A SYNTACTIC OMNI-FONT CHARACTER RECOGNITION SYSTEM

George Wolberg

Department of Computer Science
Columbia University
New York, NY 10027
wolberg@cs.columbia.edu

March 1987
Technical Report CUCS-298-87

ABSTRACT

This paper introduces a syntactic omni-font character recognition system. The "omni-font" attribute reflects the wide range of fonts that fall within the class of characters that can be recognized. This includes hand-printed characters as well.

A structural pattern-matching approach is employed. Essentially, a set of loosely constrained rules specify pattern components and their interrelationships. The robustness of the system is derived from the orthogonal set of pattern descriptors, location functions, and the manner in which they are combined to exploit the topological structure of characters.

By virtue of the new pattern description language, PDL, developed in this paper, the user may easily write rules to define new patterns for the system to recognize. The system also features scale-invariance and user-definable sensitivity to tilt orientation.

1. INTRODUCTION

Optical character recognition (OCR) has been the subject of intensive research efforts for roughly twenty-five years. The immense activity drawn to OCR is a testimony to its challenge and practicality. Clearly there is much motivation to provide automated text and data entry into computerized systems. In fact, a solution to process large volumes of data automatically would resolve the interesting irony which currently exists in today's Information Age: in the midst of all the dramatic advances in computer technology, virtually no advancements have been made in data entry, the most serious bottleneck in data processing.

In all this time, conventional OCR systems have never overcome their inability to read more than a handful of type fonts and page formats. Proportionally spaced type (which include virtually all typeset documents), and even most non-proportional typewriter fonts, have remained beyond the reach of these systems. As a result, conventional OCR has never achieved more than a marginal impact on the total number of documents needing conversion into digital form.

This paper describes a syntactic omni-font character recognition system. The goal of this system is to recognize isolated (discrete) machine or hand-printed characters that a human would be expected to identify, in the absence of contextual information. This requires the system to be scale-invariant and immune to reasonable tilt. By designing a system that yields maximal results without context, we are succeeding in working towards a system which is superior once context will be supplied. Applied over the 26 uppercase characters of the Roman alphabet the system has achieved a 95.2% recognition rate.

Syntactic OCR has been investigated by many researchers. Many systems have been devised that extract features such as the number of line endings, loops, T-junctions, L-junctions, etc. [6], [7]. A more traditional approach is to extract primitives from the input characters. The primitives are generally line segments which comprise a polygonal approximation of a character [1], [13], [18]. The segments are found by detecting high curvature points [17]. In [5], the region occupancy representation of characters is investigated. There, curves define sectors of a circle, and straight lines cut regions into subregions. Considerations of the intersections between such subregions yields their descriptions of characters. An excellent survey is found in [8].

2. OVERVIEW OF THE RECOGNITION SYSTEM

A quick glance of the system's approach is now presented. The text is inputted and transformed into a binary image. Typically, a video camera, scanner, or digitizing tablet is used for image acquisition. In this implementation, the latter device was used since it presented fewer noise problems and it assured a binary image.

The digitized page of text first passes through a thinning stage. This serves to erode the characters to their skeletons (stick figures) and thereby achieve data reduction. Each input character then undergoes a polygonal approximation which serves to map all skeleton segments onto a finite alphabet of descriptors. The eight descriptors chosen for the alphabet comprise a character basis set. Their selection, and the heart of the method for that matter, is based on the treatment of the question: what are the barest minimal components needed to recognize any given character? The virtue of this method is that the character descriptors and their coupling encapsulates the level of abstraction necessary to recognize the characters in the presence of noise. Proper consideration requires us to examine the effects of noise (i.e. foreshortened or elongated strokes, and gaps) on the ambiguity in character classification. This design approach runs parallel to those proposed in [3] and [15]. Their approaches were based on the phenomenological attributes of characters. They advocated that a theory based on ambiguities, rather than the archetypical shape of letters, leads to algorithms which will perform more accurately.

The system described in this paper is driven by a loosely constrained set of description rules that make use of the small alphabet of primitive descriptors and another set of functions which specify their spatial interrelationships. Furthermore, the presence of extraneous ornamentation (as in calligraphy) does not hinder performance since they are not included in the rules which seek out the critical components [16].

A hierarchy of processing stages is established to work on the incoming binary image that comprises the digitized page of text. They include: thinning, skeleton tracing, merging, description generation, parsing, and classification. A block diagram of the character recognition system is shown in Figure 1. The sequel is devoted to the elaboration of these modules.

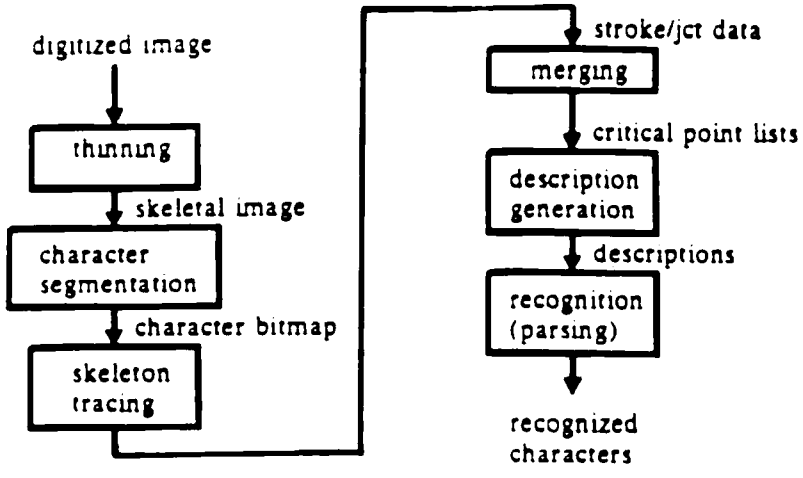

Figure 1: Character recognition subsystems.

