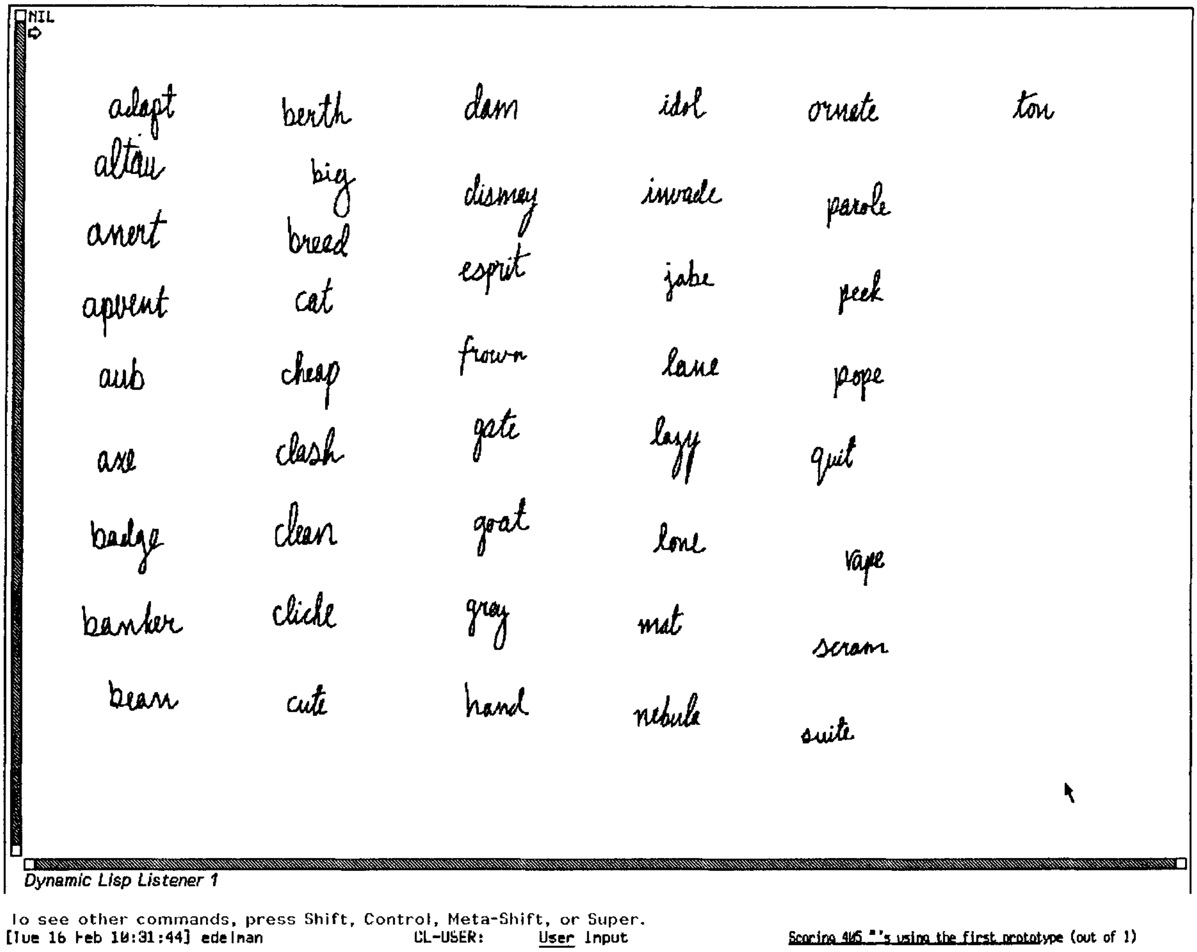
5.2.2 Training. We next tested the stability of the system's performance across writers. It was clear that quite a few new versions of characters would have to be added, mainly because of stylistic variation among the writers. Since in its present form the system was incapable of automatic learning, it had to be trained manually, by parameter adjustment and program rewriting. After studying the image sets B_1 and C_1, the adjustments that seemed to be necessary were carried out. The resulting system was then repeatedly modified, until its performance on the training set E_1 reached a level somewhat below human performance (see the first line of table 3). The main reason to halt at that point was the difficulty to achieve further progress by manual training. The modification was guided mostly by the feedback from the E_1 set, although occasionally individual character experts were modified using images from B_1 and C_1.

5.2.3 Performance Transfer. The system's performance was subsequently tested on sets E_2, E_3, E_4, B_1, and C_1. For each test set, proportion of correct interpretations in the first 1, 5, and 100 candidates, average position of correct interpretation in the outcome list, and proportion of correct lexical interpretation were recorded. The results are summarized in table 3. No system modification whatsoever was carried out between tests; thus, the results indicate the *transfer of performance* across word identity (lines 2 through 4) and across writers (lines 5 and 6).

The system configuration that brought the top-five performance on the training set of 56% yielded 37% and 35% average top-five correct rate across words and across writers, respectively. This corresponds to a performance transfer ratio across words of 0.67 and across writers—0.64 on the average. In absolute terms, the top-five correct rate averaged over the test sets, 36%, is about half as good as average human performance (72%; the best and the worst experimental figures were, respectively, 59% and 86%. See [Edelman 1988], appendix 1).

5.3 Exploring Different Parameter Settings

Several special runs on the image set B_1 were carried out in order to estimate the sensitivity of the results described above to various system parameters. The following



```
To see other commands, press Shift, Control, Meta-Shift, or Super.
[Tue 16 Feb 10:31:44] edelman          CL-USER:     User Input          Scoring 405 "'s using the first prototype (out of 1)
```

Fig. 17. Word image set C_1. The images are reproduced here at about one fourth the resolution actually used by the system.

Table 3. System performance summary. It is more informative to consider the top five outcomes rather than just the best one, as no effort has been invested in distinguishing letters that differ but slightly (such as *n* and *u*, or *g* and *q*). As a result, the top choice often differed from the correct response merely by a substitution in one of these letters. The low figure in line 3, column 4 reflects the performance of the lexical module on set E_3, which contained only pseudowords.

