NISTIR 5843

Component-Based Handprint Segmentation Using Adaptive Writing Style Model

Michael D. Garris (mdg@magi.ncsl.nist.gov)

National Institute of Standards and Technology
Building 225, Room A216
Gaithersburg, MD 20899

ABSTRACT

Building upon the utility of connected components, NIST has designed a new character segmentor based on statistically modeling the style of a person's handwriting. Simple spatial features (the thickness of the pen stroke and the height of the handwriting) capture the characteristics of a particular writer's style of handprint, enabling the new method to maintain a traditional character-level segmentation philosophy without the integration of recognition or the use of oversegmentation and linguistic postprocessing. Estimates for stroke width and character height are used to compute aspect ratio and standard stroke count features that adapt to the writer's style at the field level. The new method has been developed with a predetermined set of fuzzy rules making the segmentor much less fragile and much more adaptive, and the new method successfully reconstructs fragmented characters as well as splits touching characters. The new segmentor was integrated into the NIST public domain Form-Based Handprint Recognition System and then tested on a set of 490 Handwriting Sample Forms found in *NIST Special Database 19*. When compared to a simple component-based segmentor, the new adaptable method improved the overall recognition of handprinted digits by 3.4% and field level recognition by 6.9%, while effectively reducing deletion errors by 82%. The same program code and set of parameters successfully segments sequences of uppercase and lowercase characters without any context-based tuning. While not as dramatic as digits, the recognition of uppercase and lowercase characters improved by 1.7% and 1.3% respectively. The segmentor maintains a relatively straight-forward and logical process flow avoiding convolutions of encoded exceptions as is common in expert systems. As a result, the new segmentor operates very efficiently, and throughput as high as 362 characters per second can be achieved. Letters and numbers are constructed from a predetermined configuration of a relatively small number of strokes. Results in this paper show that capitalizing on this knowledge through the use of simple adaptable features can significantly improve segmentation, whereas recognition-based and oversegmentation methods fail to take advantage of these intrinsic qualities of handprinted characters.

Keywords: connected components, handprint, handwriting, optical character recognition, OCR, public domain, segmentation, style modeling, writer adaptation

1. INTRODUCTION

It is easy to forget that character recognition technology has been commercially available since the mid-1950s. What started out to be hardware-based systems for reading machine printed text has now evolved into a host of different software-based recognition applications including the recognition of handwriting. In his look back over the history of optical character recognition (OCR), Prof. George Nagy points out that the study of isolated character classification is an "overgrazed paddock", nonetheless he looks to the future stating, "OCR research offers many greener pastures" [1]. The National Institute of Standards and Technology (NIST) conducted an experiment in 1993 in the running of the First Census Optical Character Recognition Systems Conference that draws the same conclusions for the recognition of isolated and well-segmented handprinted characters [2]. Analyses on the results from this conference conclude that existing automated off-line handwriting recognition technology performs comparably to humans in terms of accuracy, but in regards to sustained throughput and economy, the machines win hands-down. Since the conference, NIST has continued to research and develop classifier technology [3][4][5], but it has focussed much of its effort on developing entire recognition systems and studying how they can be best evaluated using automated techniques [6][7].

While the performance of character classification has significantly improved, character segmentation still remains a problematic source of errors in handwriting recognition systems [8]. Segmentation technology in general is still rather fragile, particularly when decisions are required for splitting touching characters and putting fragmented characters back together when image quality is poor. In these cases, ambiguities are very common making decisions

(at least for the machine) very difficult. One approach to overcoming these ambiguities is to oversegment the sequence of characters and then recognize permutations of the resulting pieces. The results of recognizing the different combinations are stored in a table, and given a lexicon, a transcription from the segments is generated by finding a minimal cost path through the table. Techniques for oversegmentation have been presented by Gillies [9], Gader [10], and Shridar [11]. In the Second Census Optical Character Recognition Systems Conference, this approach to character segmentation was shown to perform very well [7]. In this conference, real handprinted responses were processed from 1990 Census Long Forms, and dictionaries ranging from 20k to 60k words were provided from the previous 1980 Decennial Census. It is generally accepted that the benefits of using this brute-force method of oversegmentation are significantly compromised when no dictionary is applicable or available. This is frequently the case when recognizing numeric information.

Other recognition-based segmentation schemes have been proposed and tested that closely integrate segmentation and recognition together in a single unified approach. Examples of this have been published by Martin [12], Keeler & Rumelhart [13], and LeCun [14]. Typically a single network architecture is trained to identify one or more character objects within a specified field of view. The network scans sequences of characters left to right possibly with some higher-level control (also encoded in the network) such as in Martin's work. This integrated approach to segmentation was also applied in the Second Census Conference, but it never performed as well as the oversegmentation method. Also, these unified network architectures tend to be connection-wise very large in dimension. As a result they have proven to be difficult (but not impossible) to train.

Another approach to character segmentation was proposed by Rocha, Pavlidis, et. al., where sequences of characters are analyzed in grayscale and saddle features between strokes of touching and broken characters are identified and processed [15]. While these saddle features contain robust gradient information, they are expensive to compute and the entire approach requires grayscale data. Much of what is being scanned in terms of documents today is not grayscale, but binary (black and white), for purposes of lower bit-rate transmissions and greater data compression, reducing storage requirements.

Other researcher's have taken a completely different point of view. Their rationale is that if the traditional methods of segmenting words or numbers into individual characters is plagued with ambiguities that cause errors, then avoid segmentation all together and apply a more holistic word-based approach to recognition. Researchers such as Govindaraju have developed and evaluated holistic word recognizers only to find out that "word separation is the weakest link in the chain and continues to be a challenging research area" [16]. Certainly part of the segmentation process can be viewed as bottom-up where inter-character statistics give at least partial clues to where word gaps occur. This being true, critical information is overlooked by avoiding character segmentation. In addition, holistic word recognizers used in isolation have yet to perform as well as traditional character-level classifiers. However, one could use them in multiple classifier voting schemes and they have been successfully applied to dictionary pruning applications [17].

Prior to the study reported in this paper, NIST has used a simple method of segmentation based on connected components. The NIST public domain Form-Based Handprint Recognition System was released in 1994 [18], and to date, more than 575 copies of the system have been distributed across 40 different countries. The technology distribution contains several contributions to the state-of-the-art for system developers, including a retrainable optimized Probabilistic Neural Network (PNN) [19] for character classification and a robust training set of labeled characters. The old segmentor simply finds all the connected components in a field on a form, throws away any components that are "too small" or "too large", and if the field is designated as numeric, disconnected top-strokes on 5's are identified and merged with their proper neighbor. If characters are fragmented due to poor image quality or form removal errors, then insertion errors are inflated. If characters touch one another, then deletion and substitution errors in the system are inflated. Despite all these limitations, the NIST system reads numeric fields up to 6 characters in length completely correct about 79% of the time.

Building upon the utility of connected components, NIST has designed a new character segmentor that maintains a traditional character-level segmentation philosophy without the integration of recognition or the use of oversegmentation and a dictionary. While recognition-based segmentation and oversegmentation techniques basically concede to not explicitly knowing how characters should be separated, letters and numbers (by their very definition)

are constructed from a predetermined configuration of a relatively small number of strokes. The new method's merit is in exploiting this knowledge through the use of several simple statistical features that capture the characteristics of a particular writer's style. Rather than rely on linguistic postprocessing as with oversegmentation schemes, this new method build of model of the style of a particular writer's handprint by measuring spatial information such as the thickness of the pen stroke and the height of the handwriting. These statistics are measured from each separate response provided by a writer on a form, allowing adaptation to takes place not just at the writer level, but at the field level. As will be seen, the same piece of program code using the same set of parameters successfully segments numbers as well as sequences of lowercase and uppercase characters without any context-based tuning.

On the other hand, the new method avoids tedious and laborious stroke decomposition and symbolic processing by using these simple statistical features to identify and compare a number of more abstract and adaptable objects. As rule-based systems using symbolic processing (popular from the inception of artificial intelligence (AI)) are developed, they quickly become cumbersome and brittle. The new method presented here has been developed with a predetermined set of rules, but the features are statistically derived (rather than topologically for example). As a result the segmentor is much less fragile and much more adaptive. Although not formally a fuzzy system, the new segmentor can be viewed as being composed of fuzzy rules and objects. As a result, the segmentor maintains a relatively straightforward and logical process flow avoiding convolutions of encoded exceptions (as is common in expert systems), and the segmentor operates very efficiently. On a Sun Microsystems SPARCStation 2 with an 80Mhz Weitek Chip[1], the segmentor sustains an average throughput of 362 characters per second.

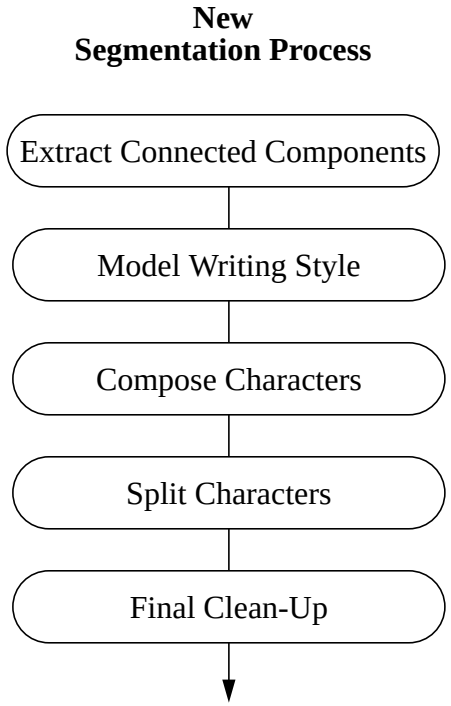


Figure 1. Simple process flow for the new segmentation method.