3. THINNING

The thinning algorithm implemented in this work is similar to that described in [11]. Other common thinning algorithms may be found in [2], [4], [10], [14]. Figure 2, shown below, illustrates the effect of thinning on text. Notice that dashes ('-') represent pixels which were stripped away, and numbers correspond to the skeletal pixels. The values of these numbers refer to the number of iterations needed before the algorithm was able to label these pixels as skeletal.

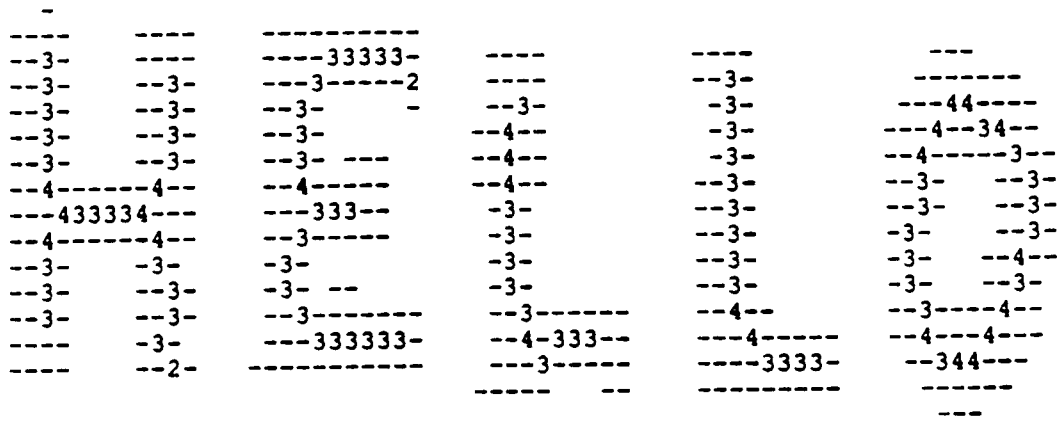

Figure 2: Thinned text.

4. SKELETON TRACING

Skeleton tracing is the first stage of the feature extraction component of this system. The information extracted in this stage includes the registration of all strokes and junctions, and the calculation of the strokes' critical points. Now that our characters are represented with lines of unity thickness, tracing of the strokes, or skeletons, becomes a fairly simple task.

4.1 Definitions

A *stroke* is defined as any segment of the skeleton that is terminated on either end by an endpoint or a junction. A *junction* corresponds to the intersection of two or more strokes. Figure 3 depicts the letter 'A' on a discrete grid. Notice that there are 6 strokes and two junctions.

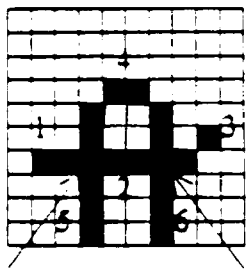

Figure 3: A digitized letter with six strokes and two junctions.

Tracing the skeletons to retrieve strokes would be rather useless unless we extracted shape information from them. With the many pixels that may lie on a stroke, a representation is needed to more readily and compactly capture the detail of the stroke. The identification of *critical points* is used for this purpose. Critical points are defined as points of high local curvature, or terminal points. The latter refers to the endpoints or junctions that delimit the stroke. Figure 4 highlights the critical points of the letter 'A' given in Fig. 3. Notice how these few points relay the basic shape information of the stroke. Of the critical points shown, only the topmost corresponds to local maximum curvature. The remainder are associated with either endpoints, or junction points.

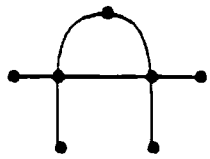

Figure 4: Highlighted critical points.

The purpose of collecting these critical points is to compress the shape information contained in the strokes. The important assumption made here is that we may extract the necessary features (defined later) from a first order polygonal approximation made on the original characters. Ultimately, we represent the bulk of information contained in the many character pixels in only these handful of critical points.

During the traversal of each stroke, measurements are made at regular intervals to find its critical points. To simplify calculations, the measurements test to see whether the slope of the current interval has changed sign. The actual size of these intervals is dependent on the size of the character, or equivalently, the coarsest resolution permissible before recognition degradation sets in. The interval size, *SMPL*, is known as the sampling distance. Although this parameter is subject to fine tuning, *SMPL* was generally set to be 1/10 of the character height.

It is common to have gaps present in the strokes. If the size of the gaps are less than *SMPL*, the sampling distance, and the gap is at the convergent ends of the strokes, then they may be considered noise and bridged. For example, consider the question of joining endpoints $E1$ and $E2$ in Fig. 5.

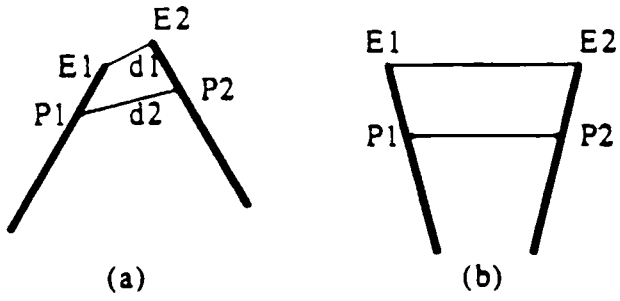

Figure 5: (a) Converging and (b) diverging strokes.

Points $P1$ and $P2$ lie *SMPL* pixels away from the endpoints on their respective strokes. Having labeled these four points, we perform two sets of calculations. We define

$$d1 = [\, E1_x - E2_x \,]^2 + [\, E1_y - E2_y \,]^2 \tag{4.1}$$

$$d2 = [\, P1_x - P2_x \,]^2 + [\, P1_y - P2_y \,]^2 \tag{4.2}$$

Notice that it is desirable to cross gaps between strokes which are converging at their endpoints, as in Fig. 5a. This is typified by d2 greater than d1. On the other hand, strokes which are diverging are typified by d2 less than d1 and are thereby not bridged. This instance is depicted in Fig. 5b. Crossing a gap is marked by the registration of the two endpoints as critical points of the same stroke. Once a gap is crossed, the traversal process continues as before.