Image Set	Correct First	Correct in Top 5	Correct in Top 100	Correct 1st Using Lexicon	Avg Position of Correct in Top 100
E_1†	30%	56%	93%	81%	13.2
E_2	8%	23%	60%	48%	14.5
E_3	15%	43%	70%	8%	10.1
E_4	22%	41%	56%	50%	4.7
B_1	16%	40%	65%	53%	10.4
C_1	19%	30%	60%	47%	11.4

†The training set.

issues were considered: stroke metrics, string metrics, the use of ligatures, the depth of heuristic search, and the step size in tracing during token extraction.

5.3.1 Influence of Stroke Metrics. The function that measures the goodness of fit between an aligned stroke prototype and the input image includes a multiplicative emphasis operation. The multiplication factor is greater than one for distances that exceed a certain threshold and smaller than one for distances below that threshold. Normally, a threshold value of $w = 5$ pixels was used. Two other values were tested. A run on the image set B_1 with $w = 4$ resulted in a significant deterioration in performance, which dropped to 16%. The average position of the correct interpretation in the results list improved, however, and reached 1.6 (those strings for which the correct interpretation is not found in the top 100 candidates do not contribute to the average position computation; thus, the average position may look good even when the general performance is unacceptable). Setting $w = 6$ brought the performance to 35%, again a deterioration. The default value of $w = 5$ seems therefore to be locally optimal as far as the correct top-five rate is concerned. A smaller value of w would improve the correct first rate, at the cost of a decrease in the correct top-five rate. This behavior of the system signifies that closer alignment is necessary before a stricter goodness of fit measure can be usefully applied.

5.3.2 Influence of String Metrics. The heuristic search for the best interpretation relies on a measure of string goodness that includes two components (see the description of the evaluation function $score_1$ in the previous chapter). The first of these is proportional to the average score of the constituent characters and the second—to the average index of image contour coverage by the string (normally, the weights of the two components of $score_1$ were equal). The test of the influence of component weights consisted of two runs. When the relative weight of contour coverage was decreased by a factor of 5, the top-five correct rate dropped from 40% to 35%. An increase of this weight by a factor of 2 resulted in a similarly degraded correct rate of 37%. It may be assumed therefore that the weights normally used in the tests were (at least locally) optimal.

5.3.3 The Use of Ligatures. The character prototypes used by the system are minimal in the sense that they include no ligatures in either the leading or the trailing position. Thus, even the best interpretation of the image of a fully connected string cannot account for its entire contour, since it is bound to leave the ligatures out. As it happens, the ligatures may constitute a substantial portion of the image, serving as a pool of contour fragments from which spurious letters are constructed. These letters in turn give rise to unwanted alternative interpretations (see, e.g., figure 14).

As we have mentioned above, the set of character prototypes included one that corresponded to a typical ligature rather than to any of the 26 characters. To complement the minimalistic approach to character definition, the system could try to account for as much of the contour as possible, by explicitly detecting the ligatures. This feature was tested, again on the image set B_1, resulting in a reduced correct rate of 35%. This result justifies the minimal approach to prototype definition: apparently, the diversity of the liguature shapes is such that attempting to detect the ligatures explicitly harms rather than helps.

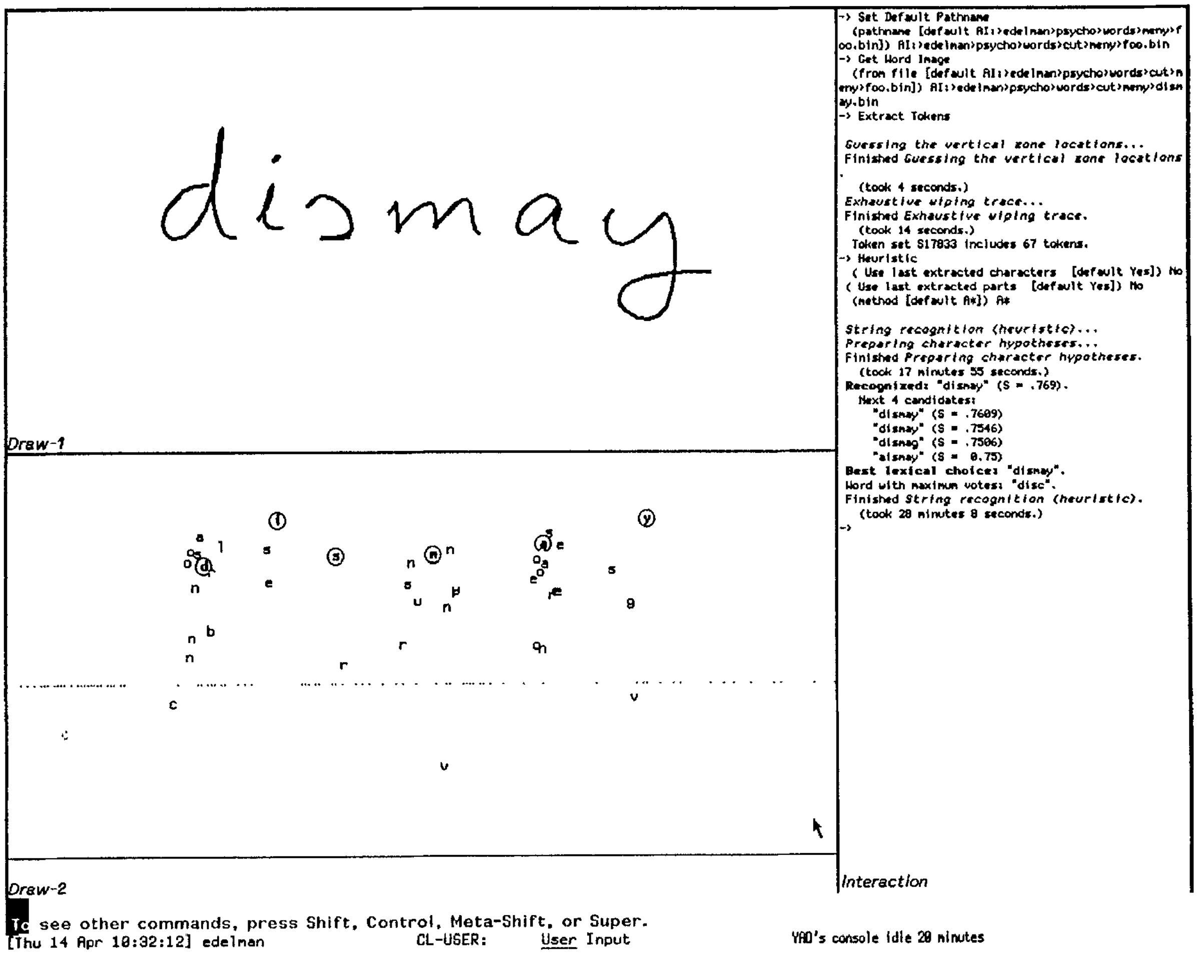
5.3.4 The Depth of Heuristic Search. When the value of the depth parameter of the heuristic search (designated in the description of algorithm BCS in the previous chapter by N) is too small, the system often fails to reach the correct interpretation. The default value, $N = 7500$, represents a compromise between considerations of performance and running time. To find out the influence of this value on performance, the system was run on B_1 with $N = 15000$. The effect of the twofold increase in N was negligible (a 2% increase in the top-five correct rate). Thus, the default value of N seems to be satisfactory.