The new segmentation method is divided into four general steps with a final clean-up step as shown in Figure 1. The first step extracts all the connected components in the isolated image of handprint. A brief discussion of connected components is provided in Section 2. A writing style model is then constructed by measuring simple statistics from the connected components. These statistical features are defined in Section 3. The next step uses the writing style model to compose characters from multiple pieces in the image. This is frequently required when scan quality is poor

1. Specific hardware and software products identified in this paper were used in order to adequately support the development of the technology described in this document. In no case does such identification imply recommendation or endorsement by the National Institute of Standards and Technology, nor does it imply that the equipment identified is necessarily the best available for the purpose.

causing characters to break up. Putting components together is also needed to join dots and detached strokes to their character bodies. Section 4 describes the steps required to put characters back together. Next, the writing style model is used to identify components containing multiple touching characters, and the characters within the image are separated. Section 5 presents the details of how this can be done both effectively and efficiently without the use of recognition or dictionaries. Finally, a couple of simple follow-up tests are described in Section 6 that look for non-overlapping character strokes (frequently they are the tops of 5's) and small punctuation-sized components. There is no punctuation in our application, so any component that appears to be a small punctuation mark is likely to be noise or a character fragment and should be discarded. Section 7 presents both accuracy and timing results, and Section 8 draws conclusions. It should be noted that for the purposes of organization, this report has been written so that the main sections discuss the general process flow of the segmentor. The implementation details (as much as possible) have been deferred to Appendix A where rules and their parameters are presented in a pseudocode fashion.

2. EXTRACT CONNECTED COMPONENTS AND COMPUTE STATISTICS

A connected component is defined as a set of black pixels where each pixel is a direct neighbor of at least one other black pixel in the component. In other words, all the black pixels in the component are *connected* forming a blob. In this application, a neighbor is defined to be either directly adjacent or directly diagonal to a pixel. This provides 8-way connectivity. An alternative would be 4-way connectivity in which case a neighbor must be directly adjacent. The choice was made simply because it provides more connectivity without significantly increasing computation requirements.

To locate a blob, the connected component utility systematically scans the input image for a black pixel. When a black pixel is found, it is grown into a run (a maximally horizontal segment of contiguous black pixels). The run is then grown into a maximal set of connected runs, which constitutes an entire blob. When all the runs for a blob are found, the utility reconstructs and returns the blob in an output image. The routine is efficient because it localizes processing to only the black pixels in the image, and it does so one blob at a time. The algorithm's implementation generally requires an amount of working memory that is small compared to the memory occupied by the input image. In addition, all the horizontal runs are stored, so by extracting connected components, we get the runs for free.

Connected components are extracted one at a time until the whole image has been scanned. To manipulate components as objects, the data structure definition listed in Figure 2 was developed. A blob is represented by the bounding box that tightly encompasses all its black pixels. The data structure holds the component's rectangular raster image, width and height, cut position, along with other statistics such as the component's geometric center and black pixel count. The original image of handprinted characters can now be represented by a list of component structures. For segmentation purposes, the list of original connected components is a reasonably good approximation to character segments, however further processing is required so that when finished we are confident that each component in the list represents a single isolated character from the original image. To support this, a suite of utilities was developed that allocate, duplicate, append, merge, sort, and search a variable length list of components.

```
typedef struct blobstruct{
    unsigned char *data;    /* raster image data */
    int w, h;               /* pixel dimensions */
    int x1, y1;             /* cut origin */
    int x2, y2;             /* bottom-right coordinate */
    int cx, cy;             /* center coordinates */
    int a;                  /* pixel area */
    int p;                  /* black pixel count */
} BLOB;
```

Figure 2. Structure definition for a component object.

3. MODEL WRITING SYLE

To adapt to variations in handwriting style, one needs to be able to statistically capture how much black ink (or pixels) in an image is likely to constitute a single character. To do this, two simple statistical features are measured from each isolated image of handwriting. The *estimated stroke width* approximates the width of the lines comprising the characters. Variations in stroke width arise from the use of different kinds of writing implements along with different amount of pressure being applied. A good stroke width estimate is the median horizontal run length in the image. Remember that our connected component utility returns the runs. Once the width of each stroke is estimated, one needs to know how tall the strokes are in the image. A simple way to *estimate character height* is to find the maximum height of all the connected components in the image. Other (potentially more robust) methods for estimating character height were studied, but none worked any better for our application.

Given an estimated stroke width, the black pixels in an image can be thought to be part of larger meta-pixels that are basically one stroke width square. This larger pixel is referred to as a *standard stroke pixel*. Extending this notion one step further, *standard stroke area* can be defined as the estimated stroke width times the estimated character height. The scan resolution of an image conveys almost nothing in terms of the expected size of the handwriting in the image. By their very definition, characters are configurations of a finite and relatively small number of strokes. Measuring the image in terms of standard pixels and standard strokes provides a relative measure of density that adapts to the handwriting style itself, thus allowing for writer-normalized units of measure at the field level. As will be seen, these simple statistical features are sufficient to characterize or model the style of handprint for the purposes of segmentation.

4. COMPOSE CHARACTERS

At times a connected component may only represent a piece of a character. Characters may be broken into pieces due to poor print or scan quality, or strokes of characters may be written in such a way that they are detached. This is naturally the case with dotted characters, and it is frequently the case of characters with broad horizontal strokes such as ‘T’ and ‘E’. Ambiguities arise from the presence of other spurious marks such as bleed through, smudges, erasures, and residual form information. These types of components should be discarded as noise, whereas other small components should be merged together in order to construct complete characters. The task of composing characters and discarding noise is outlined in Figure 3.

The list of connected components extracted from the original image is carefully processed. Components are processed from shortest to tallest, keying off the observation that smaller components tend to be good candidates for being noise or pieces of characters. If a component is small and qualifies as noise (A.1), then it is removed from the original component list and added to a noise list. If a component is a little larger and qualifies as a dot (A.2), then it is removed from the original list and added to a dot list.

If the component is neither the size of noise nor a dot, then possibly it is a piece of a character that should be merged with another component in the original list. Horizontal overlap is used to determine if two components are merge candidates (A.3). In general the overlapping pieces are to be merge, but as the components become increasingly large, there is less chance that they are pieces of the same character, but they are more likely to be distinct characters. Therefore, tests are performed to ensure the components are not too big to be merged (A.4). If the overlapping components are determined compatible, they are merged and replaced by the composite result.

Upon one pass through the original list, some components remain unchanged, others have been merged together, while others have been collected together either as noise or dots. Noise components are compared with the components remaining in the original list (the later referred to as the original components). If a noise component is found to be inside (A.5) an original component, then the noise component is merged. This allows small fragments to be rejoined to their character bodies. Those noise components not merged are discarded.

Dots are also processed with respect to the components remaining in the original list. If a dot is inside an original component, then it is merged. If the dot is qualified to be the top of an ‘i’ or ‘j’ for a single neighbor from the

original list, then it is merged (A.6). If a dot is qualified to be the top of both its left and right neighbors, then the dot is merged with the closer neighbor. Finally, if the dot is qualified to be a top of a 5 (A.7) for its left neighbor, then it is merged. Any remaining dots not merged are considered noise and are discarded. Several segmentation results of characters that required the merging of connected components are shown in Figure 4.

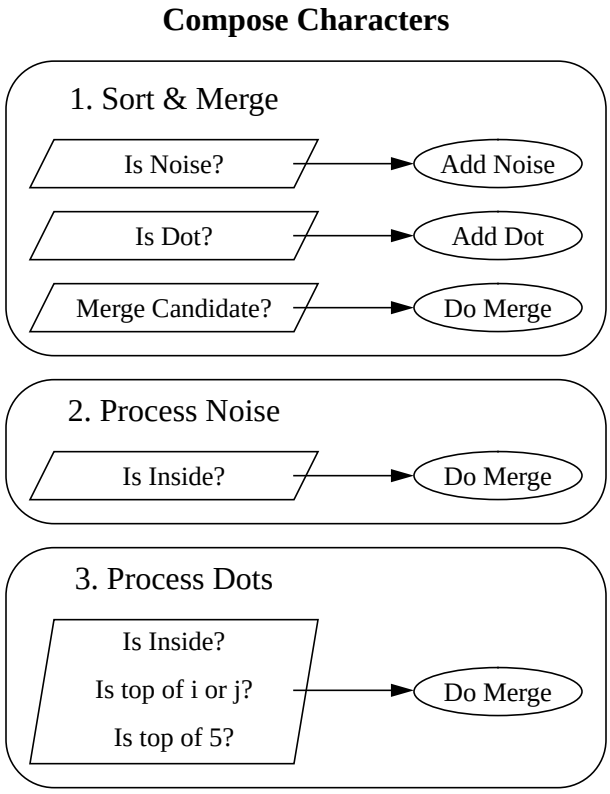


Figure 3. Steps to composing characters.

5. SPLIT CHARACTERS

Not only may a connected component represent a piece of a character, but it may problematically contain two or more touching characters. In this case the component must be subdivided so that each resulting image contains only one character. Most often, characters touch two in a row, so the first attempt at spliting tries to divide the component as a single character pair. If this doesn't work, then characters are cut from the component left-to-right, dividing the less frequent sequences of 3 or more touching characters. The task of splitting characters is illustrated in Figure 5.