This module has therefore changed the pixel representation of the image to a list representation. This new form consists of lists of critical points for the strokes and junctions. Critical points are used to generate a more compact stroke representation that conveys shape information.

5. MERGING

This stage of the system is responsible for refining the data extracted from the skeleton tracing module. In skeleton tracing, junctions, as well as endpoints, defined the endpoints of strokes. It is the goal of the merging stage to reach beyond the localness of the extracted data and combine them to yield more meaningful global information. Stated alternately, the goal of merging is to combine together strokes which were meant to be together in the first place but were broken into pieces at the junctions. Figure 6a illustrates an example in which a vertical line was intersected by two horizontal bars. Merging will combine the seven strokes into the original three, as shown in Fig. 6b.

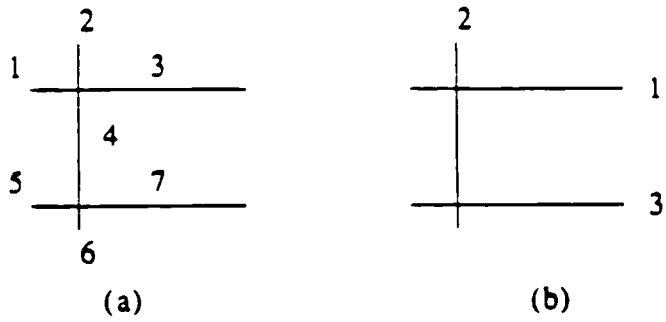

Figure 6: (a) Seven strokes are present before merging. (b) After merging, three nstrokes are formed.

The merging stage consists of two phases. The first phase joins the strokes supplied from skeleton tracing into larger chains of *new strokes*, called *nstrokes*. At each junction point, we therefore attempt to link together pairs of strokes which are approximately collinear. Stated alternately, we merge those strokes which combine to yield the best candidate for straight lines. In Fig. 6a, for example, merging has resulted in the joining of strokes (1,3), (2,4,6), and (5,7). This is in accord with the good continuity rule of Gestalt psychology [5]. Consequently, the noise introduced at the intersections by thinning is filtered out.

In the second phase, we seek to combine nstrokes together if they overlap trivially. Here it is implied that lines were meant to be continuous eventhough they overlap slightly. A "trivial" overlap, in this system, is defined to occur when less than a third of an nstroke extends past a junction to an endpoint. In Fig. 6b, for example, the resulting three nstrokes can be considered to trivially overlap: nstroke 1 with nstroke 2, and likewise for nstrokes 2 and 3. As a result, the 3 nstrokes are reduced to one nstroke which now consists of those elements which had been strokes 3, 4, and 7 in Fig. 6a. Therefore, the pattern shown in Fig. 6b can be described by a single curve rather than the intersection of three lines. This agrees with the model of the alphabet presented in Section 6. When all the *nstrokes* have been tested, we have a set of critical point lists that have had the noise filtered out. They now embody the most global shape information of the drawn lines in the character.

6. PATTERN DESCRIPTION LANGUAGE, PDL

At the heart of any pattern recognition system is a pattern modeling and description mechanism. The modeling takes the form of a set of simple geometric *primitives* which are mapped onto the input characters. Primitives are basic components of the patterns which are used as building blocks to compose a pattern. The new *pattern description language* (PDL) developed in this paper allows us to compactly describe the mapping, or composition, in detail. The PDL presented here makes use of simple primitives which are easy to extract yet meaningful to the task-domain. Furthermore, the language syntax provides flexibility for the addition of new rules.

The PDL described here is similar to the Picture Description Language introduced by Shaw [12]. The major differences lie in the broader manner in which the primitives may be coupled. Whereas Shaw's PDL supports only tail/head connectivity, the PDL described here permits arbitrary connectivity of two primitives. It is more closely related to the method described in [9].

6.1 Selection of Character Primitives

In the context of character recognition, character primitive selection requires us to determine the members of the character basis set. The results of the selection process are shown in Fig. 7. Notice that the core of the basis set are four concavities pointing up, down, left, and right. They are denoted as *CU*, *CD*, *CL*, and *CR*, respectively. The direction assigned to a concavity corresponds to the side of the opening. Note that all concavities can be defined by three critical points: one vertex and two endpoints. The remaining primitives are line segments oriented in four directions: horizontal, vertical, and the two diagonal orientations. They are denoted as *HOR*, *VERT*, *NEDIAG*, and *NWDIAG*, respectively.

The ability to specify spatial interrelationships of primitives is given by a set of *location functions*. They permit us to define ranges of points along primitives. Briefly, a stroke is divided into a small number of regions (less than five). The extent of the regions is dependent on the total length of the stroke. It is therefore a simple matter to state, for example, that the *ctr* part of a *CR* must meet with the *top* part of a *CL* to form the letter 'S'. Notice that the names of the functions allow for a meaningful description of patterns. Furthermore, the relationships may be specified in any order. The function names include *top*, *bottom*, *ctr*, *left*, *right*, *lctr* (slightly *l*eft of center), *rctr* (*r*ight of center), *tctr* (*t*op of center), and *bctr* (*b*ottom of center). Their graphical correspondence is shown in Fig. 7.