5.3.5 Tracing Step Size in Token Extraction. Most of the twenty-odd minutes it takes the system to interpret an average string are spent in alignment and evaluation of candidate stroke instances. This time depends on the number of detected tokens (anchor points): the more tokens there are, the more anchor-point combinations must be tried. In fact, the number of combinations grows as n^3 for three-point alignment (see section 2). During system training and testing, the step size in tracing has been equal to 8 pixels (about one tenth the average string height). With this step size, the average number of detected tokens for a five-letter word was about 70. To test the possibility of reducing this number, the system has been run on image set B_1 with step size equal to 10 (a 20% increase). The resulting top-five correct rate decreased by a small amount (just 3%, due to the reduced accuracy in anchor-point positioning), while the average running time per string dropped from 22 minutes to 12. This result indicates that better algorithms for token extraction may reduce the processing time significantly without compromising the performance of the system.

5.4 Difficulties of Reading Cursive Script Reassessed

At the beginning of the article, we suggested that difficulty of segmentation and variability of individual characters are the main factors that aggravate the problem of interpretation of connected strings. We attempted to assess empirically their relative importance and contribution to the error rate exhibited by the system, as follows.

5.4.1 Performance on Discrete-Character Strings. First, we ran the program on image set B_1', obtained from B_1 by erasing inter-character ligatures without affecting



```
-> Set Default Pathname
  (pathname [default AI:>edelman>psycho>words>meny>f
oo.bin]) AI:>edelman>psycho>words>cut>meny>foo.bin
-> Get Word Image
  (from file [default AI:>edelman>psycho>words>cut>m
eny>foo.bin]) AI:>edelman>psycho>words>cut>meny>dism
ay.bin
-> Extract Tokens

 Guessing the vertical zone locations...
 Finished Guessing the vertical zone locations
.
   (took 4 seconds.)
 Exhaustive wiping trace...
 Finished Exhaustive wiping trace.
   (took 14 seconds.)
  Token set S17833 includes 67 tokens.
-> Heuristic
  ( Use last extracted characters  [default Yes]) No
 ( Use last extracted parts  [default Yes]) No
  (method [default A*]) A*

 String recognition (heuristic)...
 Preparing character hypotheses...
 Finished Preparing character hypotheses.
   (took 17 minutes 55 seconds.)
 Recognized: "dismay" (S = .769).
   Next 4 candidates:
     "dismay" (S = .7609)
     "dismay" (S = .7546)
     "dismag" (S = .7506)
     "aismay" (S =  0.75)
 Best lexical choice: "dismay".
 Word with maximum votes: "disc".
 Finished String recognition (heuristic).
   (took 20 minutes 8 seconds.)
->
```

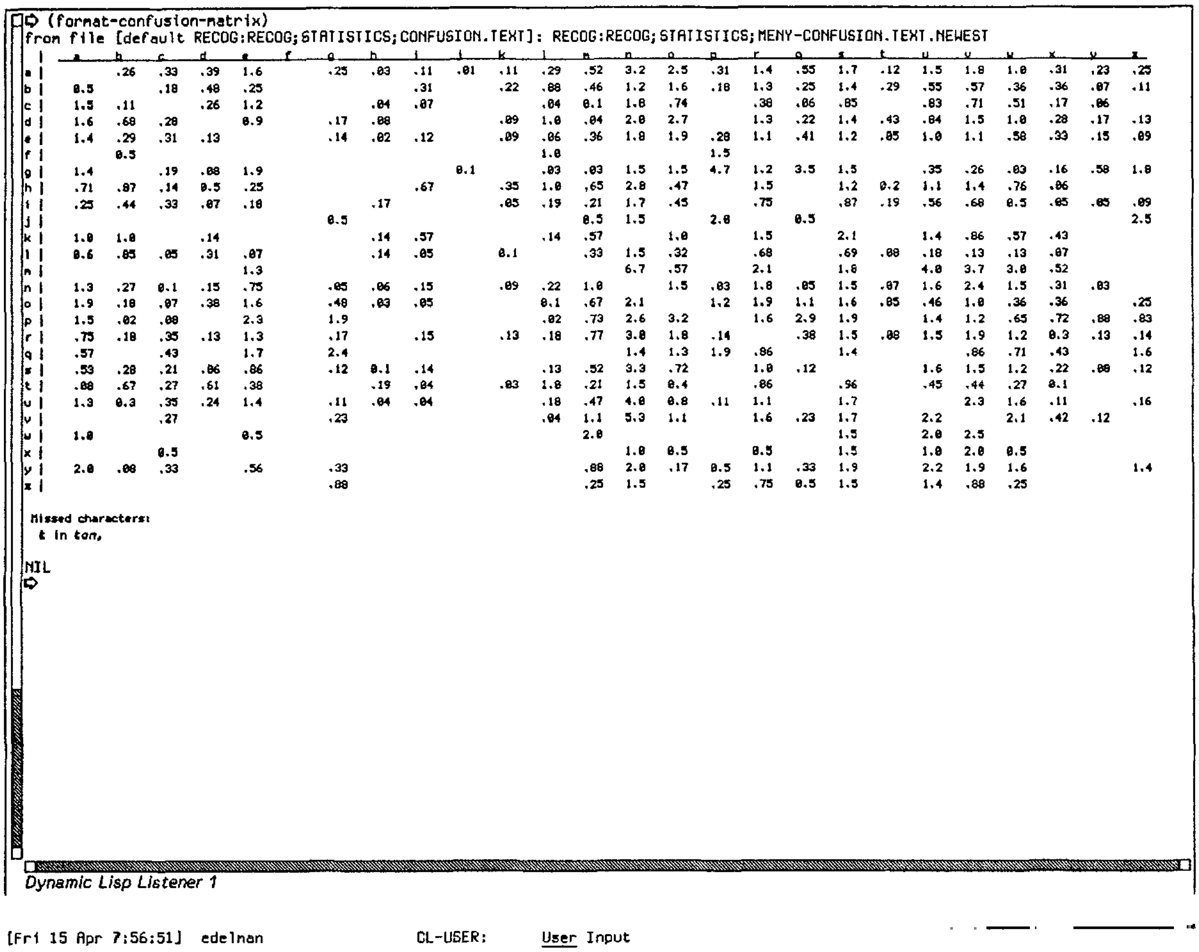
```
To see other commands, press Shift, Control, Meta-Shift, or Super.
[Thu 14 Apr 10:32:12] edelman          CL-USER:     User Input          YAD's console idle 20 minutes
```