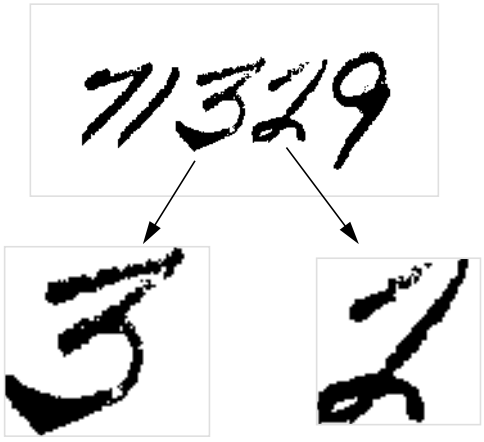
5.1 Multiple Character Detection

Before one can split touching characters, one must be able to detect that multiple characters exist in a component image. This detection must be simple to compute, relatively reliable, and adaptable to the characteristics specific to a given writer. At first, a simple *aspect ratio* was tested which is defined as

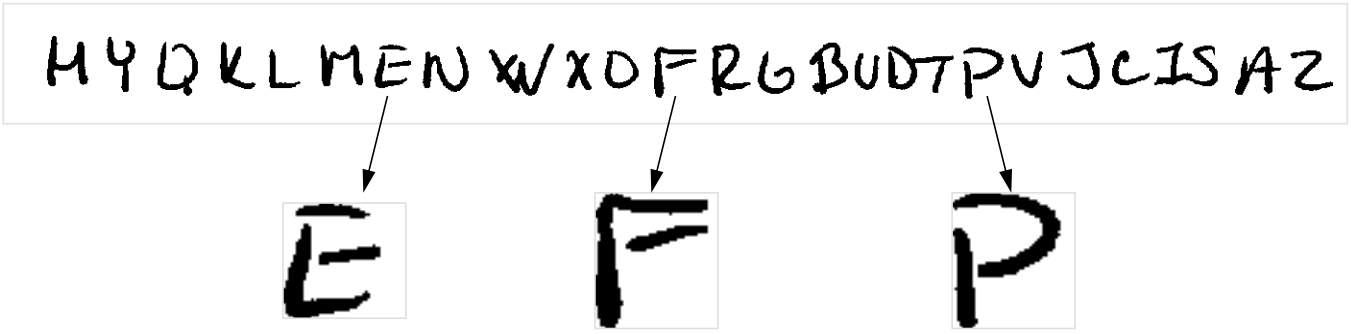
$$ar = \frac{w}{ech} \tag{1}$$

where *w* is the width of the component, and *ech* is the estimated character height for the field.

Broken Characters



Detached Strokes



Dotted Characters

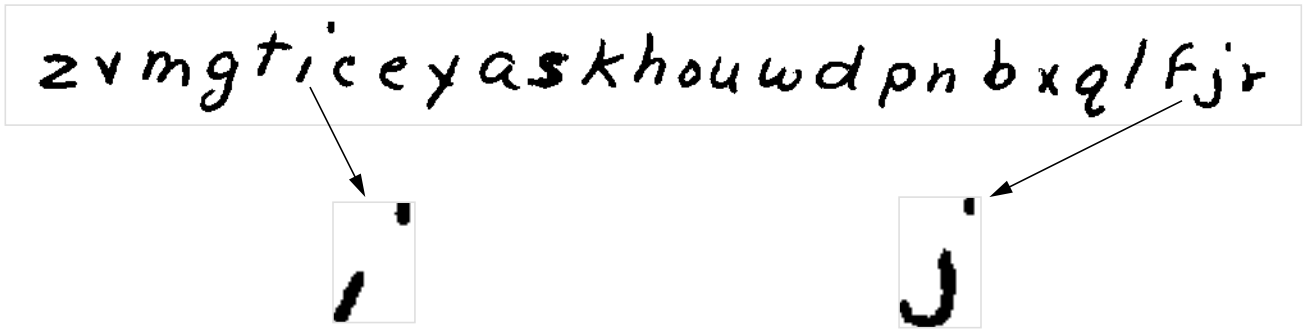


Figure 4. Segmentation results of characters composed of multiple connected components.

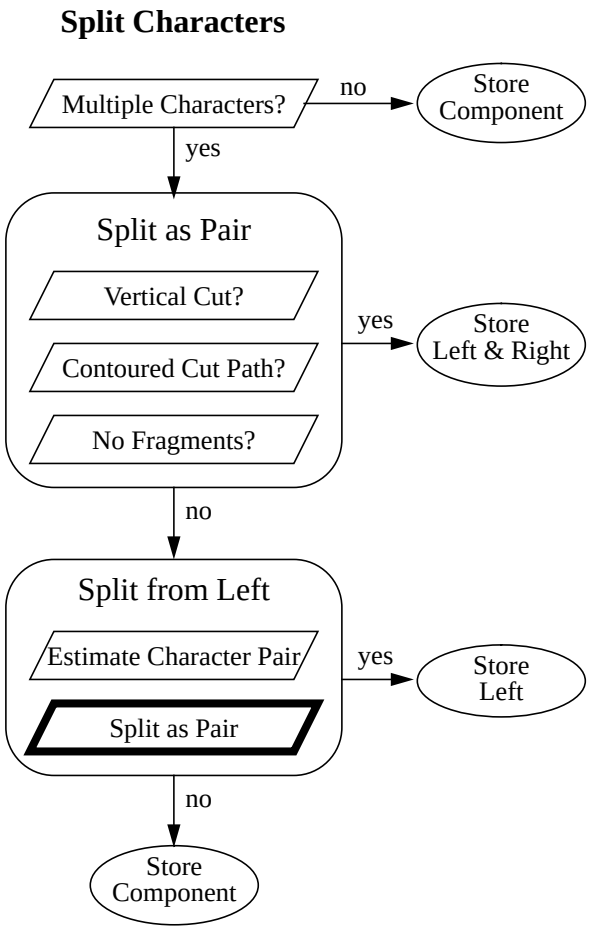


Figure 5. Steps to splitting characters.

The larger the width is to the height, the more likely the component contains multiple characters. A training set of single and touching character components was used to compute a range of aspect ratio samples, and a threshold was empirically derived. Many of the samples were correctly separated by this simple feature, but there were also a significant number of confusions. It was determined that while the overall dimensions of the component were important, it was going to be necessary to analyze the density of the black pixels within the image, and it was recognized that these densities should be measured according to some set of writer-adaptive units. It was at this point that the notion of a standard stroke was introduced and its area defined to be

$$ssa = esw \times ech \tag{2}$$

where *esw* is the estimated stroke width.

Using this definition, a second feature was computed called the *standard stroke count* or *ssc*. The standard stroke count is defined as

$$ssc = \frac{p}{ssa} \tag{3}$$

where *p* is the black pixel count of the component. This measures how many standard strokes can be constructed from the total density of black pixels in the component, and its value automatically adapts to the characteristics of the handwriting. The training set of single and touching character components was used again to compute this new feature, and

Figure 6 shows the resulting scatter plots using the two features (*ar*, *ssc*) as data points. The x-axis plots aspect ratio and the y-axis plots standard stroke count. The top plot shows the feature points derived from images of single characters, while the bottom plot shows features points from images containing multiple touching characters.

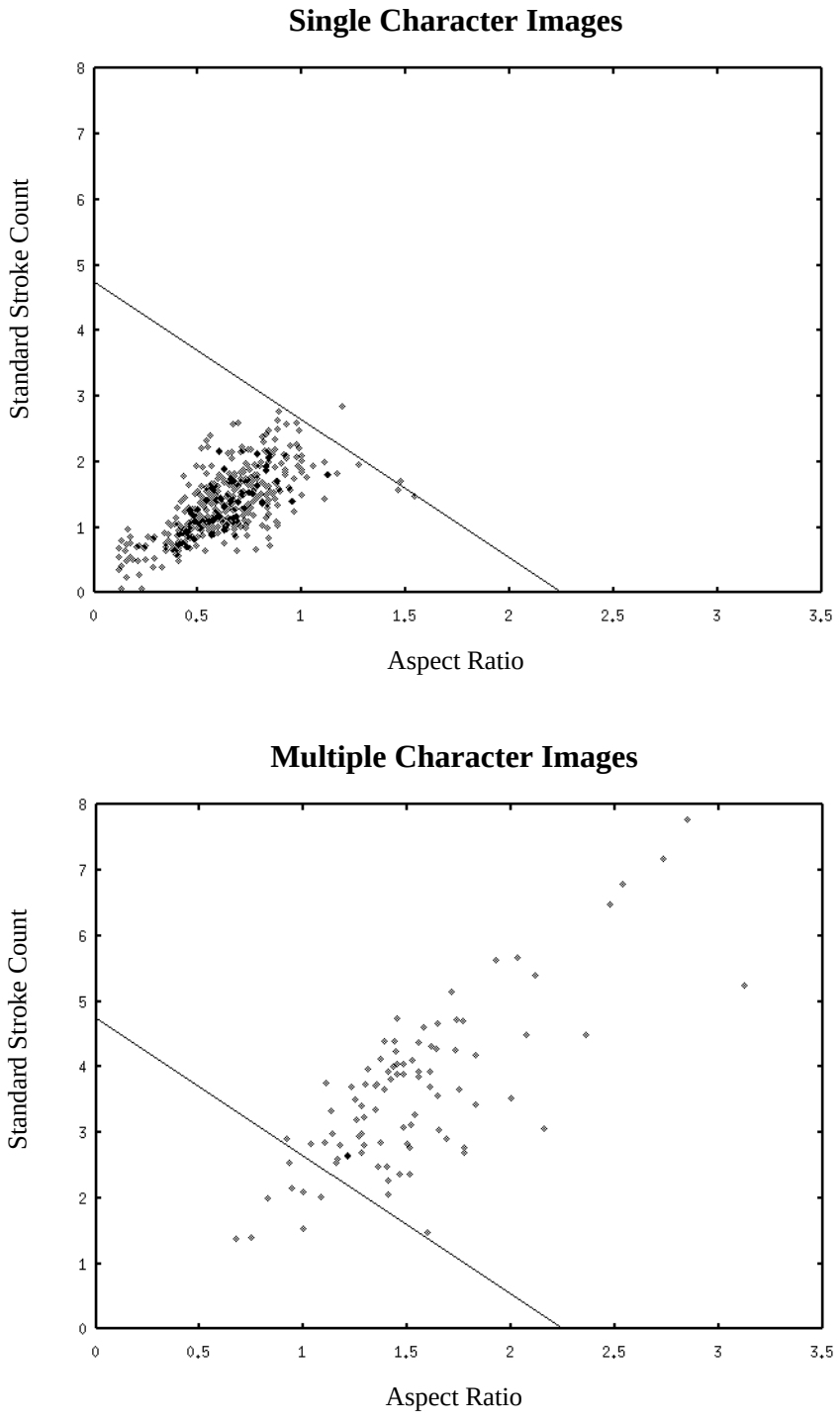


Figure 6. Top: scatter plot of single character images. Bottom: scatter plot of multiple character images.

