Symbol recognition without prior segmentation

Badr Al-Badr and Robert M. Haralick

The Intelligent Systems Laboratory, FT-10
University of Washington, Seattle, WA 98195
{badr, haralick}@cs.washington.edu

ABSTRACT

Recent advances in OCR technology are shifting the research focus from recognizing isolated, high quality characters, to reading degraded documents and cursive script. In systems that recognize such text, the segmentation phase becomes the pivotal stage in the system, to which a sizable portion of processing is devoted, and a considerable share of recognition errors is attributed.

Here we describe a new method for recognizing cursive and degraded text. Using this method, symbols on a page are identified by detecting primitives (parts of symbols), and then finding the best global grouping of primitives into symbols. On an image of text, primitives are detected using mathematical morphology operations, in a way that does not require or involve a prior segmentation step.

This paper lays out the overall strategy of a system that implements the recognition method. A following paper will report on experimental protocols and results.

This system has three major features: (1) by globally optimizing the process of combining primitives into symbols, it is robust and less sensitive to noise; (2) it does not require segmenting a text block into lines, a line into words, nor a word into characters; and (3) it is language independent in that training determines the symbol set it recognizes.

1 INTRODUCTION

The Optical Character Recognition (OCR) problem, in its most general form, is having a page that has a number of symbols and the objective is to recognize the symbols or match them against a set of known symbols. The factors that affect recognition accuracy are the mode of writing, the condition of the input page, the printing process, the quality of the paper, the presence of extraneous markings, and the resolution and quality of scanning. The OCR problem is especially important when the input is noisy, and when the symbols are connected. That is when OCR becomes hard and most systems fail.

In many cases, the symbols to recognize are connected to one another or are composed of multiple disconnected parts. For example, Roman script is handwritten cursively, and the Arabic language is machine-printed and hand-written cursively. Even for machine-printed (non-cursive) Roman script, kerning can join adjacent symbols as in the case of "fi" and "ff." Also, Roman script has symbols that are composed of multiple disconnected parts (e.g., lower case "i"). In the more general case, noise (from sources such as scanner resolution, thresholding, and print and page quality) can connect or disconnect symbols. This is the problem of connectivity and segmentation.

1.1 Survey of literature on character segmentation

In the literature, researchers have used different techniques to handle connected text. One such technique is to recognize a set of connected symbols (word) as a unit with no segmentation.[6] This means keeping a catalog of the

features of all the words that the system can recognize, which is inflexible when recognizing general text.

In the more common case, researchers use *segmentation* to separate the components of connected text. Segmentation methods for cursive[5,7,11] and machine printed[5,9] Latin text have been studied extensively. Segmentation methods follow two major approaches: (i) separate segmentation and recognition stages, with segmentation breaking up a connected portion of text into parts and then recognizing the parts; or (ii) interleaved stages of segmentation and recognition.[4]

When using *independent segmentation*, a connected portion of text can be partitioned at different locations: symbol boundaries and potential connection points.

Symbol boundaries. Here, if the segmentation step fails to accurately isolate a symbol, then most certainly that symbol will be mis-recognized; this makes the segmentation step the most critical step in the recognition process. To work effectively, segmenting at symbol boundaries relies heavily on heuristic information about the character set, the dimensions of characters, and the writing process.[2]

Potential connection points. This means segmenting a word into parts possibly smaller than a symbol. The usual recognition scheme here is to recognize the parts and then combine them into symbols. The advantage of segmenting into primitives and not characters is that it is easier to identify a set of potential connection points, which would include all the actual connection points, than to directly identify the actual points. This, however, can significantly increase or reduce the number of shape classes to recognize (depending on how a primitive is defined).

Attempting to segment at symbol boundaries is very difficult for cursive text where the connection points between symbols are inherently ambiguous, and is especially hard for handwritten script where the shapes of the symbols vary. Additionally, recognizing symbol parts can fail when segmentation produces a part that is larger or smaller than what the system can recognize.

In *interleaved segmentation-recognition*, the segmentation stage suggests a set of possible connection points at which to segment, and then the recognition stage modifies the confidence in a connection point. This technique is particularly effective in dealing with the ambiguity of cursive text; and can handle mistakes in segmentation by re-segmenting when recognition fails.[4] Some of the different ways this technique can be implemented include: elastic matching and recursive segmentation-recognition.

Elastic matching. Segment a word at all potential connection points. The resulting sequence of symbol parts are then matched to a database of stored symbols, which then produces the symbol (or set of symbols) that best matches the sequence of parts.[10] This resembles approximate string matching that attempts to find the best match between two potentially differing strings of symbols, and is very similar to the techniques of speech recognition.

Recursive segmentation-recognition. Segment a word into symbols, and then attempt to recognize the symbols. If the symbols cannot be recognized, then the procedure is recursively repeated by segmenting again.[3]

Still, methods for interleaved segmentation-recognition do not adequately address the problems associated with cursive script and noise, especially noise that affects connectivity of symbols or noise that connects symbols from consecutive text lines. Those methods examine only local neighborhoods (usually in the horizontal direction) to define the boundaries of a symbol. In many cases finding the best local match for a symbol does not necessarily produce the best match for all symbols on a line or a page.