There is no size information associated with primitives. This allows us to avoid a size normalization preprocessing stage common in many systems. The only constraint on the concavities is that their two endpoints be roughly the same distance from the vertex. If the ratio of the two chords connecting the endpoints to the vertex deviates greater than a given tolerance level from 1, then we may consider the concavity to be appended to another primitive. An example is demonstrated by the letters 'U' and 'J'. Clearly 'U' is just concave up, while 'J' is concave up that meets at its right endpoint with the bottom of a vertical line. It is the goal of a language to compactly relay this information to the recognition system. The description of the language

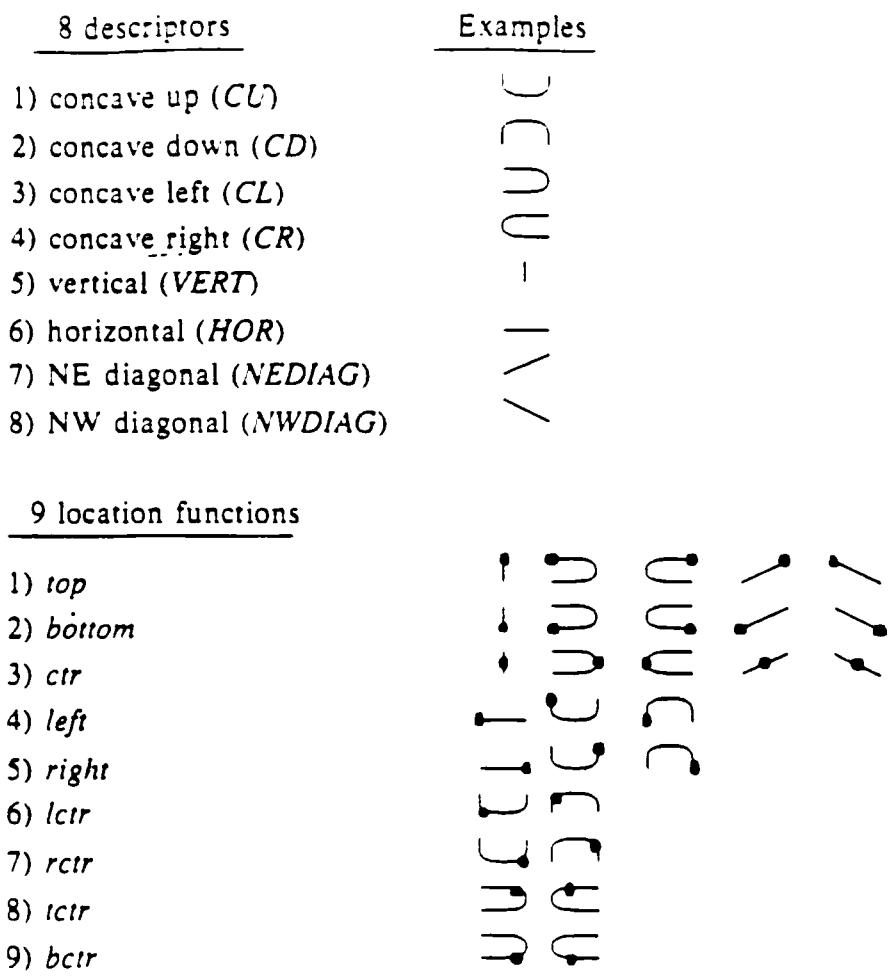

Figure 7: Character basis set: eight descriptors and nine location functions.

syntax will now be discussed.

6.2 PDL Syntax

In order to convey the description of the entire alphabet to the recognition system, a mechanism must be devised to compactly describe characters. The language proposed in this paper allows the user to describe a rich set of patterns with a straightforward syntax and a visually meaningful set of descriptors and location functions. There are two forms which description rules can take: chain and tree form.

6.2.1 Description Rules: Chain Form

Let us begin by demonstrating how two primitives may be combined. We will refer to the two primitives as $p1$ and $p2$. Their respective location functions will be refered to as $f1$ and $f2$. In general, all primitives and location functions will have a 'p' and 'f' prefix, respectively. If *name* is the name of the character comprised of these two primitives, then the description rule will look like:

name, $p1$ ($f1$, $f2$) $p2$

The above form is known as the *chain* form. It is read as follows. The $f1$ region of $p1$ meets with the $f2$ region of $p2$. We may add another primitive to be connected to $p2$ by adding the necessary location functions, and the name of the third primitive, as follows.

name, $p1$ ($f1$, $f2$) $p2$ ($f3$, $f4$) $p3$

Notice that $f3$ is applied to $p2$ and $f4$ is applied to $p3$. This template of primitives separated by a pair of location functions can be extended to any chain of connected primitives.

A large class of characters are described this way. The letter 'S', for example, may be described as either of the following:

S, *CR* (*ctr*, *top*) *CL*
S, *CR* (*bottom*, *ctr*) *CL*

Notice that the three location functions used here segment the primitive into three regions: top, center, and bottom. Furthermore, specifying primitive composition is easy and intuitively meaningful. Also, the rules can be expressed in any order.

6.2.2 Description Rules: Tree Form

Another compact representation is sometimes possible with the tree form. This compaction arises when several connections may be made on a single primitive. The implication that a primitive may have more than one dependency with another primitive suggests that a distributive rule analog may be invoked. This is the contrasting feature to the chain form which has no such mechanism.

The distributive property is implemented by adding extra levels of parentheses. If $p1$ is connected to both $p2$ and $p3$, the rule will look like this:

name, $p1$ (($f1$, $f2$) $p2$) ($f3$, $f4$) $p3$

Notice that the extra level of parentheses has caused $p1$ to be applied over $p2$ and $p3$. The location function of $p1$ is $f1$ when considering $p2$, and $f3$ when considering $p3$. Any number of operations may be made on a single primitive by adding the appropriate number of parentheses. This syntax is known as the tree form. The name is attributed to the rather complex patterns that can be described due to the facility to embed chains within chains. It allows for both tree and chain forms to be intermixed.

The more complex class of characters are handled using this approach. Some examples appear in Sec. 6.4.

6.2.3 Multi-Primitive Usage

It is often common when generating description rules to allow for a choice of primitives to be connected to a given component at a given location. For example, we may allow J to be described as a *CU* primitive to meet at its right endpoint with the bottom of *either* a *VERT* or *NEDIAG* line. The facility in the language that allows for this choice is given by square brackets to contain the choices and '/' to separate them. In the example just provided, the description rule for 'J' is given as:

J, CU (right, bottom) [VERT / NEDIAG]

6.3 Description Rule Grammar

The following syntax will generate any sentence (description rule) *S* in PDL.