The line drawn in both plots is a linear discrimination that spans the points (0.0, 4.75) and (2.25, 0.0). The equation for the line is

$$ssc = (-2.11)\,ar + 4.75 \quad (4)$$

and it was empirically derived to maximize the number of correctly labelled single character images. Given a component image, its aspect ratio and standard stroke count are computed. If the point falls below the detector line, the component is determined to contain a single character. If the point lies above the detector line, then the component is determined to contain multiple touching characters. As can be seen, this simple 2D detector quite accurately distinguishes between images of single and multiple characters. Much of the new segmentor's splitting capabilities hinges on the existence of this inexpensive but adaptable detector. The feature point (*ar*, *ssc*) and the multiple character detector are also considered part of the writing style model.

5.2 Vertically Straight Cut

A component determined to contain multiple touching characters must be further analyzed to derive a strategy for spliting the characters. The first step splits the touching characters along a vertically straight line that optimally divides the component into two pieces. The multiple character detector is used to determine the optimal position of the vertical cut.

Successive splits are conducted left-to-right across the component and the left and right pieces resulting from each cut are evaluated. The aspect ratio and standard stroke count (referred to as a feature point) are computed for each piece and the position of these feature points are analyzed with respect to the line used in the multiple character detector. Only those vertical cuts where both left and right feature points remain below the detector line are considered. Among this set of possible cuts, perpendicular distances are computed from the left and right feature points to the detector line and the larger of the two distances is stored along with the x-position of the vertical cut. The optimal vertical cut is selected as the x-position corresponding to the minimum of the stored distances.

Figure 7 documents the selection of an optimal vertical cut for the original image of two touching characters displayed in image (A). The connected component has pixel dimension 80×56, the estimated stroke width is 6, and the estimated character height is 56. Using these statistics, the feature point (*ar*, *ssc*) is computed to be (1.4, 4.0) and is plotted in graph (C). The point is above the detector line, so the image is determined to contain multiple characters. Successive vertical cuts are evaluated across the x-range [26..55] according to graph (D). Each vertical cut divides the original image into a left and right piece. Feature points (*ar*, *ssc*) are computed for the left and right pieces, and their perpendicular distances to the detector line are measured. The points plotted in graph (D) represents the maximum perpendicular distance from each left and right pair. By minimizing the maximum perpendicular distances across the range of cuts, the vertical cut is selected whose left and right pieces both contain maximal pixel data and both pieces qualify as single characters. The vertical cut position selected for this example is (x = 39), and the vertical cut is drawn in image (B).

5.3 Contoured Cut Path

The optimal vertical cut is sufficient for dividing many touching characters, but for other cases (such as that shown in Figure 8(A)) a single straight cut does not satisfactorily divide the component. In these cases, a more sophisticated non-straight *path* is required. A technique for deriving this path was developed that builds upon having first selected the vertically straight cut.

In general, the vertical cut represents a good first approximation for where the touching characters should be split. This straight path requires a certain amount of bending and shaping so that it follows slanted and curved character contours in order to minimize excessive cutting of character strokes. Starting at the x-position of the optimal vertical cut, a search (or trace) is initiated from the top of the component downwards and from the bottom of the component upwards. The trace downwards (the top-trace) performs much like sand being dribbled down the side of a complex

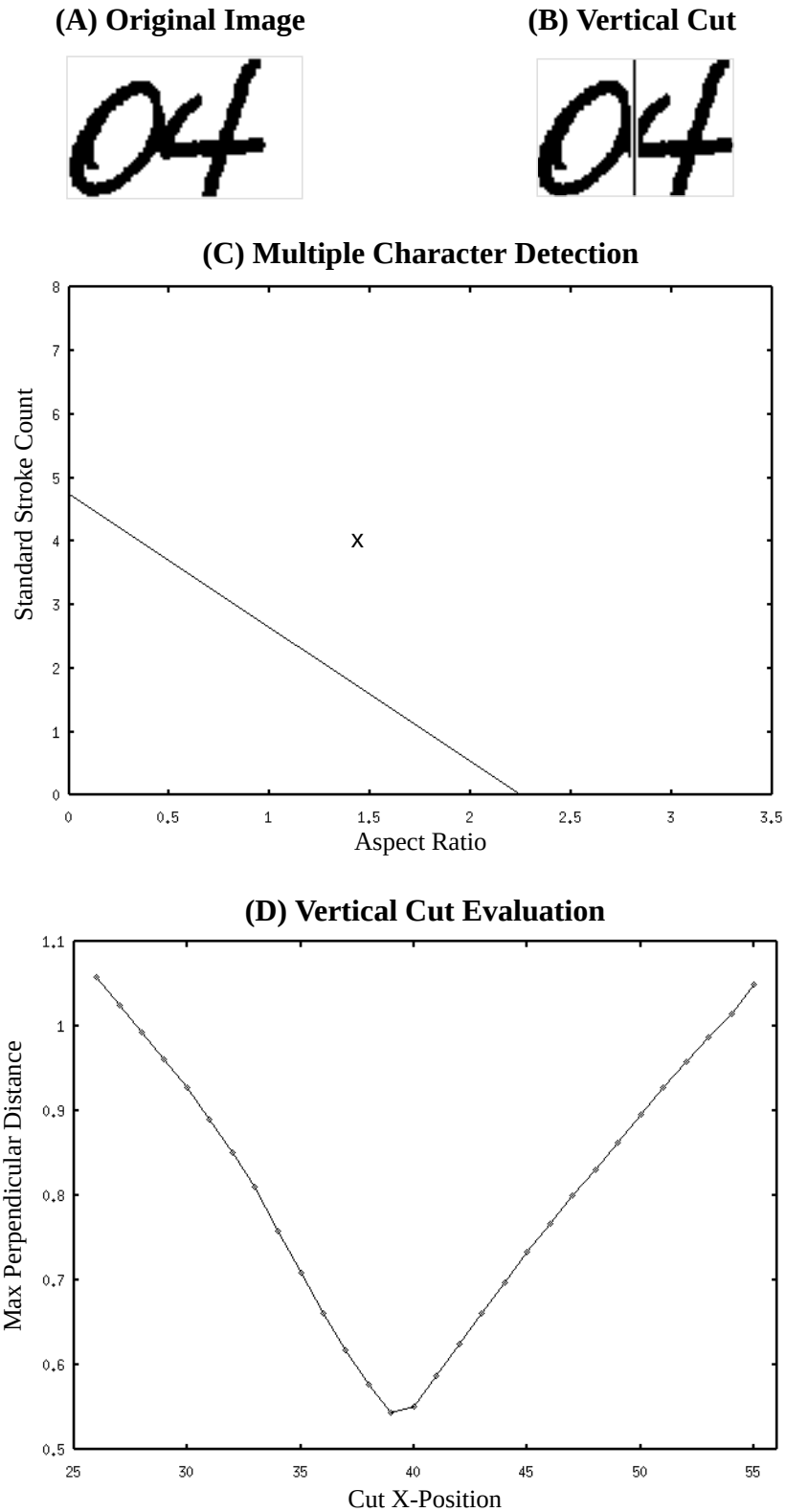


Figure 7. Example of locating an optimal vertical cut: (A) the original image, (B) the selected vertical cut, (C) the detection of multiple characters, and (D) the evaluation across the set of possible vertical cuts.

surface. The trace is free to fall vertically until it hits the edged of a character at which point it flows along the contour falling progressively downwards until either the bottom of the image is reached or it gets stuck in a local minima or concavity. The trace upwards (the bottom-trace) performs similarly.

At this point we have two paths that started from the top and bottom of the component at the vertical cut position and were permitted to trace along the complex contours of the characters. Figure 8(B) plots the top and bottom traces on the example image. A good strategy for splitting characters is to minimize the number of cuts across character strokes (transitions from white to black to white pixels). Frequently there is only one such cut transition required and this is typically located at the minimum distance between the two trace paths. The cut path is constructed by locating the two closest points between the traces and then merging the segments of the top and bottom traces leading up these points. The contoured cut path and the resulting split are shown in images (C) and (D).

(A) Vertical Cut

(B) Top and Bottom Trace

(C) Contoured Path

(D) Resulting Segments

Figure 8. Example of splitting two touching characters: (A) the selected vertical cut, (B) the results from the top and bottom trace, (C) the contoured cut path, and (D) the segmentation results.

A few exceptions must be handled. If both traces pass through the entire image, then the path that splits the component nearest the middle of the image is chosen. At times, the resulting left or right side of the split will be empty or it may contain only noise. When this happens, the starting position of the top and bottom traces are stepped away from that side and recomputed.

5.4 Split Caused Fragments?

The detection of multiple characters and the splitting of components do not utilize any recognition feedback. Instead, simple statistical features that model the style of the handprint are used, and inevitably some instances of single character components are going to represent statistical outliers. These cause unnecessary splits and therefore segmentation errors. This is particularly true of capital letters which tend to be larger in term of size and structure than digits and lowercase characters. As a result, the segmentor can be a bit greedy and unnecessarily split single characters causing fragments. In order to compensate and to help ensure satisfactory results, a simple but important test was designed to detect if fragments are introduced by the split (A.8). If they are, then the component is not divided.

5.5 Split from the Left

Occasionally, no initial vertical cut or satisfactory contour cut path can be found for a component that is determined to contain multiple characters. The entire scheme for finding an optimal vertical cut operates on the assumption that two and only two characters are in the component. While this is usually the case, at times more than two consecutive characters may touch one another forming a single connected component. As consecutive vertical cuts are evaluated left-to-right through the component, not a single cut will produce left and right feature points that both lie below the detector line. In other words, one side or the other will always contain sufficient pixel data to qualify as having multiple characters in itself.