Finally, one method of recognizing cursive and connected text avoids segmentation, altogether. This method detects a set of symbol primitives on a block of text, without a priori segmenting the text into lines, words, or symbols. Each symbol is defined as a spatial arrangement of symbol primitives; and whenever the arrangement of a symbol is found on the image, the symbol is detected.[1] This method, however, requires manually defining each symbol in terms of primitives and depends heavily on font type and size.

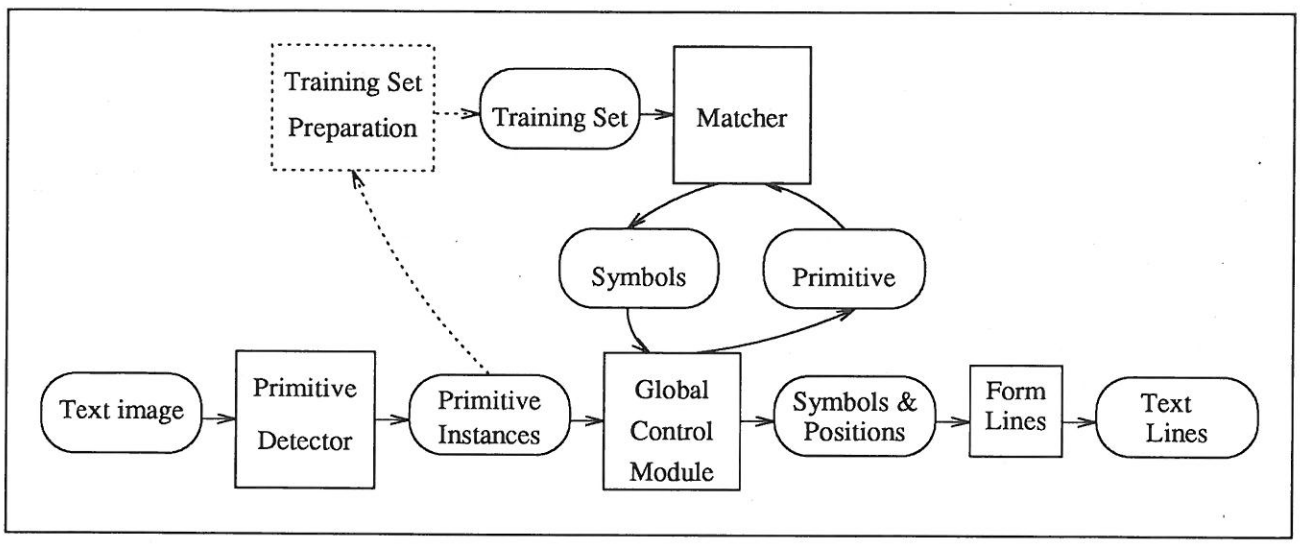

Figure 1: The organization of the symbol recognition system. (*Legend:* The boxes correspond to processes and the ovals correspond to data structures. The dotted lines denote off-line processes.)

1.2 System overview

This paper describes a symbol recognition system that overcomes the limitations of the above techniques in recognizing noisy and cursive text. To recognize symbols on an image of a block of text, the system will first detect a set of *primitives* (parts of symbols) on the block, and then find the best grouping of primitives into symbols. The advantage of this recognition-by-parts method is that the shape primitives can be found on a text block without prior segmentation. Hence, escaping a computationally costly and inaccurate segmentation step.

The system defines a symbol as a collection of primitives at a certain spatial configuration. A symbol is recognized in any local area that has enough correct primitives, at the right relative locations, to make that symbol. However, since a primitive can be a considered part of several alternative symbols, the assignment of primitive to symbol must be done globally and not locally. Also, in the presence of noise, not all primitives of a symbol might show up; primitives that do not belong to any symbol might show up; and primitives from different symbols might merge. In those cases, maintaining a global view on the process of grouping the primitives into symbols can help. When a primitive can belong to several symbols, it should be assigned to the symbol that makes the most global sense, e.g., the assignment that leaves less primitives on the block not belonging to any symbol, and makes the most symbols have good matches. To find the best *overall* grouping of primitives into symbols for the text block, our system searches the space of groupings of primitives into symbols. The search here assures that the grouping found minimizes a global objective function. The objective function measures the dissimilarity between grouped primitives and symbol prototypes, as well as the number of primitives that could not be grouped into symbols.

This paper lays out the foundation and overall strategy for the recognition system. A subsequent paper will report on experimental protocols and results. The system has three major components: (1) the primitive detector, which detects primitives on text images, (2) the matcher, which proposes groupings of primitives into symbols, and (3) the global control module, which controls the matcher and selects among groupings. Figure 1 shows the overall organization of the system.

In the next section, we will explain how primitives are detected on an image. In section 3, we will formulate the OCR problem as a state space search problem. Then, in section 4, we will describe the recognition system (the matcher and global control) and explain how it recognizes symbols.