Fig. 18. An image of a word from the set B_1', *dismay.* The image was obtained by removing the ligatures between the letters.

character shapes (see figure 18). On this set, the system achieved 33% correct rate for the top interpretation, twice the figure for B_1. In the top five interpretations the correct rate was 47%, exceeding the corresponding figure for B_1 by 7%. The average distance of correct interpretation from the top diminished from 10.4 to 8.7. Thus, disconnecting the characters within strings improved the general performance slightly, whereas the correct rate for the top interpretation increased by a factor of two. The relative stability of the top-five correct rate with respect to changing segmentation conditions shows that the system is to a certain extent capable of segmenting connected strings (see also the next subsection). This capability is an emergent property: there are no explicit segmentation rules in the system. On the other hand, the sharp increase in the correct-first rate brought about by character disconnection indicates that perhaps explicit segmentation heuristics [Srihari and Božinovič 1987; Maier 1986] may be helpful, after all.

5.4.2 Character Confusion Data. Another method of assessing the relative importance of correct segmentation is to calculate the *character* error rate directly. A sufficiently high character error rate may mask the effect of incorrect segmentation and account alone for the observed string errors. To investigate this possibility, the system recorded two types of character error events during the test runs: missed instances and falsely detected ones. The results were presented in the form of confusion matrices. For example, in figure 19 the entry in row *b*, column *a* signifies that in half of the relevant strings an *a* overlapped the detected *b*.[8] The confusion matrix thus provides an estimate of the false detection rate for various characters.

Many of the entries in the confusion matrix of figure 19 are greater than 1. This is due in part to compound characters such as *w* overlapping more than one instance of a simple character such as *v*, and in part to characters that include ligatures. These confounding factors must



```
⇨ (format-confusion-matrix)
from file [default RECOG:RECOG;STATISTICS;CONFUSION.TEXT]: RECOG:RECOG;STATISTICS;MENY-CONFUSION.TEXT.NEWEST
  | a    b    c    d    e    f    g    h    i    j    k    l    m    n    o    p    r    q    s    t    u    v    w    x    y    z
a |      .26  .33  .39  1.6       .25  .03  .11  .01  .11  .29  .52  3.2  2.5  .31  1.4  .55  1.7  .12  1.5  1.8  1.0  .31  .23  .25
b | 0.5       .18  .48  .25                 .31       .22  .88  .46  1.2  1.6  .18  1.3  .25  1.4  .29  .55  .57  .36  .36  .07  .11
c | 1.5  .11       .26  1.2            .04  .07            .04  0.1  1.0  .74       .38  .06  .85       .83  .71  .51  .17  .06
d | 1.6  .68  .28       0.9       .17  .08            .09  1.0  .04  2.0  2.7       1.3  .22  1.4  .43  .84  1.5  1.0  .28  .17  .13
e | 1.4  .29  .31  .13            .14  .02  .12       .09  .06  .36  1.8  1.9  .28  1.1  .41  1.2  .05  1.0  1.1  .58  .33  .15  .09
f |      0.5                                               1.0                 1.5
g | 1.4       .19  .08  1.9                      0.1       .03  .03  1.5  1.5  4.7  1.2  3.5  1.5       .35  .26  .03  .16  .58  1.0
h | .71  .07  .14  0.5  .25                 .67       .35  1.0  .65  2.8  .47       1.5       1.2  0.2  1.1  1.4  .76  .06
i | .25  .44  .33  .07  .10            .17            .05  .19  .21  1.7  .45       .75       .87  .19  .56  .68  0.5  .65  .05  .09
j |                               0.5                           0.5  1.5       2.0       0.5                                     2.5
k | 1.0  1.0       .14                 .14  .57            .14  .57       1.0       1.5       2.1       1.4  .86  .57  .43
l | 0.6  .85  .05  .31  .07            .14  .05       0.1       .33  1.5  .32       .68       .69  .08  .18  .13  .13  .07
m |                     1.3                                          6.7  .57       2.1       1.8       4.0  3.7  3.0  .52
n | 1.3  .27  0.1  .15  .75       .05  .06  .15       .09  .22  1.0       1.5  .03  1.8  .05  1.5  .07  1.6  2.4  1.5  .31  .03
o | 1.9  .10  .07  .38  1.6       .48  .03  .05            0.1  .67  2.1       1.2  1.9  1.1  1.6  .05  .46  1.0  .36  .36       .25
p | 1.5  .02  .08       2.3       1.9                      .02  .73  2.6  3.2       1.6  2.9  1.9       1.4  1.2  .65  .72  .88  .83
r | .75  .10  .35  .13  1.3       .17       .15       .13  .18  .77  3.0  1.8  .14       .38  1.5  .08  1.5  1.9  1.2  0.3  .13  .14
q | .57       .43       1.7       2.4                                1.4  1.3  1.9  .86       1.4            .86  .71  .43       1.6
s | .53  .28  .21  .06  .86       .12  0.1  .14            .13  .52  3.3  .72       1.0  .12            1.6  1.5  1.2  .22  .08  .12
t | .08  .67  .27  .61  .38            .19  .04       .03  1.8  .21  1.5  0.4       .86       .96       .45  .44  .27  0.1
u | 1.3  0.3  .35  .24  1.4       .11  .04  .04            .18  .47  4.0  0.8  .11  1.1       1.7            2.3  1.6  .11       .16
v |           .27                 .23                      .04  1.1  5.3  1.1       1.6  .23  1.7       2.2       2.1  .42  .12
w | 1.0                 0.5                                     2.0                           1.5       2.0  2.5
x |           0.5                                                    1.0  0.5       0.5       1.5       1.0  2.0  0.5
y | 2.0  .08  .33       .56       .33                           .88  2.0  .17  0.5  1.1  .33  1.9       2.2  1.9  1.6            1.4
z |                               .08                           .25  1.5       .25  .75  0.5  1.5       1.4  .88  .25

 Missed characters:
  t in ton,

NIL
⇨
```

```
[Fri 15 Apr 7:56:51]  edelman          CL-USER:          User Input
```