S → *CHAR ',' ATRB DSCRLIST*

DSCRLIST → *DSCRLIST DSCR*
→ ε

DSCR → *'(' FCT ',' FCT ')' ATRBS*
→ *'(' DSCRLIST DSCR ')' DSCR*

ATRBS → *ATRB*
→ *'[' ATRB ATRBLIST ']'*

ATRBLIST → *ATRBLIST '/' ATRB*

CHAR → *'A'*
→ *'B'*
→ *'C'*
. . .
→ *'Z'*

ATRB → *CU*
→ *CD*
→ *CL*
→ *CR*
→ *HOR*
→ *VERT*
→ *NEDIAG*
→ *NWDIAG*

FCT → *top*
→ *bottom*
→ *left*
→ *right*
→ *tctr*
→ *bctr*
→ *lctr*
→ *rctr*

Note that all characters in single quotes, as well as the descriptors and location functions,

are terminals.

6.4 Character Description Rules

A partial list of descriptions for the alphabet of characters is given below.

A, CD ((lctr, left)[HOR/NEDIAG/NWDIAG]) (rctr, right)[HOR/NEDIAG/NWDIAG]
B, VERT ((((top,top)CL) (ctr,bottom)CL) (ctr,top)CL) (bottom,bottom)CL
D, VERT ((top,top)CL) (bottom,bottom)CL
E, CR (ctr, left)HOR
F, VERT ((top,left)HOR) (ctr,left)HOR
H, VERT (ctr,left)HOR (right,ctr)VERT
K, VERT ((ctr,bottom)NEDIAG) (ctr,top)NWDIAG
L, VERT (bottom,left)HOR
M, CD (right, left)CD
M, CD (ctr,left)CU (ctr,ctr)CD
N, CD (ctr,left)CU
N, VERT (top,top)NWDIAG (bottom,bottom)VERT
P, VERT ((top,top)CL) (ctr,bottom)CL
R, VERT ((top,top)CL) (ctr,bottom)CL (bctr,top)NWDIAG
S, CR (ctr,top)CL
T, VERT (top,ctr)HOR
W, CU (ctr,bottom)VERT
Z, CL (bottom,ctr)CR

The above rules may be enhanced and supplemented by the user to provide different interpretations of the characters. This may improve the recognition rate further.

7. DESCRIPTION GENERATION

The description generator decomposes the observed characters into the character primitives. Effectively, this corresponds to mapping, or fitting, primitives to the data which at this stage consists of critical point lists. Each of the primitives can themselves be represented by critical points. Therefore, description generation is the final data reduction stage that reduces the critical point lists into sequences that directly correspond to primitive descriptors.

7.1 Definition of Primitives

We define eight quantized directions as shown in Fig. 8. The numbers associated with the directions are known as the *direction codes*. These vectors serve as the components of the high-level concave primitives.

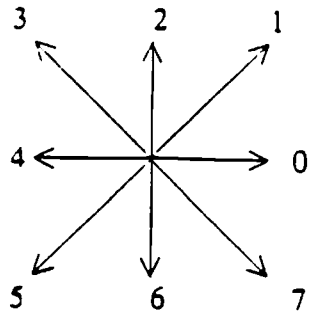

Figure 8: The eight quantized directions.

Concavities may be expressed as a sequence of the eight vectors given in Fig. 8, i.e. a chain code. In general, a concavity is characterized by any unidirectional sequence of vectors that lie between the *terminal vectors* for that concavity. Terminal vectors simply refer to the vectors found at both ends of a concavity. For example, the illustrations given in Fig. 9 all represent a valid concave down, *CD*. The attribute that links these variations into a common *CD* description is given by the set of terminal vectors which must be present.

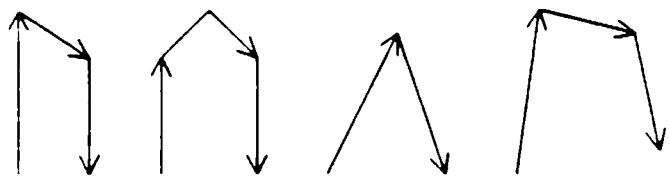

Figure 9: Valid *CD* concavities.

The set of valid terminal vectors for each of the concavities is given in Fig. 10. This figure is broken down into two parts: one for counter-clockwise and one for clockwise traversal of the concavity. In either case, each concavity has a pair of direction vectors associated with the start and end of the traversal. For example, consider a *CU* traversed counter-clockwise. The (6,7) entry indicates that the *CU*, when traversed counter-clockwise must begin with either direction

vector 6 *or* 7. Furthermore, the (1,2) entry indicates that it must terminate with either direction vector 1 *or* 2.

	Counter-clockwise		Clockwise	
	Start	Stop	Start	Stop
CU	(6,7)	(1,2)	(5,6)	(2,3)
CD	(2,3)	(5,6)	(1,2)	(6,7)
CL	(0,1)	(3,4)	(7,0)	(4,5)
CR	(4,5)	(7,0)	(3,4)	(0,1)

Figure 10: Set of terminal vectors for concavities.

The first step in description generation requires us to inspect the sequence of directions traversed along the vectors defined by the critical points. This implies that a list of direction codes must be extracted from the critical point list.

We begin by partitioning the discrete space into eight regions. The slopes of successive critical points are then computed and assigned a direction code. Concavities are then fitted to sequences of directions with the use of state tables.

7.2 Description Registration

After establishing a sequence of directions to be represented by a concavity, it is necessary to measure the concavity for asymmetry, and then register it in a description list. If the concavity is found to be asymmetric, it becomes necessary to break the concavity into a more symmetric one with an additional line appended to it (i.e. 'U','J'). Also, if the concavity does not satisfy the north, south, east, or west orientation, it is registered by its two chords (i.e. 'L'). This permits us to add emphasis on the fact that such concavities actually have less tolerance for distortion in our alphabet.