2 PRIMITIVE EXTRACTION

Scanning a block (line, paragraph, or page) of text produces a text image. On this image, the primitive detector finds locations where instances of a symbol primitive are present, and calculates a number of features of the primitive instance. Instances of primitives are found by applying morphological transforms like erosion and the hit-and-miss transform to an input block, with the shape of the primitive as a structuring element. Morphological features have been used successfully for recognition of disconnected characters in general, and recognition of cursive character recognition in particular.[1] Before we describe the primitive extraction procedure, we briefly introduce the mathematical morphology operations that we use.

2.1 Mathematical morphology operations

We will briefly define the terms and notation, and explain the mathematical morphology operations that we use in this work. For a more complete discussion of mathematical morphology, we refer the interested reader to chapter 5 of the book of Haralick and Shapiro.[8] Mathematical morphology is an algebraic system of operators that can decompose complex shapes into their meaningful parts. Morphology operations can simplify images and remove irrelevant features, while preserving their fundamental shape characteristics. The main morphological operations are dilation and erosion. From them we can compose the other operations, hit-and-miss transform and closing residue, that we will use. Note that mathematical morphology operations are represented in the language of set theory: The set of all black pixels in a binary image is a complete description of the image.

Dilation is the morphological transformation that combines two sets by adding the elements of one set with each of the elements of the other set (using vector addition). If A and K are sets in $\mathcal{Z} \times \mathcal{Z}$, where $\mathcal{Z}$ is the set of integers, then the dilation of A by K is the set of all possible sums of pairs of elements, one of the pair coming from A and the other from K. To clarify, let $A = \{(1,2),(4,3)\}$ and $K = \{(3,3),(1,2)\}$; the dilation of A by $K = \{(4,5),(2,4),(7,6),(5,5)\}$.

Definition 2.1 *The* **dilation** *of A by K is denoted by $A \oplus K$ and is defined by*

$$A \oplus K = \{c \in \mathcal{Z} \times \mathcal{Z} \mid c = a + k \text{ for some } a \in A \text{ and } k \in K\}.$$

In practice, the sets A and K are not thought of symmetrically. A is the set representation of the image undergoing morphological processing, while K is referred to as the *structuring element*, the shape used to dilate A and produce $A \oplus K$. Intuitively, dilating an image by a disk structuring element is like expanding the image in all directions with an amount equal to the radius of that disk.

The *erosion* of A by K is the set of all elements x for which $x + k$ belongs to A for every k in K.

Definition 2.2 *The* **erosion** *of A by K is denoted by $A \ominus K$ and is defined by*

$$A \ominus K = \{x \in \mathcal{Z} \times \mathcal{Z} \mid x + k \in A \text{ for every } k \in K\}.$$

The structuring element K may be visualized as a probe that slides across the image A. Whenever K translated to x is contained in A, x belongs to $A \ominus K$. Erosion is the morphological dual of dilation, in that eroding the foreground with K is equivalent to dilating the background with the reflection of K. The erosion operation can be used to detect shapes on the foreground that contain a set of pixels that has the same spatial configuration as the structuring element.

The *his-and-miss* transform is an operation that detects shapes that have specified foreground *and* background components. It is useful for detecting corner points, isolated points, and border points; and can perform template matching. It is a combination of two erosion operations, with two structuring elements, one on the foreground, and one on the background. Let J and K be two structuring elements and t be a point that specifies the position of K relative to J.

Definition 2.3 *The* **hit-and-miss** *transform of A by $(J,K)_t$ is denoted by $A \otimes (J,K)_t$ and is defined by*

$$A \otimes (J,K) = (A \ominus J) \cap (A^c \ominus K_t)$$

where A^c is the complement or inverse of A and K_t is K translated by t.

This operation can be visualized as having two probes that slide across the image A. Whenever J translated to x is contained in A and K_t translated to x is contained in A, x belongs to $A \otimes (J,K)_t$.

In order to define the last operation that we will use, closing-residue, we introduce the next definition. The *closing* of A by K selects precisely the points of A^c that match the reflection of K. Closing an image with a disk structuring element smoothes the contours, eliminates small holes, and fills gaps on the contours.

Definition 2.4 *The* **closing** *of A by K is denoted by $A \bullet K$ and is defined by*

$$A \bullet K = (A \oplus K) \ominus K.$$

Definition 2.5 *The* **closing-residue** *of A by K is denoted by $A \bar{\bullet} K$ and is defined by*

$$A \bar{\bullet} K = A - (A \bullet K).$$

The closing-residue of A by K detects on A all the locations where K does not fit in the background. It is useful for detecting concavities and holes in an image.

2.2 The primitive detector

The main task of the primitive detector is to find instances of a set of *primitive types* on a text image. A primitive type is completely specified by its underlying morphological operation and structuring element(s). The erosion and closing-residue operations each require one structuring element, while the hit-and-miss operation requires two structuring elements and a point that specifies their relative location. To detect instances of a primitive type on a text image, the primitive detector applies the morphological operation of the primitive type with its structuring element. Figure 2 shows the letter "A" and the structuring elements for three primitive types, whose operation is erosion, which can be used to recognize the letter.