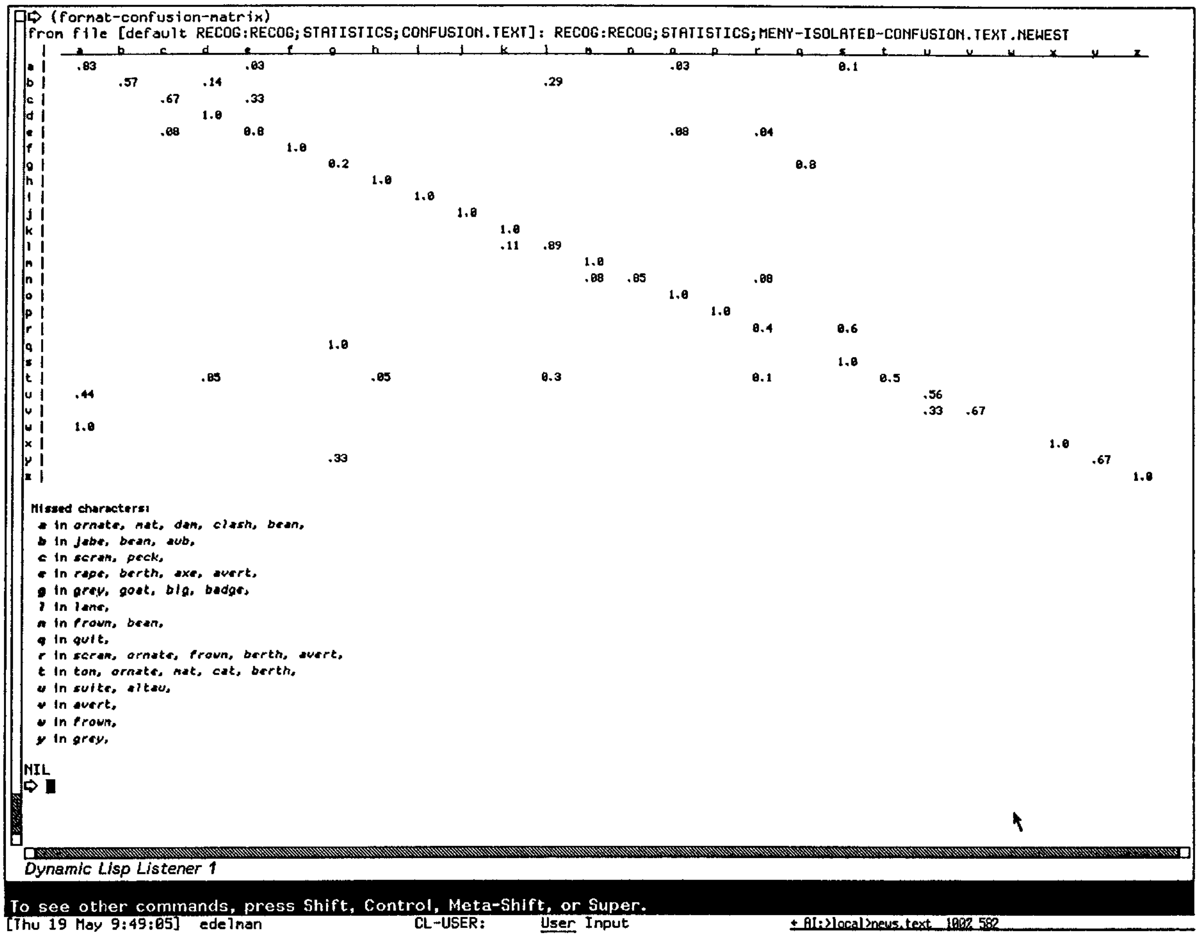
Fig. 19. The confusion table for the image set B_1. The entry in row *b*, column *a* signifies, for example, that in half of the relevant strings an *a* overlapped the detected *b*. The confusion matrix provides an estimate of the false detection rate for various characters.

be removed to permit a faithful estimate of recognition rate for individual characters. Accordingly, the true confusion matrix has been computed using the set B_1 and manual segmentation information. A list of segmentation points (x coordinates) was provided for each one of the 43 images of the set. After detecting the character instances, the system picked for each segmentation slot the best character that wholly resided in it. This character was then compared with the true answer and the corresponding entry in the confusion matrix was updated.

The result appears in figure 20. This matrix is less crowded than the previous one. It should be remembered, however, that while it reflects truthfully the situation in isolated character recognition, the string interpretation system must cope with a flood of character instances like the one that gave rise to the matrix in figure 19.