When more than two characters touch in a single component, successive characters are split from the left end of the component image by *estimating* a subimage the width of two characters (A.9) again using the writing style model. The subimage is analyzed (as the entire component image was earlier) for an optimal vertically straight cut, and once found, the contoured cut path is traced. Given the cut path, the left character in the component is carved off and appended to the component list. The process is repeated until the remaining component image contains only a single character.

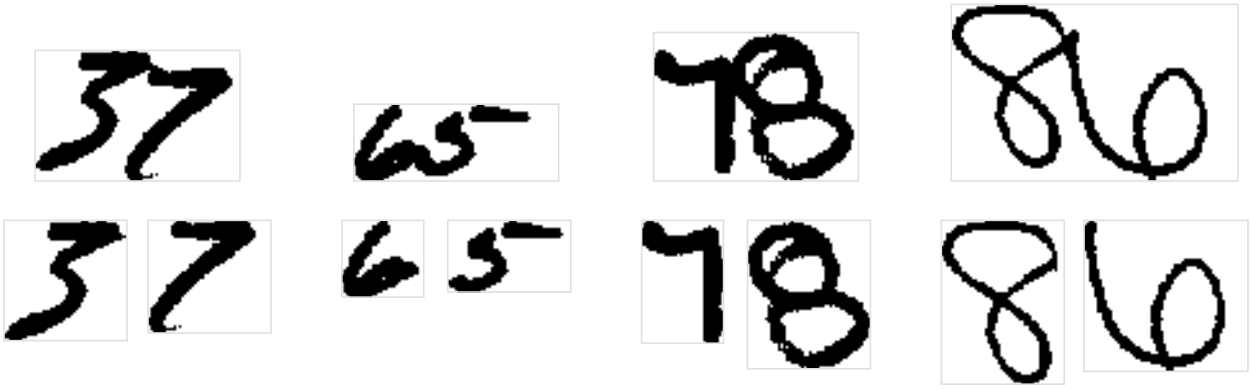
Several segmentation results of connected components that required splitting are shown in Figure 9. The top set of images in the figure show results of splitting touching character pairs. The middle set of images show the results of successively splitting characters from the left side of the component image. The bottom set of images demonstrate how the method is capable of splitting alphabetic characters as well. Notice that the first component split in the uppercase example is ambiguous. Evidently, the writer accidently wrote an ‘X’ and later realized that the correct character should have been a ‘W’. The writer than extended the strokes to form the correct character. The segmentor analyzes the component image, correctly determines that it is statistically too large to be a single character, and it proceeds to divide the component. The resulting segments (interestingly enough) form reasonable characters. There appears to be an ‘X’ followed by a ‘V’ or ‘N’. This is an example of how ambiguities can still creep in and cause segmentation errors. Recognition feedback could resolve the ambiguities in some cases, but probably not in this example.

6. FINAL STEPS

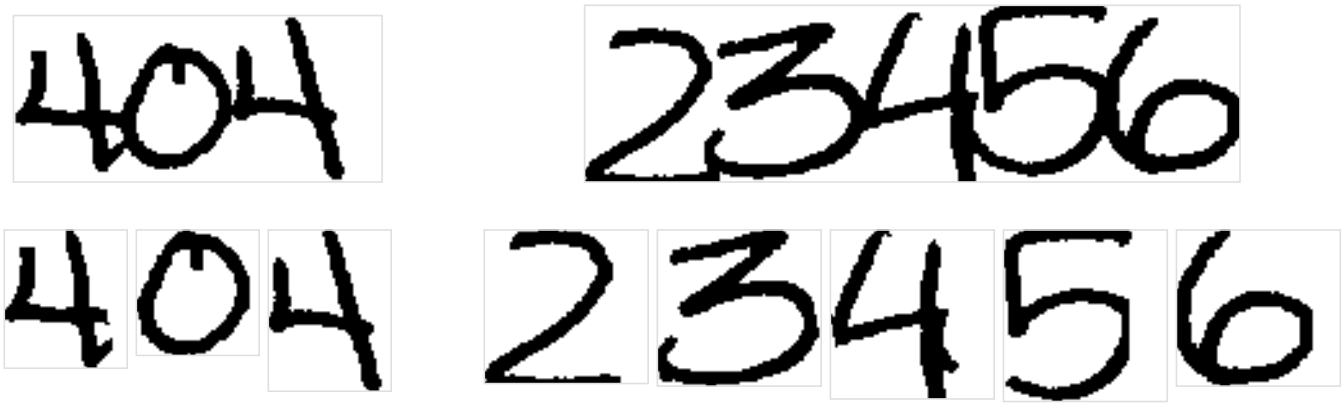
One class of character strokes that must be handled differently are the top strokes on 5’s. Frequently these strokes are detached and do not horizontally overlap with their character bodies. These strokes are often quite long, so they are not treated as noise or dots. A pass through the remaining list of components is required to detect and merge these non-overlapping horizontal top strokes. Each component in the list is tested to be a top of a 5 (A.7) with respect to its left component neighbor. If it is, then the components are merged and replaced by the composite result.

For this application, one last pass through the components is required. The fields used in this study do not contain any punctuation. Therefore, components remaining in the list that appear to be small punctuation marks (A.10) most likely represent some type of noise or fragmentation of characters that the previous processes could not deal with. For example, they may be characters fragments that were unsuccessfully joined, residual form information, or other spurious marks caused by the writer. Regardless of their source, they are not single and complete characters, so they should be discarded as noise.

Two Characters Touching



More Than Two Characters Touching



Uppercase Characters Touching

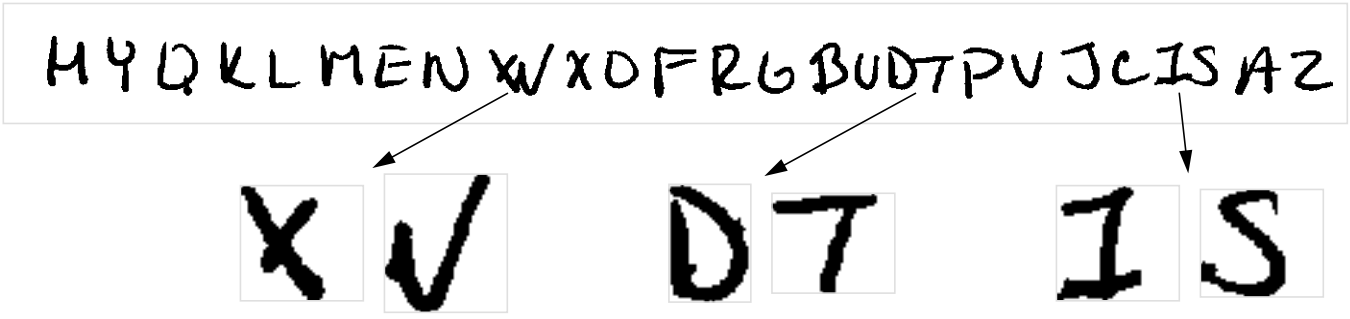


Figure 9. Segmentation results of connected components containing multiple touching characters.

7. RESULTS

While several examples of successfully segmented images have already been shown, this section analyzes the impact the new segmentor has on both the accuracy and the efficiency of a handprint recognition system. The NIST Form-Based Handprint Recognition System [18] was used to test the new statistically adaptable segmentation method. Once modified, the recognition system was run across a large collection of forms, and the recognition performances before and after applying the new segmentor were measured and compared.

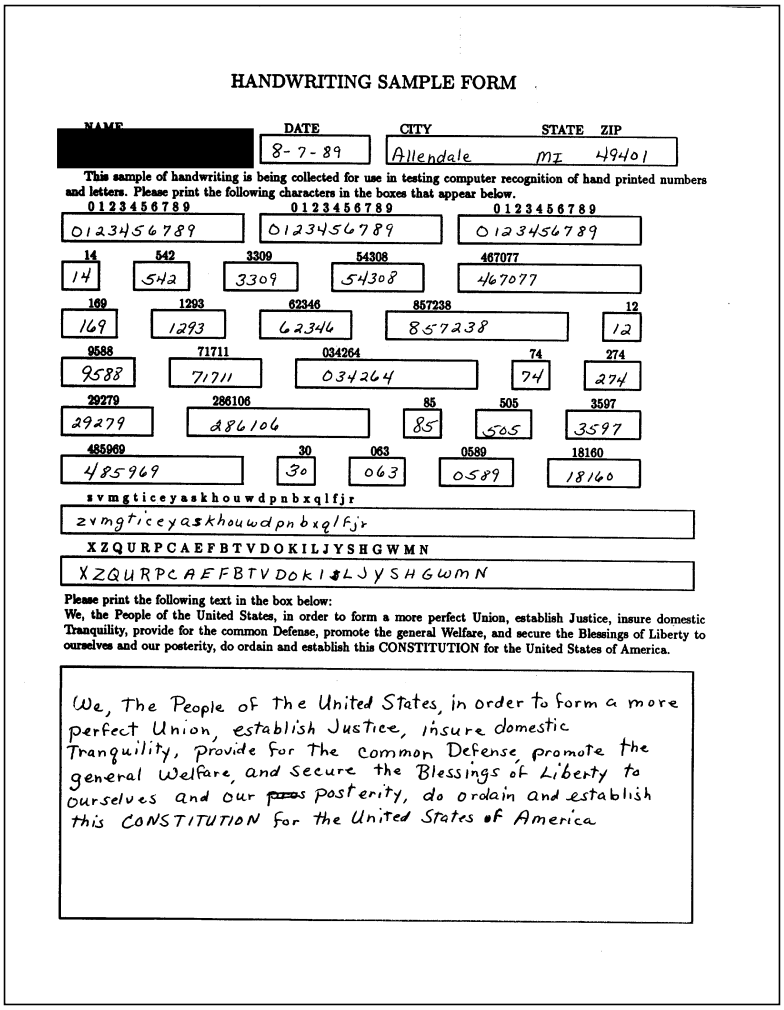


Figure 10. Completed Handwriting Sample Form from SD19.