For a particular primitive type, applying its morphological operator to a text image produces a new image with instances of the primitive type showing up as blobs at different locations. Each pixel in a blob specifies the location where an occurrence of the primitive was detected on the image. Figure 3 shows the result of applying each of the three primitives types (defined in figure 2), which use erosion, to the image of letter "A." In this example, each primitive application resulted in one recognition blob showing up. Information about the blobs is extracted from the resulting image by a connected components labeling operation.

The relevant information about a *primitive instance* (a blob) is the location of the centroid of the blob, and a feature vector whose members include the size of the blob, and its bounding box. Hence the application of the operator that defines primitive type p to a text image results in a number of primitive instance tuples of the form: $(p, r, c, \vec{v})$, where (r, c) is the row and column coordinates of the centroid of the primitive instance relative to image coordinates (the upper left corner), and $\vec{v}$ is the feature vector.

The shapes of the primitives (or structuring elements) that we use include lines at different lengths and angles, corners (where each corner is formed by two lines that share an endpoint) at different line lengths and angles, elliptic arcs at different radii and beginning and extent angles, circles and disks at different radii, and rectangles at different sizes. Initially we use a large set of primitive types and eventually keep only the ones that are judged to be predictive.

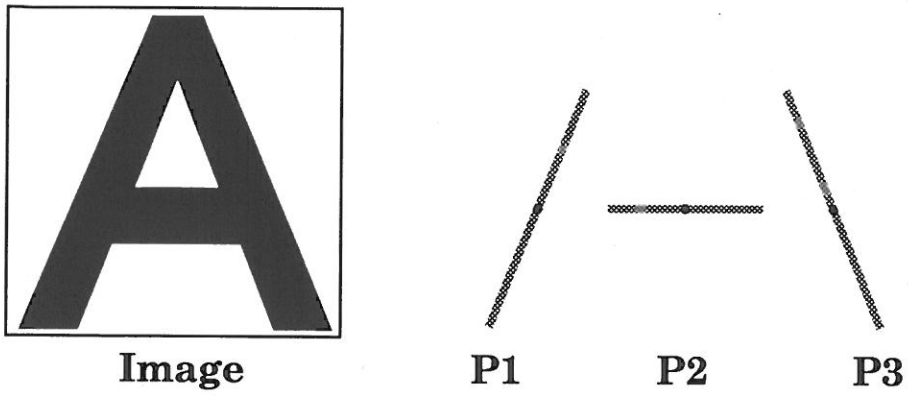

Figure 2: An example of primitive definition and detection. On the right is an image of the character "A" and on the left are structuring elements of three primitives, $P1$, $P2$, and $P3$, with their center points marked.

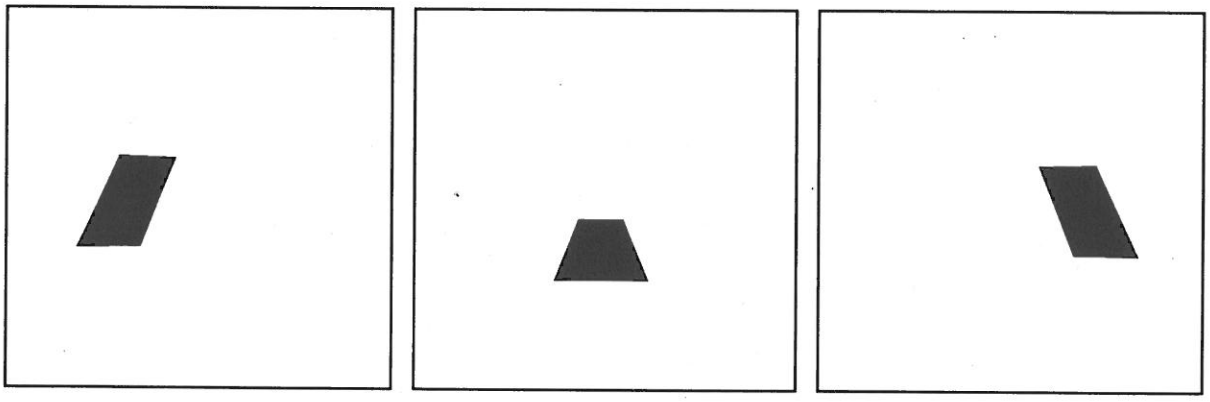

Figure 3: The result of eroding the above image with each of the three primitives, $P1$, $P2$, and $P3$.

3 PROBLEM FORMULATION

As we have argued earlier, the problem of grouping the primitives into symbols and hence recognizing the symbols, is best solved by posing it as a search problem. Let $\mathcal{P}$ be a set of predefined primitive types, each with an underlying morphological operator and structuring element(s). Let $\mathcal{L}$ be a set of symbol classes. This is the symbol set that the system recognizes. When recognizing printed characters, it is the set of all characters to be recognized. Each symbol class is to be defined in terms of a number of primitive types and their locations relative to one another using a training set. Further, let $\mathcal{X}$ be the set of all points in the Cartesian space, $\mathcal{X} = \mathcal{Z} \times \mathcal{Z}$.