Note the presence in figure 20 of the diagonal entries that are absent in figure 19. A diagonal entry equal to 1 means that the corresponding character has been always detected correctly. In every case, the sum of the entries along the rows of the matrix is always 1. The 37 misrecognized characters are listed in figure 20 below the matrix. In addition, in five cases it happened that a segmentation slot held no character at all. The estimate of the character recognition rate from this experiment was 78.5% (42 missed characters out of the total of 195). The string recognition rate was 35%. If this figure is compared to the 16% actual correct best rate for B_1 (see table 3), it may be concluded that segmentation errors constitute a considerable proportion of the total made by the system.

5.4.3 Potential Performance. To assess the *potential* of the system, we have modified the above experiment to look in each segmentaion slot for any (not just the best) instance of the true character. Under this condition, the



```
⇨ (format-confusion-matrix)
from file [default RECOG:RECOG;STATISTICS;CONFUSION.TEXT]: RECOG:RECOG;STATISTICS;MENY-ISOLATED-CONFUSION.TEXT.NEWEST
  | a     b     c     d     e     f     g     h     i     j     k     l     m     n     o     p     r     q     s     t     u     v     w     x     y     z
a | .83                     .03                                                         .03                     0.1
b |       .57         .14                                             .29
c |             .67         .33
d |                   1.0
e |             .08         0.8                                                          .08         .04
f |                               1.0
g |                                     0.2                                                                  0.8
h |                                           1.0
i |                                                 1.0
j |                                                       1.0
k |                                                             1.0
l |                                                             .11   .89
m |                                                                         1.0
n |                                                                         .08   .85               .08
o |                                                                                     1.0
p |                                                                                           1.0
r |                                                                                                 0.4         0.6
q |                                     1.0
s |                                                                                                             1.0
t |                   .05                     .05                     0.3                           0.1               0.5
u | .44                                                                                                                     .56
v |                                                                                                                         .33   .67
w | 1.0
x |                                                                                                                                           1.0
y |                                     .33                                                                                                         .67
z |                                                                                                                                                       1.0

 Missed characters:
  a in ornate, mat, dam, clash, bean,
  b in jabe, bean, aub,
  c in scram, peck,
  e in rape, berth, axe, avert,
  g in grey, goat, big, badge,
  l in lane,
  n in frown, bean,
  q in quit,
  r in scram, ornate, frown, berth, avert,
  t in ton, ornate, mat, cat, berth,
  u in suite, altau,
  v in avert,
  w in frown,
  y in grey,

NIL
⇨
```

```
Dynamic Lisp Listener 1
To see other commands, press Shift, Control, Meta-Shift, or Super.
[Thu 19 May 9:49:05]  edelman            CL-USER:        User Input              + AI:>local>news.text  100% 582
```

Fig. 20. The confusion table for image set B_1'.

character recognition rate was 93.3% (13 missed characters out of 195), and the string recognition rate was 70%. In comparison, people recognize correctly 96.8% of handprinted characters [Neisser and Weene 1960], 95.6 % of discretized handwriting [Suen 1983] and about 72% of cursive strings (see [Edelman 1988], appendix 1). The present system appears therefore to have the potential to attain human-like performance in cursive script recognition.

5.5 Possible Improvements

It appears that within the proposed approach a number of improvements can be made. All of these pertain to implementation details rather than to the principle on which the approach is based, namely, the alignment of pictorial descriptions.

5.5.1 Pre-alignment Processing. The present system embodies an effort to obtain maximum performance in cursive string recognition using the alignment approach. Consequently, in its development we have attached secondary importance to issues of information extraction prior to alignment. Experience with the system indicates that this brute-force approach may be unwarranted. Indeed, Ullman ([1986], p. 45) lists three stages in the recognition process that may precede alignment: selection, segmentation and description. In reading, these three stages correspond to isolating a word from its neighbors, segmenting it into characters or strokes and obtaining a suitable description of the regions that will be used for matching character prototypes to the image. Paying more attention to these early processing stages may result in a significant performance improvement.

Consider for instance the process of detection of anchor points (tokens). In the present system, zoning and token extraction take about 20 seconds, which is less than 2% of the total running time per string. They employ straightforward algorithms that are fast but not always reliable. For example, zoning is done by computing the horizontal density histogram of the image (after [Božinović and Srihari 1985]). This technique fails, for example, if the writing baseline is curved, or if most of the string's letters contain ascenders (as in the word *felt*). In the current implementation, a failure at this stage leads to a completely wrong interpretation, because the classification of tokens by zoning is used to prune the correspondence search during alignment. Several such failures were in fact observed in the test runs. If no zoning were used, these failures would not happen (at the expense of longer correspondence time).

Currently, the dependency of the complexity of correspondence computation on the number of image tokens is cubic (see section 2). A better scheme for token classification could improve this situation considerably [Huttenlocher and Ullman 1987]. For example, sign and magnitude of contour curvature may be used to label tokens. Ultimately, token labeling could result in a linear dependency of correspondence complexity on the number of tokens.

5.5.2 Better Metrics. At present, the degree of fit between a transformed model and the image is evaluated essentially by computing for each model point the distance to its nearest neighbor point in the image and taking the maximum over all model points. Some potential problems with this approach were discussed above. A better degree of fit evaluation function (e.g., one that is sensitive to orientation as well as position discrepancies between model and image contours) can reduce the number of false instances and alleviate the interpretation ambiguity.

5.5.3 Post-alignment Adjustment. Experience with the present system shows that most, but not all, of the character shape variability can be compensated for by the affine transform used in the alignment of character prototypes. The remaining variability seems to be small enough to be removed by local methods. In other words, an *adjustment stage* could be inserted after the initial alignment. One such local method is Burr's elastic matching [Burr 1981, 1983]. In elastic matching, the image or the prototype (or both) is modeled as elastic membranes that can deform under the influence of distorting forces. In Burr's work these forces were provided by attraction between nearest neighbors in the image and the model. According to Burr ([1981], p. 102), coarse registration is desired initially, to improve the chances that the nearest-neighbor relation captures the true correspondence between model and image. Thus, alignment followed by elastic adjustment could reduce the post-alignment discrepancies, resulting in a smaller number of missed character instances.