The comparison was conducted using 490 Handwriting Sample Forms (HSF) from the beginning of NIST Special Database 19 (SD19) [20]. These forms were scanned as binary images at 300 pixels per inch (ppi). A copy of an HSF form is shown in Figure 10. The systems used in this study recognized the digit fields along with the randomly ordered lower and uppercase alphabets and stored the results to a file. The NIST OCR scoring package [6] was then used to reconcile the output from the system with what the writer's were instructed to enter in each field and performance statistics were automatically compiled.

The results are broken out and analyzed according to digits, uppercase, and lowercase fields. The results for digits are summarized in the three tables listed in Figure 11. The first table shows character and field level accuracies achieved on the 490 HSF forms. The first column of the table lists the accuracies using the old segmentation method, the second column lists the accuracies using the new method, and the third column lists the difference in recognition

performance between the two methods. The accuracies are computed as the number correctly recognized divided by the total potential. The values used to compute the percentages are displayed in parentheses. Remember that the old segmentation method simply treated each connected component as a plausible character image (except for disconnected tops on 5's), and blobs that were too small or too big. A resolution dependent scalar threshold of 100 black pixels was used to determine if a component was too small, whereas 1750 was used for too big. As can be seen from the first table, using the new statistically adaptive segmentor improves character recognition by 3.4% and field recognition by 6.9%. As a result, the recognition system is now able to read 86% of the numeric fields on HSF forms entirely correct.

The second table listed in Figure 11 shows the accumulated number of errors broken out by substitutions, insertions, and deletions. The first column in the table lists the totals compiled across the set of 490 HSF forms using the old segmentor, the second column lists the total errors using the new segmentor, and the third column lists the relative amount of improvement achieved using the new adaptive segmentation method (1.0 - (new error / old error)). The table shows a dramatic decrease of 82% in the number of deletions when using the new segmentor in the recognition system. This is due to the fact that the old method has no way of separating touching characters, and if two characters touch, they are extracted and classified as a single connected component. Using dynamic string alignment to score the results, the merged characters are typically tallied as a single substitution followed by a character deletion. The large reduction in deletion errors testifies to the new segmentor's ability to identify and split multiple touching characters. Notice that the majority of digit recognition errors remaining are substitution errors. This suggests that much of the burden for improved recognition now lies on the character classifier. It should be noted that the classifier, an optimized PNN [19], was not retrained when the new segmentor was added. Perhaps retraining the classifier on characters that the new segmentor has split will improve recognition and reduce substitution errors. This is left for a future study.

The third table of results shown in Figure 11 lists confusion pairs whose error distributions significantly changed between using the old and new segmentation methods. These changes are listed for both significant improvements and degradations. The testing set was divided up into 10 equal partitions and the errors on each partition were compiled into separate confusion matrices. Means and standard deviations were computed for each confusion pair across the partitions resulting in one set of error distributions for the recognition system using the old segmentation method and another set of error distributions for the new segmentation method. Given the distribution statistics of each corresponding confusion pair between the two systems, a Student's *t* test was used to determine statistically significant differences [21]. The difference between two distributions is measured as

$$t = \frac{\mu_1 - \mu_2}{\sqrt{\frac{\sigma_1^2 + \sigma_2^2}{n}}} \tag{5}$$

where n is the number of partitions, (μ_i, σ_i) are the distribution statistics from system i for a specific confusion pair, and the result t is in units of root mean square standard error.

Given the normalized distance t, a probability ρ is derived either numerically or via table look-up for each cell in the confusion matrix [22]. This is the probability that $|t|$ could be at least this large by chance, so the smaller the value of ρ, the less likely the two distributions are the same. The confusion pairs listed in the table have ρ less than 2% (in which case we are 98% sure the difference is significant), and they are sorted on their corresponding value of t. Confusion pairs with positive t represent significant improvements using the new statistically adaptive segmentor, whereas those with negative t represent degradations. The first column in the table lists the confusion pairs, (r)ight confused as (w)rong; the next two columns list the error distributions for the system using the old segmentation method, and the next two columns list those for the new segmentation method; the third column from the right lists the absolute error differences between the two systems' mean values; and the last two columns list the results of the Student's t test.

Examining the bottom table, it is difficult to postulate why the particular confusion pairs listed as significant improvements exist. This is probably due to the fact that the most significant contribution (as seen in table 2) made by

the new segmentation method on digits is that of splitting touching characters. Merged characters are somewhat ambiguously scored as a substitution followed by a deletion. Therefore, the old segmentor incurred a high number substitutions where the first character in the touching pair was aligned with whatever character the classifier decided the joined pair most closely resembled. Take the confusion pair (6, 0) for example. One interpretation is that a '6' frequently touches other characters on its right, and when this occurs, the classifier often identifies the pair as a '0'. The new segmentation method is able to split the pair, so what was originally substituted for the '6' no longer occurs, and the number of 6's labeled as '0' significantly decreases.

Digit Recognition			
	Old Seg.	**New Seg.**	**Improv.**
Character Accuracy	92.9% ($\frac{59208}{63700}$)	96.3% ($\frac{61317}{63700}$)	3.4%
Field Accuracy	79.2% ($\frac{10863}{13720}$)	86.1% ($\frac{11814}{13720}$)	6.9%

Digit Errors			
	Old Seg.	**New Seg.**	**Improv.**
Substitutions	2454	2022	17.6%
Insertions	398	343	13.8%
Deletions	2038	361	82.3%

Pair		**System 1**		**System 2**		**Mean Δ**	**Student's *t***	
r	**w**	μ_1	σ_1	μ_2	σ_2	μ_1-μ_2	***t***	$\rho\times10^2$
6	0	8.1	1.7	3.8	2.1	4.3	5.02	0.01
6	4	3.8	1.6	1.0	0.9	2.8	4.85	0.02
5	2	1.5	0.7	0.4	0.3	1.1	4.67	0.04
0	4	2.0	1.1	0.4	0.3	1.6	4.58	0.08
7	1	10.0	5.1	3.3	1.3	6.7	4.02	0.23
9	0	4.6	2.4	1.3	1.2	3.3	3.88	0.18
3	0	3.9	1.4	1.8	1.1	2.1	3.83	0.13
3	4	2.2	0.5	0.9	1.0	1.3	3.72	0.26
7	0	1.7	0.8	0.7	0.5	1.0	3.59	0.27
9	1	6.6	4.8	2.2	0.8	4.4	2.87	1.76
1	7	0.9	0.5	1.5	0.5	-0.6	-2.85	1.07
7	2	0.6	0.3	1.4	0.8	-0.8	-2.87	1.42
4	5	0.4	0.0	1.1	0.7	-0.7	-3.32	0.89

Figure 11. The impact of the new adaptive segmentor on digit recognition.

The confusion pairs listed as significant degradations are more readily explained. The new segmentor tends to cut towards the middle of horizontal strokes than towards either of the two ends. When the top stroke of a character on the left touches a '1' on the right, the split may cause the '1' to look more like a '7'. According to the bottom table, the increase of this type of substitution is significant, and it represents a trade-off with using the new segmentation method. But keep in mind that with the old segmentor, touching characters were guaranteed to be an error every time.

The results of recognizing uppercase fields is shown in Figure 12. The character accuracies are substantially lower than that for digits, and the overall impact of using the new segmentation method is also diminished. In fact, the character error is too high for field level statistics to be interesting, so they are not reported. Nonetheless, the same

exact code used to segment digits, did in fact improve the recognition of uppercase characters by 1.7%. Looking at the second table, it can be seen that both insertions and deletions have been significantly reduced, yet the substitution rate is relatively unchanged. These changes in error rates can be explained by analyzing the bottom table. The confusion pairs contributing to significant improvements are characters that contain large horizontal or diagonal strokes that often are written detached from their character bodies. Notice the large number of E's that are no longer incorrectly classified as another character. This testifies to the new segmentation method's ability to compose characters from multiple pieces. Large detached strokes are no longer treated as separate characters, so the number of insertion errors (as shown in the middle table) are effectively reduced by 33%.

Uppercase Recognition			
	Old Seg.	**New Seg.**	**Improv.**
Character Accuracy	84.5% $(\frac{10763}{12740})$	86.2% $(\frac{10982}{12740})$	1.7%

Uppercase Errors			
	Old Seg.	**New Seg.**	**Improv.**
Substitutions	1656	1630	1.6%
Insertions	595	397	33.3%
Deletions	321	128	60.1%

Pair		**System 1**		**System 2**		**Mean Δ**	**Student's *t***	
r	**w**	μ_1	σ_1	μ_2	σ_2	μ_1-μ_2	***t***	$\rho\times10^2$
E	A	0.7	0.3	0.1	0.0	0.6	5.69	0.03
E	P	1.3	0.7	0.1	0.0	1.2	5.69	0.03
T	I	3.4	1.4	0.7	0.7	2.7	5.59	0.01
F	H	0.9	0.6	0.0	0.0	0.9	4.93	0.08
E	W	0.5	0.3	0.0	0.0	0.5	4.74	0.11
G	T	0.9	0.6	0.2	0.0	0.7	3.83	0.40
I	L	0.5	0.3	0.1	0.0	0.4	3.79	0.43
E	M	0.4	0.3	0.0	0.0	0.4	3.79	0.43
E	H	0.9	0.8	0.1	0.0	0.8	3.39	0.79
Y	I	3.4	1.7	1.5	0.6	1.9	3.35	0.65
F	I	2.0	1.3	0.6	0.3	1.4	3.22	0.90
E	I	4.5	1.9	1.9	1.7	2.6	3.16	0.55
E	N	0.3	0.3	0.0	0.0	0.3	2.85	1.92
Z	W	0.3	0.3	0.0	0.0	0.3	2.85	1.92
Q	P	0.5	0.3	1.2	0.8	-0.7	-2.71	1.84
I	J	0.2	0.3	0.7	0.5	-0.5	-2.74	1.45
N	J	0.0	0.0	0.3	0.3	-0.3	-2.85	1.92
Q	Z	0.1	0.0	0.7	0.7	-0.6	-2.85	1.92
W	I	0.8	0.3	1.8	0.9	-1.0	-3.16	0.88
M	T	0.3	0.0	2.0	1.6	-1.7	-3.44	0.74
L	I	0.4	0.0	0.8	0.3	-0.4	-3.79	0.43
N	A	0.1	0.0	0.5	0.3	-0.4	-3.79	0.43
W	J	0.2	0.0	1.2	0.7	-1.0	-4.74	0.11
G	A	0.3	0.0	0.8	0.3	-0.5	-4.74	0.11
W	V	0.7	0.7	2.8	1.2	-2.1	-4.98	0.02
M	A	0.8	0.8	3.3	1.3	-2.5	-5.06	0.01
W	U	0.4	0.0	1.2	0.3	-0.8	-7.59	0.00

Figure 12. The impact of the new adaptive segmentor on uppercase recognition.