After detecting all instances of the primitive set $\mathcal{P}$ on a text block, the input to the search problem is the set, $\mathcal{S}$, of M primitive instances

$$\mathcal{S} = \{(p_m, r_m, c_m, \vec{v}_m)\}_{m=1}^{M}$$

where the components of the tuples have the same meaning as explained above. Additionally, another input is a reference or training set, T, that is made up of primitive instances that have been already grouped into K symbols, with each group labeled with the class of the symbol. This will be used as a model for grouping primitives. Each element of T is called a symbol occurrence

$$T = \{(G_k, n_k, b_k)\}_{k=1}^{K}$$

where, for the k^{th} symbol occurrence, the first component, G_k, is a set of I_k primitive instances, $G_k = \{(p_g, r_g, c_g, \vec{v}_g)\}_{g=1}^{I_k}$, the second component, $n_k \in \mathcal{L}$, is the class of the symbol represented by G_k, and the third component, b_k, is the bounding box that encloses all the primitives of the symbol occurrence.

The objective of the search is to group the primitive instances in $\mathcal{S}$ into symbols, as best as possible, and return the symbols and their locations. The search should return a grouping

$$R = \{S, A_1, \ldots, A_J, q_1, \ldots, q_J, l_1, \ldots, l_J\}$$

for some positive integer J. Here the set $\{S, A_1, \ldots, A_J\}$ constitute a partitioning of $\mathcal{S}$. The search groups a set of primitive instances, A, into a symbol by matching the set with symbol occurrences in the training set. Each grouping, A, has a location $q \in \mathcal{X}$, and symbol identity $l \in \mathcal{L}$. The set S includes the primitive instances that were not grouped into a symbol and are considered extraneous.

The best grouping of the primitive instances in $\mathcal{S}$ into symbols minimizes the objective function, $\mathcal{F}$, which measures the number of unmatched primitives and how bad the groupings match their training set counterparts,

$$\mathcal{F}(\{S, A_1, \ldots, A_J, l_1, \ldots, l_J\}) = |S| * w + \sum_{j=1}^{J} \Delta(G_{l_j}, A_j)$$

where w is a positive scalar weight, G_{l_j} is the set of primitive instances of a symbol occurrence in the training set whose label is l_j and that was matched to A_j, and $\Delta(C, D)$ is a function that computes the distance (or dissimilarity) between two sets, C and D, of primitive instances. The computation of Δ is detailed in section 4.1.

Now posing the problem as a state-space search problem, Σ, we can characterize it by four components:

$$\Sigma = \langle \mathcal{T}, \omega, s, \mathcal{G} \rangle$$

where the components have the following meaning:

- $\mathcal{T}$ is a set of states, $\mathcal{T} : 2^{\mathcal{S}} \times \mathcal{X}^* \times \mathcal{L}^*$ (where the superscript $*$ denotes a sequence of zero or more elements of a set).
- ω is an operator that operates on a state and returns a set of states, $\omega : \mathcal{T} \rightarrow 2^{\mathcal{T}}$.
- $s \in \mathcal{T}$ is the start state.
- $\mathcal{G} \subseteq \mathcal{T}$ is a set of goal states.

Here, each state t is a 3-tuple made up of a set of subsets of $\mathcal{S}$ that completely partition it, a sequence of point locations, and a sequence of symbol classes

$$t = \left(\left\{ S^t, A_1^t, \ldots, A_{|t|}^t \right\}, \langle q_1^t, \ldots, q_{|t|}^t \rangle, \langle l_1^t, \ldots, l_{|t|}^t \rangle \right)$$

where $|t|$ is the number of symbol labels already assigned in state t. The set S^t is the set of primitive instances in $\mathcal{S}$ not grouped into a symbol yet. Each set A_i^t is a set of primitive instances interpreted as a symbol of class l_i^t. Hence, the space of the search is that of (partial) groupings of primitive instances into symbols.

The start state, s, is the root of the search tree and has $s = (\{S^s\}, \langle\emptyset\rangle, \langle\emptyset\rangle)$. A goal state $u \in \mathcal{G}$ is a leaf of the tree, where S^u has no subsets that can be grouped into symbols.

If node y were a descendant of node x, then y would have one more class label assigned, so $|y| = |x| + 1$ and $S^y \subset S^x$. In other words, applying the operator ω to a node x returns a set of states Y; a state $y \in Y$ has a group of primitive instances removed from S^x and assigned to a new set $A_{|y|}^y$ with label $l_{|y|}^y$.

In the above characterization of the recognition problem, segmentation and recognition are interleaved, while maintaining a global view on the process. The emphasis here is on finding a solution that is optimal with respect to a whole block of text. This can be contrasted with the previously discussed methods that consider only local neighborhoods.

4 THE RECOGNITION SYSTEM

In the recognition system, searching is conducted by the global control module and the matcher. The task of the *global control module* is to oversee recognition by controlling the search. Being at a certain node of the search tree, it selects a primitive instance from the text block and passes it to the matcher. The *matcher* then, for each symbol in the training set of which this primitive instance can be part, checks for the presence of other primitive instances that make up the symbol and computes a distance (dissimilarity) measure. If it finds enough primitive instances at the right relative locations, then it groups those instances into a symbol and returns the symbol identity and value of the distance measure.