6 Discussion

6.1 Comparison with Previous Work

It is hard to compare our results directly with those of other researchers, since all previously implemented systems depended on lexical knowledge. An indirect comparison is possible, if the lexicalized results are considered. The lexical module employed by the present system works independently of the visual recognition module and uses a 30,000-word dictionary. The average lexicalized recognition rate for legal words is 50%. This may be compared with 77%, 29% and 32% achieved by the system of Srihari and Božinović [1987] for three different writers, without retraining and using a small, 700-word lexicon (when a 7800-word lexicon was used, the performance for the first writer dropped to 48%; the other two writers were not tested in that condition). Thus, the present system performs better than its predecessors when using lexical knowledge. In addition, it demonstrates a substantial capability to interpret cursive script by a purely visual method, without recourse to a lexicon.

6.2 On the Use of Motor Knowledge in Recognition

The system described here uses alignment by affine transformation to remove most of the variability of handwritten shapes. Our choice of anchor points for alignment is motivated by several empirically determined features of handwriting:

— Handwritten trajectories can be approximated by polynomial expressions whose coefficients are determined by a small number of control points using an optimization principle.
— The locations of the control points for a given trajectory can be inferred visually from its shape.
— The six-parameter affine transform can capture the variability of stroke shapes across different letters produced by the same writer, as well as across writers.

Putting motor knowledge to use in recognition was first proposed by Halle and Stevens [1962], who called it *analysis by synthesis* (ABS). In analysis by synthesis, letters are recognized through their motor programs, which are deduced by guessing an initial program and iteratively updating it, using the difference between the synthesized shape and the actual one as feedback [Yoshida and Eden 1973]. Thus, ABS actively uses motor knowledge during recognition. In contrast, our approach uses motor knowledge indirectly, in the extraction and use of the anchor points. Psychological findings (reviewed in [Edelman 1988]) support our approach by indicating that while people may use motor knowledge in reading, they do not seem to do so by mentally reproducing the process of writing.

Finally, we mention the possibility that in humans a common representation of contours is involved in planning arm movements and in processing visual information in tasks such as reading cursive script and detecting salient curves in images. Computationally, both kinds of processes can be described in terms of optimization [Edelman and Flash 1987; Sha'ashua and Ullman 1988]. The co-occurrence of certain kinds of acquired dysgraphia and dyslexia (see [Edelman 1988] for a discussion) may also be interpreted as supporting this conjecture.

References

Aho, A.V., Hopcroft, J.E., and Ullman, J.D. 1974. *Design and Analysis of Computer Algorithms*. Addison-Wesley: Reading, MA.

Ben-Israel, A., and Greville, T.N.E. 1974. *Generalized Inverses: Theory and Applications*. Wiley: New York.

Biederman, I. 1985. Human image interpretation: recent experiments and a theory. *Comput. Vision, Graphics, Image Process.* 32: 1–47.

Bookstein, F.L. 1978. *The measurement of biological shape and shape change*, Lecture Notes in Biomathematics, 24, S. Levin, ed., Springer-Verlag: Berlin.

Božinović, R.M., and Srihari, S.N. 1985. ROCS: a system for reading off-line cursive script. SUNY at Buffalo TR-85-13, September 1985.

Burr, D.J. 1981. Elastic matching of line drawings. *IEEE Trans. Patt. Anal. Mach. Intell. PAMI* 3: 708–713.

Burr, D.J. 1983. Matching elastic templates, 260–270. In *Physical and Biological Processing of Images*, O.J. Braddick and A.C. Sleigh (eds.), Springer-Verlag: Berlin.

Chen, S., and Penna, M. 1986. Shape and motion of nonrigid bodies. *Comput. Vision, Graphics, Image Process.* 36: 175–207.

de Boor, C., and Lynch, R.E. 1966. On splines and their minimum properties. *J. Math. Mech.* 15: 953–969.

Duda, R.O., and Hart, P.E. 1973. *Pattern Classification and Scene Analysis*. Wiley: New York.

Duvernoy, J., and Charraut, D. 1979. Stability and stationarity of cursive handwriting. *Pattern Recognition* 11: 145–154.

Edelman, S. 1988. Reading and writing cursive script: A computational study. Ph.D. thesis, Dept. of Applied Math., Weizmann Institute of Science.

Edelman, S., and Flash, T. 1987. A model of handwriting. *Biological Cybernetics* 57: 25–36.

Eden, M. 1961. On the formalization of handwriting, *Proc. Symp. Appl. Math.* 12: 83–88, Amer. Math. Soc., Providence, RI.

Flash, T. 1983. Organizing principles underlying the formation of arm trajectories. Ph.D. thesis, Harvard MIT Div. of Health Sciences and Technology.

Flash, T., and Hogan, N. 1985. The coordination of arm movements: An experimentally confirmed mathematical model, *Journal of Neuroscience* 5 (7): 1688–1703.

Foster, D.H. 1975. An approach to the analysis of the underlying structure of visual space using a generalized notion of visual pattern recognition. *Biological Cybernetics* 17: 77–79.

Goad, C. 1986. Fast 3-D model-based vision, 371–391. In *From Pixels to Predicates*. A.P. Pentland (ed.), Ablex: Norwood, NJ.

Grimson, W.E.L. 1981. *From Images to Surfaces: A Computational Study of the Human Early Visual System*. MIT Press: Cambridge, MA.

Grimson, W.E.L., and Lozano-Perez, T. 1987. Locating overlapping parts by searching the interpretation tree, *IEEE Trans. Patt. Anal. Mach. Intell. PAMI* 9: 469–482.

Halle, M., and Stevens, K. 1962. Speech recognition: a model and a program for research. *Inst. Rad. Eng. Trans.* IT 8: 155–159.

Hanson, A.R., Riseman, E.M., and Fisher, E. 1976. Context in word recognition. *Pattern Recognition* 8: 35–45.

Hayes, K.C., Jr. 1980. Reading handwritten words using hierarchical relaxation. *Comput. Graphics, Image Process* 14: 344–364.

Hogan, N. 1982. Control and coordination of voluntary arm movements. In *Proc. 1982 Amer. Control Conf.*, M.J. Rabins and Y. Bar-Shalom, (eds.), pp. 522–528.