On the other hand, there are considerably more confusion pairs contributing to degradations than there are for digits. It is observed that most of the characters being substituted in the lower portion of the bottom table contain three or more large strokes such as with W's, M's, and N's. These characters push the statistical limits of the multiple character detector, and therefore tend to be unnecessarily split. For example, when a large 'W' is split, the resulting left piece will likely be confusable with a 'U' or a 'V' and the right piece will be scored as an insertion. Both of these confusion pairs are flagged as significant degradations in the table. While the greedy splitting of uppercase characters cuts the deletion errors by 60%, insertion errors are somewhat offset (only 33%) and substitutions remain high. Perhaps the greedy splitting of large characters could be compensated for by developing a specialized multiple character detector just for uppercase characters.

Lowercase Recognition			
	Old Seg.	**New Seg.**	**Improv.**
Character Accuracy	75.3% $(\frac{9591}{12740})$	76.6% $(\frac{9763}{12740})$	1.3%

Lowercase Errors			
	Old Seg.	**New Seg.**	**Improv.**
Substitutions	2907	2747	5.5%
Insertions	404	152	62.4%
Deletions	242	230	5.0%

Pair		**System 1**		**System 2**		**Mean Δ**	**Student's *t***	
r	**w**	μ_1	σ_1	μ_2	σ_2	μ_1-μ_2	***t***	$\rho\times10^2$
i	l	28.3	2.4	19.9	2.7	8.4	7.37	0.00
k	l	1.7	0.8	0.7	0.3	1.0	3.59	0.38
p	l	0.8	0.5	0.3	0.0	0.5	3.35	0.85
j	v	0.7	0.5	0.2	0.0	0.5	3.35	0.85
i	c	0.6	0.5	0.1	0.0	0.5	3.35	0.85
y	l	2.7	0.9	1.6	0.7	1.1	3.15	0.60
g	j	0.6	0.3	0.3	0.0	0.3	2.85	1.92
i	j	0.3	0.3	0.7	0.3	-0.4	-2.68	1.52
i	g	0.0	0.0	0.3	0.3	-0.3	-2.85	1.92
w	j	0.1	0.0	0.4	0.3	-0.3	-2.85	1.92
e	v	0.1	0.0	0.4	0.3	-0.3	-2.85	1.92
i	z	0.0	0.0	0.7	0.6	-0.7	-3.83	0.40

Figure 13. The impact of the new adaptive segmentor on uppercase recognition.

The last set of results shown in Figure 13 are from the lowercase alphabets on the 490 HSF forms. As with uppercase, there is only a modest improvement in character recognition accuracy of 1.3%. Looking at the second table, insertion errors are dramatically reduced by 62% while substitutions and deletions show smaller improvements of about 5%. This reduction in insertions errors can be explained in part by analyzing the confusion pairs in the bottom table. The pairs that contribute significantly to the improvement of the system are substituted characters such as dotted 'i' and 'j' and the characters 'k' and 'y' which at times are written with small disconnected strokes. This testifies to the new segmentor's ability to compose characters from small fragmented strokes, such as the proper locating and merging of dots to their respective character bodies. The old segmentor did not merge these small pieces and often they were sufficiently large to be classified as single characters which in turn inflated the number of insertion errors. As can be seen from the middle table, the new segmentation method is quite effective in minimizing these types of errors.

Several observations can be drawn from the lowercase confusion pairs contributing significantly to the degradation of the system when using the new segmentor. Notice that i's are now confused more frequently with j's. This represents natural ambiguity between these two handprinted letters. By preserving the dots on these characters, the ambiguity has been enhanced. The remaining degradations are harder to explain. For example, it is suspected that the confusion of i's with z's occurs because the classifier has been insufficiently trained on dotted i's. Remember that the classifier was not retrained between the old and new systems. When spatially normalized, a dotted 'i' will scale more harshly than one without a dot, especially if the dot is relatively distant from its body. By retraining the classifier on more harshly normalized i's, the frequency of confusing i's with other characters is expected to diminish. Looking at the new segmentation method's errors in the middle table, it again appears that much of the recognition burden lies with the character classifier.

In regard to efficiency, the new statistically adaptive segmentor was timed on digit and alphabetic fields extracted from the first 50 HSF forms in SD19. Thirty fields from each form were processed (28 digit fields and 2 alphabetic fields) totaling 184 characters (130 digits, 27 lowercase, and 27 uppercase). In all, 1500 fields accounting for 9200 characters were segmented and timed on a Sun Microsystems SPARCStation 2 with a 80MHz Weitek CPU upgrade. Ignoring the time to read the images from file and writing out results, and only measuring the time required to conduct the segmentation, it took 25.4 seconds to segment the 1500 field images, yielding an effective throughput of 362 chars/sec. Analyzing the execution profile of the segmentor shows at least 50% of the processing time is dedicated to finding connected components. This suggests that the new segmentation method with all of its added capabilities requires not quite twice the time of the old method. These results demonstrate that the new statistically adaptive segmentor remains reasonably fast.

8. CONCLUSIONS

A new adaptable segmentation method for handprinted characters has been presented that builds upon the utility of connected components. Results demonstrate that modeling the writing style of handprint with simple statistical features (stroke width and character height) can significantly improve character segmentation. The method capitalizes on the knowledge that letters and numbers are constructed from a predetermined configuration of a relatively small number of strokes. By adapting to various styles of handprint, fuzzy rules are able to compose characters from disconnected strokes and fragments, split multiple touching characters, and effectively deal with noise. At the heart of the segmentor is an efficient, adaptable, and reliable method for detecting multiple touching character that uses aspect ratio and standard stroke count features that adapt to a writer's style at the field level. Recognition-based and oversegmentation methods fail to take advantage of these intrinsic qualities of handprinted characters. The impact of the new adaptable segmentor on character recognition was evaluated by integrating the new method into the NIST public domain Form-Based Handprint Recognition System and then testing the modified system on a set of 490 Handwriting Sample Forms found in *NIST Special Database 19*. Although no linguistic postprocessing is available for oversegmentation methods when processing numeric fields, modeling the writing style enabled the new segmentor (when compared to a simple component-based segmentor) to improve the overall recognition of handprinted digits by 3.4% and field level recognition by 6.9%, while effectively reducing deletion errors by 82%. The same program code and set of parameters improved the recognition of uppercase and lowercase characters by 1.7% and 1.3% respectively. The segmentor maintains a relatively straight-forward and logical process flow avoiding convolutions of encoded exceptions as is common in expert systems. As a result, the new segmentor operates very efficiently, and a rate of 362 characters per second was achieved. Analyses suggest that much of the burden for improving the performance of the system from this point relies on improving character classification. The source code for this new adaptable character segmentor is expected to be included in the next release NIST's public domain handprint recognition system. This system is available free of charge on ISO-9660 CDROM by sending a letter of request to the author.

REFERENCES

[1] G. Nagy, "At the Frontiers of OCR," Proceedings of the IEEE, vol. 80, no. 7, pp. 1093-1100, July 1992.

[2] R. A. Wilkinson, J. Geist, S. Janet, P. J. Grother, C. J. C. Burges, R. Creecy, B. Hammond, J. J. Hull, N. J. Larsen, T. P. Vogl, and C. L. Wilson, "The First Census Optical Character Recognition System Conference," Technical Report NISTIR 4912, National Institute of Standards and Technology, July 1992.

[3] J. L. Blue, G. T. Candela, P. J. Grother, R. Chellappa, and C. L. Wilson, "Evaluation of pattern classifiers for fingerprint and OCR applications," *Pattern Recognition*, vol. 27, no. 4, pp. 485-501, 1994.

[4] O. M. Omidvar, J. L. Blue, and C. L. Wilson, "Improving Neural Network Performance for Character and Fingerprint Classification by Altering Network Dynamics," Proceedings of The World Congress on Neural Networks, July 1995.

[5] C. L. Wilson, J. L. Blue, and O. M. Omidvar, "The Effect of Training Dynamics on Neural Network Performance," Technical Report NISTIR 5696, National Institute of Standards and Technology, August 1995.

[6] M. D. Garris and S. A. Janet, "Scoring Package release 1.0," *NIST Special Software 1*, vol. SP, National Institute of Standards and Technology, Oct. 1992.

[7] J. Geist, R. A. Wilkinson, S. Janet, P. J. Grother, B. Hammond, N. W. Larsen, R. M. Klear, M. J. Matsko, C. J. C. Burges, R. Creecy, J. J. Hull, T. P. Vogl, C. L. Wilson. The Second Census Optical Character Recognition Systems Conference. Technical Report NISTIR 5452, National Institute of Standards and Technology, May 1994.

[8] M. D. Garris and D. L. Dimmick, "Evaluating form designs for optical character recognition," Technical Report NISTIR 5364, National Institute of Standards and Technology, Feb. 1994.