The global control module takes that set of proposed symbols and makes each symbol a descendant node of the current node and evaluates each node. The control module then searches again from the node with the best value. This process continues until all the primitives are either grouped into symbols or are determined to be extraneous.

In the following sections we will go into more detail in describing how the matcher and the global control model operate and interact when recognizing symbols. We will then explain how text lines are formed from the recognized symbols. Figure 1 graphically shows the relations and interfaces between the system components.

4.1 The matcher

The task of the matcher is to get a primitive instance from the global control module and, by consulting a model for symbols, checks for other primitive instances that are at the correct relative positions that help complete a symbol. When a given neighborhood lends support to multiple symbols, all the matching symbols are returned, each with an associated quality of match rating.

The matcher is implemented as a nearest neighbor classifier. As such, it uses the training set, T, as a model for symbols when doing the matching. Each symbol occurrence, o, in T is made up of three components

$$o = (G, n, b)\,.$$

Here G is the set of primitive instance of the symbol, $G = \{(p_g, r_g, c_g, \vec{v}_g)\}_{g=1}^{I_o}$, n is the class of the symbol represented by G, and b is the bounding box that encloses all the primitives of the symbol occurrence.

In the training set, T, the row and column coordinates, (r_g, c_g), of a primitive instance g, in a symbol occurrence, are with respect to the upper left corner of the symbol's bounding box, b. The training set has a number of sorted indices to assist in accessing the symbol occurrences, with one index per primitive type.

We prepare the training set off-line, in a four step process. First, we manually delineate, by bounding boxes, all the symbols on a set of training text image. Then we label each symbol's bounding box by the class of the symbol. This is done automatically by reading text files that correspond to the text images, and, in reading order, mapping symbols to boxes. The next step is to have the primitive detector detect instances of all primitive types on the training images. Finally, we group the primitive instances into symbols by grouping together all the primitive instances that fall into a symbol's bounding box, and labeling them by the label of that symbol.

Each time it is invoked, the matcher receives a primitive instance, h, and the set of unused primitive instances, S. The row and column coordinates, (r_h, c_h), of a primitive instance h, are with respect to the input image coordinates (upper left corner of image). The location of this instance determines the neighborhood where a symbol is to be found. Assuming that the type of primitive p_h was π, the matcher will check for symbol occurrences in the training set that are known to include a primitive of type π, and will return several alternative solutions. Each alternative corresponds to a match between a group of primitive instances in the neighborhood of h and a symbol occurrence in the training set. Each returned alternative includes a matched symbol occurrence, a symbol position, the set of primitive instances used in the match, and a measure of the quality of match. Formally, an invocation of the matcher returns a set of N tuples

$$U = \{(o_n, q_n, A_n, \Delta)\}_{n=1}^{N}$$

where each tuple is a possible grouping of some primitives in S into a symbol: o is the symbol occurrence in the training set that was matched, q is the position of the recognized symbol on the text block that includes instance h, A is the set of primitives used from S to make up the recognized symbol, and Δ is the value of the distance (dissimilarity) measure between A and the primitive set of the matching symbol occurrence o.

The matcher uses the training set, T, to find a set of candidate matching symbols and computes the distance between each candidate and the neighborhood around h. The set of candidate symbols, C^π, is the set of all symbol occurrences in T that have an instance of primitive type π

$$C^\pi = \{(G, n, b) \in T \mid \exists g = (\pi, r, c, \vec{v}) \in G\}.$$

Each candidate symbol occurrence $o = (G, n, b)$ in C^π has an instance of primitive type π designated as its origin. A symbol occurrence in T that has m instances of primitive type π will be represented m times in C^π, with each time a unique instance of π as the origin. The coordinates of the bounding box, b, and all other primitive instances in G are relative to the location of the location of the π instance.

The matcher computes the distance (dissimilarity) between the neighborhood of h and each symbol occurrence $o = (G, n, b)$ in C^π. When matching with occurrence o, the neighborhood of h is defined by the bounding box b. Hence the neighborhood set of h relative to o is

$$N = \{(p_g, r_g, c_g, \vec{v}_g) \in S \mid ((r_g, c_g) - (r_h, c_h)) \in b\}$$

i.e., it is the set of all primitive instances in S whose location relative to h is within box b.

For an occurrence o, the distance is computed between the set, G, of primitive instances of o, and N, the set of all the primitive instances in the testing set S that are in the bounding box b around h. The distance measure matches up the primitive instances from the two sets and accumulates the distance between each matching pair. Between the two sets, only primitives of the same type can be matched. Primitive instances that are unmatched are penalized by the weight w.