Therefore, there are three actions which may be taken on a concavity. They include:

1) Registering one concavity alone. This is characterized by the 'U' example.
2) Registering one concavity and one line. This is typified by the 'J' example.
3) Registering two lines. This is characterized by the 'L' example.

There are several rules which dictate the actual registration of primitives.

1) No registered primitive can be entirely embedded in another. This means that they must either overlap or be attached at their endpoints. In this manner, there is continuity in the shape description.
2) If a concavity is broken into a concavity and a line and the line is at the trailing end of the sequence, then temporarily store the line. It may be overwritten with a more significant

concavity on the next fitting. Otherwise, if the line is at the start of the sequence, definitely register it.

8. RECOGNITION

For each pattern under consideration, a set of rules are applied individually. Each rule attempts to verify the existence of the referenced descriptors, as well as their interrelationships, as specified by the rule's location functions. This recognition is achieved by parsing. The two chunks of data consisting of the character formation rules and the list of descriptors interact in the parsing process.

The hierarchical syntactic structure of PDL rules is made explicit by the graphical representation of a *parse tree*. For example, consider the following rule for 'H'.

H, VERT (ctr, left) HOR (right,ctr) VERT

Figure 11 is generated for the rule given above. Notice that all the leaves consist of descriptors while the nodes are comprised of location functions that verify the proximity and interrelationship of the referenced descriptors. The leaves of the tree are ordered from left to right — in the same order as that in which the rules are read.

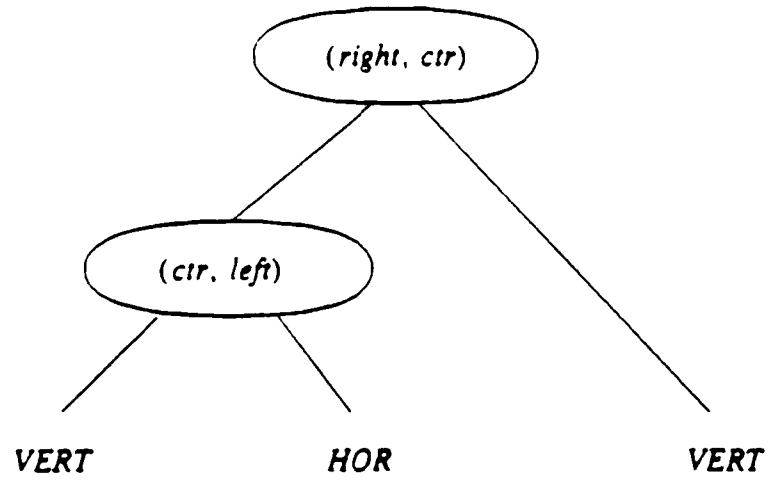

Figure 11: Parse tree for 'H' rule.

In reality, the parse tree exists only as a sequence of actions made by stepping through the tree construction process. At each node in the traversal, an evaluation is made regarding the existence of the descriptors, and their proximity as specified by the referenced location functions. If, at any time, the proximity criteria (discussed in Sec. 8.1) is violated, then the character associated with the tree is eliminated as a candidate for recognition. However, if the criteria is satisfied, then the node takes on the value of one of its two children, and the tree traversal continues.

Once a tree has been traversed from all of its leaves to the top root, then the corresponding character is considered a candidate for recognition and its confidence is given by a number that was computed during the traversal. Effectively, the result of parsing all the PDL rules is to generate a set of parse trees, known as a *forest*, with associated confidence values. The winner of the recognition process is that character corresponding to the tree with the highest value. Simply stated, this confidence value is the number of descriptors that were matched, less a penalty for borderline proximity.

8.1 Proximity

A criteria must be established to determine the proximity of two descriptors as specified by a PDL rule. Recall that a location function applied to a concavity will return a point on the primitive. We now resolve this point into a rectangle. Two points are determined to meet the location function specification if their associated rectangles intersect. The size of the rectangle is proportional to the size of the concavity. For example, the *CD* concavity in Fig. 12a is shown to be divided into regions *A*, *B*, and *C*. Region *A* is the set of pixels that satisfy the *left* function, and regions *B* and *C* satisfy the *ctr* and *right* functions, respectively. Also shown in Fig. 12a are the associated rectangles for the regions.

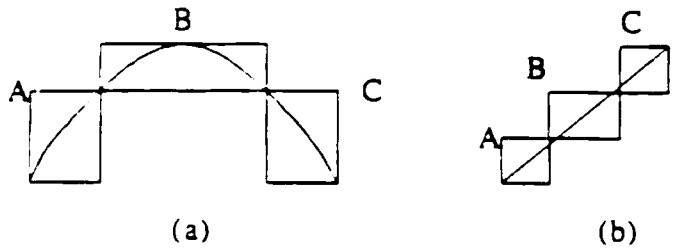

Figure 12: (a) Concavity and (b) line segment are divided into three regions.

Now that the referenced points have been resolved into rectangles, it is necessary to determine their relative positions. If the rectangles intersect, we can safely say that the two points are sufficiently close together. However, if the rectangles do not intersect, then we must measure their relative displacement. We define Δx as the minimum distance between the two bitmaps along the x-axis. If the two bitmaps overlap, we set Δx to 0. The same procedure is applied to the y-coordinates to arrive at a Δy distance.

Recall from Section 4 that an interval of noise is intended to be confined to *SMPL* pixels, where *SMPL* is the sampling distance. If such an interval is allowed in the x and y directions, then the following expression must be satisfied in order to consider the two rectangles to be sufficiently close together.

$$\Delta x^2 + \Delta y^2 < SMPL^2 \tag{8.1}$$

Eq. (8.1) is known as the *proximity criteria*. Later, we will see how it is used in computing the

confidence value for the character associated with the parse tree.