[9] A. Gillies, D. Hepp, R. Rovner, M. Whalen, "Handwritten Text Recognition System for Processing Census Forms," Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics, vol. 3, pp. 2335-2340, October 1995.

[10] P. D. Gader, M. Mohamed, and J. H. Chiang, "Segmentation-Based Handwritten Work Recognition," Proceedings of the USPS Advanced Technology Conference, Washington, DC, 1992.

[11] F. Kimura, M. Shridar, and N. Narasimhamurthi, "Lexicon Directed Segmentation-Recognition Procedure for Unconstrained Handwritten Words," Proceedings of the Third International Workshop on Frontiers in Handwriting Recognition, Buffalo, NY, 1993.

[12] G. Martin, M. Rashid, D. Chapman, and J. Pittman, "Learning to See Where and What: Training a Net to Make Saccades and Recognize Handwritten Characters," Advances in Neural Information Processing Systems 5, Morgan Kaufmann, pp. 441-446, 1993.

[13] J. Keeler and D. E. Rumelhart, "A Self-Organizing Integrated Segmentation and Recognition Neural Net," Advances in Neural Information Processing Systems 4, Morgan Kaufmann, pp. 496-503, 1992.

[14] O. Matan, C. J. C. Burges, Y. LeCun and J. S. Denker, "Multi-Digit Recognition Using a Space Displacement Neural Network," Advances in Neural Information Processing Systems 4, Morgan Kaufmann, pp. 488-495, 1992.

[15] J. Rocha, B. Sakoda, J. Zhou, and T. Pavlidis, "Deferred Interpretation of Grayscale Saddle Features for Recognition of Touching and Broken Characters," Proceedings of Document Recognition, SPIE, vol. 2181, San Jose, CA, pp. 342-350, February 1994.

[16] V. Govindaraju and S. N. Srihari, "System for Reading Handwritten Documents," Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics, vol. 3, pp. 2347-2352, October 1995.

[17] S. Madhvanath, "Holistic Lexicon Reduction for Handwritten Work Recognition," Proceedings of Document Recognition III, SPIE, vol. 2660, San Jose, CA, February 1996.

[18] M. D. Garris, J. L. Blue, G. T. Candela, D. L. Dimmick, J. Geist, P. J. Grother, S. A. Janet, and C. L. Wilson, "NIST form-based handprint recognition system," Technical Report NISTIR 5469, National Institute of Standards and Technology, July 1994.

[19] D. F. Specht, "Probabilistic Neural Networks," *Neural Networks*, Vol. 3(1), pp 109-119, 1990.

[20] P. J. Grother, "Handprinted Forms and Character Database, *NIST Special Database 19*," Technical Report and CD-ROM, National Institute of Standards and Technology, March 1995.

[21] G. W. Snedecor and W. G. Cochran, *Statistical Methods*, 8th Edition, pp. 53-57, Iowa State University Press, Ames Iowa, 1989.

[22] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes, The Art of Scientific Computing (FORTRAN Version)*, pp. 466-467, Cambridge University Press, Cambridge, 1989.

APPENDIX A. FUZZY RULES AND ADAPTABLE OBJECTS

A.1 Is Noise?

If (c.a < (0.5 × *ssa*) then Noise

where structure member (a) is the pixel area of the component (c) and *ssa* is the pixel area of a standard stroke width.

A.2 Is Dot?

If (c.w < (2 × *esw*)) && (c.h < (3 × *esw*)) then Dot

where structure member (w) is the pixel width of the component (c) and *esw* is the estimated stroke width. This allows a small diagonal stroke to be considered a dot. Handprinted dots are seldom square, but they are typically a small tick-mark.

A.3 Horizontally Overlapping Merge Candidates

If (c1.x1 <= c2.x1 && c2.x1 <= c1.x2) || (c2.x1 <= c1.x1 && c1.x1 <= c2.x2) then Horizontal Overlap

where structure members (x1) and (x2) are the left and right edges of the component in its parent image. If exactly one component is found to overlap with the piece being tested, then it is selected as a merge candidate. If exactly two components overlap, then the component to the left is chosen as the merge candidate with the following bias

If (right overlap distance > (1.6 × left overlap distance)) then Pick Right else Pick Left

where left and right is determined according to each component's origin position in its parent image. This bias was imposed because it was observed that, more often than not, a character piece belonged with its overlapping neighbor on the left than on the right. This probably has to do with the fact that English is written left-to-right. If per chance more than two component are found to overlap the piece, then the component with maximum overlap is selected as the merge candidate.

A.4 Merge Candidates Compatible?

If (both very tall && both very close in height) || (both tall && both horizontally far apart)
then Not Compatible

where the compatibility tests for components (c1) and (c2) are

very tall :	(c.h ≥ (0.6 × *ech*))
both very close in height :	((min(c1.h, c2.h) / max(c1.h, c2.h)) > 0.9)
tall :	(c.h ≥ (0.3 × *ech*))
both horizontally far apart :	(abs(c1.cx - c2.cx) ≥ (0.3 × *ech*))

and the structure member (cx) is the x-coordinate for the center of the component and *ech* is the estimated character height. These test are performed to see if the candidate components are too big to be merged.

A.5 Is Inside?

If (c1.x1 ≥ c2.x1) && (c1.x2 ≤ c2.x2) && (c1.y1 ≥ c2.y1) && (c1.y2 ≤ c2.y2) then c1 Inside c2

where structure members (y1) and (y2) are the top and bottom edges of the component in its parent image. This conditional statement tests to see if component (c1) is fully inside component (c2). In other words, is the first component's bounding box completely contained within the second's bounding box. As a general rule, if one component is completely inside another, then they should be merged together.

A.6 Is Top of i or j?

If (c1.y2 < c2.y1) then c1 Above c2

determines if component (c1) is above component (c2). Just because a component has been determined to be dot-sized, does not mean that it is actually a dot. It may be a character fragment or noise. To be a dot, it must be above its neighbor. Here, left and right neighbors are determined according to their center x-coordinates.

If (dot to neighbor-line distance < (2 × *esw*)) then Dot is In-Line

tests to see if the dot-sized component is sufficiently in-line with its neighbor. A line is computed connecting the left-most black pixel in the bottom row of the component's image to the left-most black pixel in the top row. A distance is then computed between the center point of the dot to the perpendicular intersection of the line projecting from it neighbor. This facilitates the matching of dots to characters such as 'i' and 'j' that are written at a slant as illustrated in Figure 14. If the component qualifies as a dot for both its left and right neighbors, then the neighbor whose top is closer to the dot is selected. Where the distance from neighbor (n) to dot (d) is computed between the top point of n (n.cx, n.y1) and the center point of d (d.cx, d.cy).

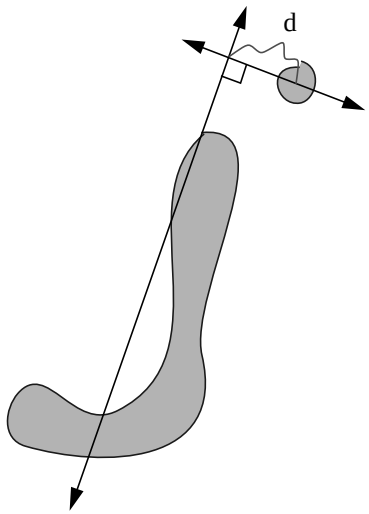


Figure 14. Illustration of the dot to neighbor-line distance (d).

A.7 Is Top of 5?

If (top is shorter && not too far right && not too far left && not too far down && dash-like) then Top of 5

where the tests for top component candidate (t) with left neighbor component (n) are

shorter :	(t.h < n.h)
too far right :	((t.x1 - t.x2) < min((t.w × 0.5), (n.w × 0.5)))
too far left :	((n.x1 - t.x1) < min((t.w × 0.5), (n.w × 0.5)))
too far down :	((t. y2 - n.y1) < (n.w × 0.5))
dash-like :	((t.p / esw) < (t.l + *esw*))

and the structure member (p) is the black pixel count of the component and (l) is the diagonal length of the component. The top horizontal stroke of a 5 is frequently written so that it is detached from its body. This stroke is dash-like, meaning the component is comprised of a single horizontal stroke that spans the entire width of the image. The height of the stroke should be uniformly close to the estimated stroke width, so dividing the black pixel count in the component by the estimate stroke width should be very close to the diagonal length of the component (at least within a stroke width).

A.8 Caused Fragments?

If (original dominant blob count ≠ (left dominant blob count + right dominant blob count - 1))
then Fragments Exist

where *dominant* implies a connected component (blob) larger than noise or dots. All the dominant blobs in the original component image are counted, as are all the dominant blobs in the left and right pieces. These are the pieces resulting after a splitting of the original image. If the segmentation is satisfactory, then the split should divide only one of the dominant blobs in the original image. Therefore, the number of dominant blobs resulting after the split should be one greater than those in the original component image.

A.9 Estimated Character Pair

ecp = (1.5 × maximum single character width)

where the maximum single character width is computed by evaluating successive vertical cuts of the component left-to-right until the resulting feature point of the left piece exceeds (goes above) the detector line. This represents the largest single character possible based on the statistics of the handwriting. The resulting x-position is multiplied by the factor of 1.5 accounting for the likelihood that two characters in a row will be less than twice the maximum width for a single character.

A.10 Small Punctuation

If (c.p < (0.5 × *ssa*)) && (not one-like) then Small Punctuation

determines if a component is the size of a small punctuation mark, and the test for one-like is

If ((c.h > (0.4 × ech)) && ((c.p / esw) ≤ c.l) then One-Like

If a component is one-like, then it will in general not be too short, and it will be comprised of a single vertical stroke that spans the entire height of the image. The width of the stroke should be uniformly close to the estimated stroke width, so dividing the black pixel count in the component by the estimated stroke width should be very close to the diagonal length of the component.