The subsets of primitive instances of type π in sets G and N are G^π and N^π, respectively. Also, let P be the set of all (unique) primitive types of instances in G. To denote all the possible matching pairs for primitive type π, we define a set of relations between primitives of type π in two sets G^π and N^π as

$$F^\pi = \{f \subseteq G^\pi \times N^\pi \mid f \text{ is one-to-one and single-valued}\}.$$

The distance between G and N is defined as

$$\Delta(G, N) = \sum_{\pi \in P} \left[\min_{f \in F^\pi} \left\{ w(|G^\pi| - |f|) + \sum_{(g,n) \in f} d(g, n) \right\} \right]$$

where w is a positive scalar weight, d is a function that returns a positive distance between two primitive instances of the same type, with the constraint that w is greater than the maximum distance returned by the function d.

As N is the set of primitive instances in a rectangular neighborhood around h, we need to extract from it only the instances that were actually used in the match. The set of used primitives, A, becomes the set of all primitive instances from N that were used in each relation $f_{\min}$ that minimized the above sum, for all unique primitive types π in P,

$$A = \{h\} \cup \{g \in N \mid g \in \text{range}(f_{\min}),\ \forall f_{\min} \in F^\pi \text{ and } \pi \in P\}.$$

Note that $\Delta(G, N) = \Delta(G, A)$, since A includes all the primitive instances from N that affect the distance.

The position, q, of the matched symbol is the centroid of o's bounding box, b, when the origin of the box is translated to the location of h, (r_h, c_h).

Hence when the matcher is invoked with primitive h, the set to be returned to the global control module, U, will include a tuple for each symbol occurrence in C^π

$$U = \{(o, q, A, \Delta(G_o, A)) \mid o \in C^\pi\}.$$

4.2 The global control module

The main task of the global control module is to oversee the grouping (or partitioning) of the set of primitive instances

$$\mathcal{S} = \{(p_m, r_m, c_m, \vec{v}_m)\}_{m=1}^{M}$$

into symbols in a way that minimizes the objective function $\mathcal{F}$.

The procedure of grouping the primitives into symbols is as follows. The control module starts with a search tree that has only the root node, $s = (\{S^s\}, \langle\emptyset\rangle, \langle\emptyset\rangle)$. It incrementally builds the search tree by calling the matcher, which expands a node and returns its descendants. At a particular node, $x = \left(\left\{S^x, A_1^x, \ldots, A_{|x|}^x\right\}, \langle q_1^x, \ldots, q_{|x|}^x\rangle, \langle l_1^x, \ldots, l_{|x|}^x\rangle\right)$, the control module sends the set of unused primitives, S^x, to the matcher along with a primitive instance, h, selected from S^x. The primitive instance, h, specifies a neighborhood where to look for the next symbol. An intelligent choice for this primitive is one that is very likely to be part of the next symbol in reading order: the next symbol to the left (for English text) or at the beginning of the next line if we are at the end of a line. At the root of the tree, the control module can choose a primitive that is likely to be part of the first symbol in the block.

The matcher, in turn, returns a set of I matched symbols, $U = \{(o_i, q_i, A_i, \Delta(G_{o_i}, A_i))\}_{i=1}^{I}$, with each matched symbol, o, having an associated position, q, set of primitives used from S^x in the match, A, and the value of the distance between o's primitive set, G_o, and A, $\Delta(G_o, A)$.

The global control module makes each returned symbol in U a descendent node of the current node, x. A descendant node, y, that corresponds to match i will have $A_{|y|}^y = A_i$, $A_i^x = A_i^y$ for $i \leq |x|$, and $S^y = S^x \cup A_{|y|}^y$. Also, $q_{|y|}^y = q_i$ and $l_{|y|}^y = n$, where $o_i = (G, n, b)$.

Node y is considered to be a leaf node if whenever it calls the matcher with a primitive instance $h \in S^y$, it gets back a set, U, whose minimal match quality, Δ, exceeds a preset threshold. At this point the primitive instances remaining in S^y are considered extraneous. If node y is not a leaf, it is evaluated by the function

$$F(y) = F(x) + \Delta(G_{o_i}, A_i)$$

where $F(s) = 0$. Otherwise, it is evaluated by the objective function

$$\mathcal{F}(y) = F(x) + \Delta(G_{o_i}, A_i) + |S^y| * w.$$

The evaluation function, F, is monotonically increasing when applied to a node and the chain of its descendants. The search strategy used is depth first branch-and-bound. The minimal value of the objective function for a leaf so far is used to prune nodes with equal or larger values. This is guaranteed not to prune a node that will lead to the optimal solution because F is monotonically increasing and that for any leaf, y, $\mathcal{F}(y) \geq F(y)$ (w is positive).

Among the descendants of a node, the next node to search is the one with the least value, and the process is repeated again by selecting a primitive and sending it to the matcher. Note that a selected primitive, h, might not lead to any matches that exceed the threshold. This requires selecting another primitive instance from the set of unused primitives, until a selection results in good matches or else the node is determined to be a leaf. The expansion process is repeated until all nodes are either leaves or get pruned. The best grouping (solution) is the leaf, $z = \left(\left\{S^z, A_1^z, \ldots, A_{|z|}^z\right\}, \langle q_1^z, \ldots, q_{|z|}^z\rangle, \langle l_1^z, \ldots, l_{|z|}^z\rangle\right)$, that has the minimum value for the objective function.