Hogan, N. 1984. An organizing principle for a class of voluntary movements, *Journal of Neuroscience* 4 (11): 2745–2754.

Huttenlocher, D.P., and Ullman, S. 1987. Object recognition using alignment, *Proc. 1st Intern. Conf. Comput. Vision*, London, pp. 102–111.

Kahan, S., Pavlidis, T., and Baird, H.S. 1987. On the recognition of printed characters of any font and size, *IEEE Trans. Patt. Anal. Mach. Intell. PAMI* 9: 274–287.

Karlin, S. 1969. The fundamental theorem of algebra for monosplines satisfying certain boundary conditions and applictions to optimal quadrature formulas. In *Approximations, with Special Emphasis on Spline Functions*. I.J [illegible] berg (ed.), Academic Press: New York, pp. 467–484.

Lowe, D. 1986. *Per* [illegible] *tion and Visual Recognition*. Kluwer: Boston, [illegible]

Maier, M. 1986. Se [illegible] n scripted documents. *Proc. 8th Intern. Con* [illegible] pp. 1056–1058.

Marr, D. 1982. *V* [illegible] Francisco, CA.

McClelland, J.L., [illegible] 981. An interactive activation model of contex [illegible] eption: Part 1. An account of basic finding [illegible] *ew* 88: 375–407.

Neisser, U., and W [illegible] ote on human recognition of hand-printed char [illegible] *trol* 3: 191–196.

Okuda, T., Tanaka, E. [illegible]. 1976. A method for the correction of garbled words u [illegible] e Levenshtein metric, *IEEE Trans. Computers* C-25: 172–1 [illegible]

Paivio, A. 1978. The relationship between verbal and perceptual codes. *Handbook of Perception*, E.C. Carterette and M.P. Friedman (eds.), Academic Press: New York, vol. 8, pp. 375–397.

Pavlidis, T. 1977. *Structural Pattern Recognition*. Springer-Verlag, Berlin.

Pearl, J. 1984. *Heuristics*. Addison-Wesley: Reading, MA.

Persoon, E., and Fu, K.S. 1977. Shape discrimination using Fourier descriptors. *IEEE Trans. Syst. Man Cybern. SMC* 7: 170–179.

Ritter, K. 1969. Generalized spline interpolation and nonlinear programming. In *Approximations, with Special Emphasis on Spline Functions*, I.J. Schoenberg (ed.), Academic Press: New York, pp. 75–117.

Serra, J. 1982. Image Analysis and Mathematical Morphology. Academic Press: New York.

Sha'ashua, A., and Ullman, S. 1988. Structural saliency: The detection of globally salient structures using a locally connected network. *Proc. 2nd Intern. Conf. Comput. Vision*, Tampa, FL, pp. 321-327.

Srihari, S.N., and Božinović, R.M. 1987. A multi-level perception approach to reading cursive script. *Artificial Intelligence* 33: 217-255.

Stentiford, F.W.M. 1985. Automatic feature design for optical character recognition using an evolutionary search procedure, *IEEE Trans. Patt. Anal. Mach. Intell. PAMI* 7: 349–355.

Suen, C.Y. 1983. Handwriting generation, perception and recognition. *Acta Psychologica* 54: 295–312.

Suen, C.Y. Berthod, M., and Mori, S. 1980. Automatic recognition of handprinted characters—The state of the art. *Proc. IEEE* 68: 469–487.

Torre, V., and Poggio, T.A. 1986. On edge detection. *IEEE Trans. Patt. Anal. Mach. Intell. PAMI* 8 (2): 147-163.

Travers, J.R., and Olivier, D.C. 1978. Pronounceability and statistical "Englishness" as determinants of letter identification. *Amer. J. Psychol.* 91: 523–538.

Ullman, S. 1986. An approach to object recognition: aligning pictorial descriptions. MIT AI Memo 931, December 1986.

Yoshida, M., and Eden, M. 1973. Handwritten Chinese character recognition by an analysis-by-synthesis method, *Proc. 1st Conf. Patt. Recog.* pp. 197-204.

Zwikker, C. 1963. The advanced geometry of plane curves and their applications. Dover: New York.

Notes

[1] Our result may be compared to that of Duvernoy and Charraut [1979], who found that the first five factors of a Karhunen-Loewe expansion suffice to capture the variation in the shape of the letter *a* illustrated in figure 2.

[2] Stentiford [1985] reported a character recognition system that uses 316 features.

[3] The relation *above* is not invariant under affine transformations, a property that should theoretically preclude its use in this context. The phenomenon of perceptual upright (see [Edelman 1988] for a discussion) indicates, however, that people too use such relations in character recognition.

[4] The use of the token coverage here partially compensates for the asymmetry of the distance function at the stroke level. The lost symmetry can be reintroduced at this stage, because the problem of interference due to the neighboring letters is nonexistent at the string level.

[5] Obviously, "correct" here can mean only "being in an agreement with the majority of opinions." This notion of correctness by convention is even more important when it is known that nonsense strings are allowed. For example, in figure 14 the writer may well have intended to write *adeyst*, in which case the system's first choice interpretation would be correct.

[6] The original spell-checking function returns for a given string all legal words that can be obtained from it by a deletion, insertion or substitution of a single letter, or by transposition of two adjacent letters. It has been modified to disallow letter transposition, which is a common typing error that is not relevant to the present context.

[7] In the present system, no effort has been invested in distinguishing between highly similar letters (e.g., *e* and *c*, or *g* and *q*). Consequently, the top choice often differs from the correct interpretation by a substitution of one of these letters by its look-alike, making the consideration of the top five returned interpretations more informative than just the best one.

[8] Note that the confusion relation between characters is asymmetric. A string is defined as relevant for the purpose of computing the confusion between c_1 and c_2 (that is, false detection of c_1 in c_2) if it contains c_2 but not c_1. The last requirement is obligatory in cursive strings because of position indeterminacy of character instances: a rightly detected c_1 may accidentally overlap c_2, distorting the confusion estimate. This phenomenon does not exist if the characters are presented one at a time (e.g., [Neisser and Weene 1960; Suen 1983]).