8.2 Confidence Value

At each node, a confidence value is computed and added to the existing total for that parse tree. Initially, all confidence values for characters are set to 0. If the evaluation is satisfied, then we increment the confidence value for that character by 1. In addition, we wish to subtract a *penalty* term owing to the distance between rectangles. A reasonable heuristic to arrive at such a penalty is given as

$$penalty = \frac{\Delta x^2 + \Delta y^2}{SMPL^2} \tag{8.2}$$

Therefore, given that Eq. (8.1) is true, the term

$$(1 - penalty)$$

is added to the current confidence value. However, if the evaluation is not satisfied, then we proceed directly to parse the next rule in the rules file without storing the value computed thus far.

This procedure of parsing and computing confidence values is iterated for each PDL rule. Note that rules having to match more components will potentially yield higher confidence values and are thereby favored over those rules matching fewer descriptors. Of course, multiple rules for each character assures a finer matching metric. Finally, the candidate associated with the highest confidence value is selected as the recognized character.

9. RESULTS AND CONCLUSIONS

In the character recognition domain, it is often difficult to compare performance levels between systems due to variation in the character databases. This system, however, has been shown to recognize a wide range of styles. For a database of 489 characters representing the 26 uppercase Roman letters, a 95.2% recognition rate was achieved. In the remaining 4.8%, the actual character had the second highest value 2.7% of the trials, and it had the third highest value .9% of the trials. The characters, written by many different users, ranged from acceptable to poor in quality. The poorer characters were replete with gaps, and stroke overlaps.

Three sample runs are shown in Fig. 13. They illustrate the character skeleton, a list of generated descriptions (including critical points), and a list of candidates for recognition along with their confidence values. Notice that the candidates include subpatterns within the character. These characters, though, are recognized as subpatterns through their lower confidence values.

Errors have usually been attributed to ambiguous merges at the junction points. For example, it is not always clear which of several strokes should be merged at the junction points. This problem can be remedied by a more sophisticated merging routine that keeps track of several choices for merger. This would be akin to dynamic programming. Of course this will add to the description list, and consequently the search in the parsing procedure; however, a more complete analysis will be achieved.

The system offers several unique features. Firstly, it provides scale-invariant recognition. This is a consequence of the character primitives being defined only in terms of the terminal vectors. The sampling rate is a function of the size. However, in this system, experiments with character bitmaps ranging from 16×16 to 48×48 pixels, a sampling rate of 3 consistently yielded good results.

Secondly, the system includes a user-definable tolerance to tilt orientation. Since the user may define the maximum and minimum slopes of strokes that may still be considered horizontal or vertical, respectively, the sensitivity to tilt is adjustable. Excessive immunity to tilt, however, has the ill-effect of degrading the recognition of those characters which make use of the *NEDIAG* and *NWDIAG* descriptors. This is the case since their ranges in slope are paying the price for the increase in tilt insensitivity.

Additionally, another feature is the ease in the user's ability to modify the set of patterns to be recognized. This takes form in two ways. Firstly, since the rules file is interpreted, it must only be edited to edit a pattern. Secondly, if the user is not quite sure how to describe the pattern, he must only run several versions and inspect the output of the description generator. Taking those results which underly the output, the user may be assisted in writing PDL rule(s) for that pattern. In this manner, the system can be used to aid the formation of rules for new patterns.

The system is written in C and runs on a DEC PDP-11/45 under the UNIX† operating system. Roughly two characters were recognized per second. However, two of the modules