4.3 Line formation

Once we find the best grouping of primitive instances into symbols, z, the last step is to form lines out of the recognized symbols. Since we have the position, q_i^z, (row and column coordinates) of each recognized symbol, i, we can easily find the lines by clustering on the set of column coordinates of all symbols. An initial estimate of the number of lines is obtained by horizontal projection. Within a line, the symbols are sorted via their row coordinates.

5 DISCUSSION

In this paper, we have briefly reviewed and discussed techniques to handle noisy and connected text. We have laid out the strategy of a symbol recognition method that does not require a prior segmentation stage, and hence avoids some of the limitations of the segmentation based techniques. In this system, segmentation into symbols is a byproduct of the recognition process. The system has three major components: The primitive detector, the matcher, and the global control module. We have explained the processing done by each component and how the modules interact.

Here we discuss ways to improve the performance of the system. For one, we need a method to assess the predictive power of primitive types. Having that, then for the primitive detector, we can automatically select a subset of the primitive types, $\mathcal{P}$, that has the most predictive power and remove the rest of the primitive types, which would reduce the complexity of the system. Also, when computing the functions, $\mathcal{F}$, F, and Δ, at the global control module and the matcher, we can use variable weighting values that depend on the predictive power of a primitive type, instead of using the constant weight w. This would provide for more accurate matching and grouping and, hence, recognition.

In our method, we estimate the identities and positions of symbols on a block of text. The unit we take to be a text block can vary from word, to line, to paragraph, depending on the difficulty of isolating each of those units on a text image. The tradeoff here is between reduced search space and increased error rate that results from incorrect segmentation. For instance, if isolating the different words on a block is relatively easy and accurate, then global recognition can be done at the word level, meaning that the system would recognize a word at a time rather than a block or text block at a time. Likewise, if lines were easy to separate then global recognition can be done at the line level, and so on. As a rule of thumb, the recognition method should be applied to the smallest unit of text that can be isolated reliably.

One way to improve the recognition performance of the system is to improve the objective function and node evaluation function. One way of doing that is to use syntactic and lexical information and constraints on font type and size within a word. We expect to gain a lot of insight into improving the system by extensively experimenting with it on a variety of documents.

This system contributes in three major areas. (1) Robustness: by using global grouping of primitives into symbols it is less sensitive to noise. (2) Recognition without prior segmentation: it does not require segmenting in advance a block into lines, a line into words, nor a word into characters, segmentation is a byproduct of recognition. (3) Language independence: training determines the symbol set it recognizes.

6 ACKNOWLEDGMENTS

The authors would like to acknowledge the participation and help of James Ahrens, Mary Johnson, and Loyd Myers in early stages of this work.

REFERENCES

[1] Badr Al-Badr and Robert M. Haralick. Recognition without segmentation: Using mathematical morphology to recognize printed Arabic. In *Proceedings of 13th National Computer Conference*, pages 813–829, Riyadh, Saudi Arabia, November 1992.

[2] Adnan Amin and Sabah Al-Fedaghi. Machine recognition of printed Arabic text utilizing natural language morphology. *IEEE Trans. on Systems, Man, and Cybernetics*, (35):769–788, 1991.

[3] R. G. Casey and G. Nagy. Recursive segmentation and classification of composite character patterns. In *Proceedings of the 6th International Joint Conference on Pattern Recognition*, pages 1023–1026, Munich, F.R.G., October 1982.

[4] Christopher E. Dunn and P. S. P. Wang. Character segmentation techniques for handwritten text—a survey. In *Proceedings of the 11th IAPR International Conference on Pattern Recognition*, volume 2, pages 577–580, The Hague, Netherlands, August 1992.

[5] D. G. Elliman and I. T. Lancaster. A review of segmentation and contextual analysis techniques for text recognition. *Pattern Recognition*, 23(2/3):337–346, 1990.

[6] Raouf F. H. Farag. Word-level recognition of cursive script. *IEEE Transactions on Computers*, 28(2):172–175, February 1979.

[7] Hiromichi Fujisawa, Yasuaki Nakano, and Kiyomichi Kurino. Segmentation methods for character recognition: From segmentation to document structure analysis. *Proceeding of the IEEE*, 80(7):1079–1091, July 1992.

[8] Robert M. Haralick and Linda G. Shapiro. *Computer and Robot Vision*, volume 1. Addison Wesley, 1992.

[9] R. L. Hoffman and J. W. McCullough. Segmentation methods for recognition of machine-printed characters. *IBM Journal for Research and Development*, 15:153–165, March 1971.

[10] Sargur N. Srihari and Radmilo M. Bozinvic. A multi-level perception approach to reading cursive script. *Artificial Intelligence*, (33):217–255, 1987.

[11] Krishnan R. Tampi and Sridhar S. Chetlur. Segmentation of handwritten characters. In *Proceedings of the 8th International Joint Conference on Pattern Recognition*, pages 684–686, Paris, France, October 1986.