† UNIX is a trademark of AT&T Bell Laboratories

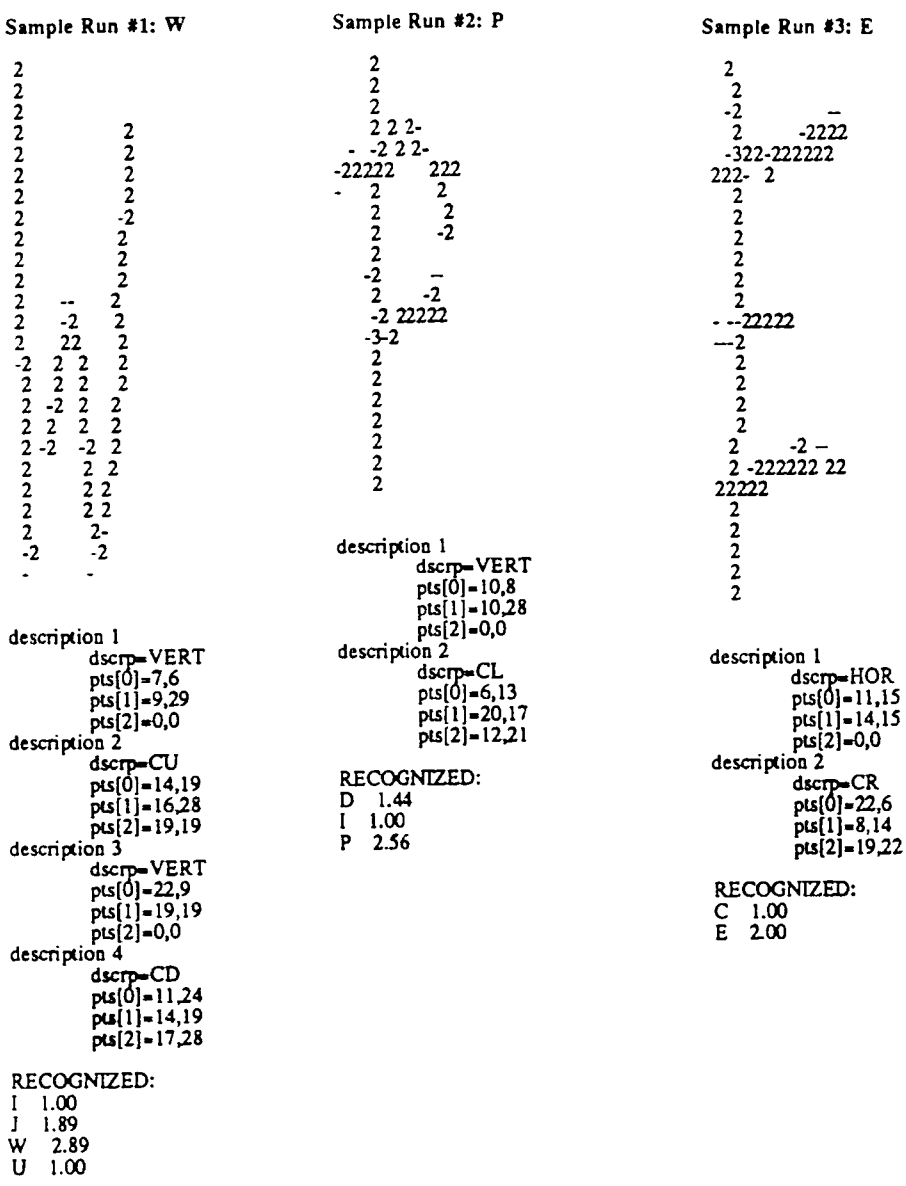

Figure 13: Sample runs.

(thinning and parsing PDL rules) are ideally suited for parallel processing and therefore subject to large speedups.

10. SUMMARY

The character recognition system discussed in this paper has made use of the syntactic rule-based approach. Our understanding of character formation allows us to piece together those critical primitives that are essential to the character. This knowledge becomes effective only at the final stage of the recognition system when the character formation rules are actually parsed. Up to that point, the system extracts a list of descriptors that is the result of fitting members of the character basis set to the observed character.

Since hand-printed characters exhibit a wide range of styles, it is crucial that our system be flexible and captures only those shapes, which may be hidden under many others, that are intrinsic to the character represented by the given pattern. A set of character formation rules is created to specify the interrelationships of primitives necessary to recognize the characters. This format has the disadvantage that the entire forest of trees must be parsed, and parsing is generally not a fast operation. However, since our rules define the minimal set of components for each character, there are relatively few levels for each tree (usually no more than four levels). Furthermore, the parsing stage can be implemented in parallel by assigning one processor per parse tree. An advantage of this technique is that intuitive rules are used to define characters. These rules are easy to write and can be added or deleted from the list of rules at any time. The proximity evaluations made at the nodes of these trees make use of parameters that are user-definable. Therefore, the user may determine the tolerance of the eight orientations introduced in Section 7. In addition, the proposed method can be applied to the recognition of any pattern that can be composed of the primitives provided for by PDL. Implemented together with context and a more sophisticated merging routine, the system's 95.2% recognition rate will be further increased and thereby deal with an even larger class of characters.

REFERENCES

[1] F. Ali and T. Pavlidis, "Hierarchical Syntactic Shape Analyzer," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-7, pp. 2-9, Jan. 1979.

[2] C. Arcelli and G.S. di Baja, "A Width-Independent Thinning Algorithm," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-7, pp. 463-474, July 1985.

[3] B. Blesser, R. Shillman, T. Kuklinski, C. Cox, M. Eden, and J. Ventura, "A Theoretical Approach to Character Recognition Based on Phenomenological Attributes," in *Proc. 1st Int. Jt. Conf. Pat. Recog.*, pp. 33-40, Nov. 1973.

[4] H. Blum, "A Transformation for Extracting New Descriptions of Shape," in *Symposium on Models* for *the Perception of Speech* and *Visual Form*, Cambridge, Mass: MIT Press, 1964.

[5] J.M. Brady and B.J. Wielinga, "Reading the Writing on the Wall," in *Computer Vision Systems*, Hanson and Riseman, (Eds.), NY: Academic Press, 1978.

[6] D.L. Caskey and C.L. Coates, Jr., *Proc. 1st Int. Jt. Conf. Pat. Recog.*, pp. 41-49, Nov. 1973.

[7] N. Fujii, H. Sugawara, E. Yamamoto, C. Ito, and T. Fujita, "Some results on handprinted Kanji character recognition using the feature extracted from multiple standpoint," *Trans. IECE Japan*, vol. PRL81-32, 1981.

[8] S. Mori, K. Yamamoto, and M. Yasuda, "Research on Machine Recognition of Handprinted Characters," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-6, pp. 386-405, July 1984.

[9] H. Nagahashi, and M. Nakatsuyama, "A Pattern Description and Generation Method of Structural Characters," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-8, pp. 112-118, January 1986.

[10] T. Pavlidis, "An Asynchronous Thinning Algorithm," *Comput. Graph. Image Processing*, vol. 20, pp. 133-157, 1982.

[11] A. Rosenfeld and A. Kak, *Digital Picture Processing*, Volume 2, NY: Academic Press, 1982, p.232.

[12] A.C. Shaw, "Parsing of graph-representable pictures," *J. Ass. Comput. Mach.*, vol 17, pp. 453-481, 1970.

[13] H. Takahashi, "A simple recognition method for handprinted Kanji characters by using primitive connective directions of thinning," *Trans. IECE Japan*, vol. PRL82-8, pp. 57-62, May 1982.

[14] F.M. Wahl, "A New Distance Mapping and its Use for Shape Measurement on Binary Patterns," *Comput. Graph. Image Processing*, vol. 23, pp. 218-226, 1983.

[15] P.S.P. Wang, "A New Character Recognition Scheme with Lower Ambiguity and Higher Recognizability," *Pattern Recognition Lett.*, vol. 3, pp. 431-436, December 1985.

[16] G. Wolberg, ''An Omni-font Character Recognition System,'' M.E.E thesis, Cooper Union School of Engineering, Oct. 1985. (Available from UMI, Ann Arbor, Michigan).

[17] K. Yamamoto and S. Mori, ''Recognition of handprinted characters by outer most point method,'' *Pattern Recognition*, vol. 12, pp. 229-236, 1980.

[18] K. Yamamoto, and A. Rosenfeld ''Recognition of handprinted Kanji characters by a relaxation method,'' in *Proc. 6th Int. Conf. Pattern Recognition*, pp. 395-398, Oct. 1982.