Automating Forensic Graphology

New Mexico
Supercomputing Challenge

Final Report

March 28th, 2015

Team #122

Sangre de Christo Homeschool

Team Member:

André Green

Project Mentor(s):

Monique Morin
Andrew Green

Automating Forensic Graphology

Executive Summary:

Forensic graphology, or hand writing analysis, was first recorded in third century Rome and by the sixth century the Emperor Justinian had dictated guidelines for the use of handwriting comparison in Roman courts. For much of its history forensic graphology has been subjective and unquantified, leading to a number of injustices. Modern analysis includes a number of techniques from the physical sciences but they are financially and otherwise inaccessible to most people. Computers, on the other hand, are easily within reach of many private citizens.

This project explores techniques for comparing images of alphabetic letters as a first step toward quantifiable digital handwriting analysis. The aim was to compare two arbitrarily positioned and oriented samples of the same letter. In preparation for comparison, the sample of unknown origin was translated through cropping then rotated by matching the aspect ratio to that of the known sample.

After alignment of the images, I explored two techniques for comparing individual letters. The pixel method developed a goodness-of-fit metric based on pixel intensity. The vector method produced histograms of slopes providing signatures for each letter. A goodness-of-fit metric was based on normalized histogram channel differences.

Display graphics were developed for the vector method to aid the user in understanding the model through seeing the effects of adjusting slope and range selectors.

The entire project was written in Java using NetBeans as the Integrated Development Environment. Some images were generated using the GNU Image Manipulation Program (GIMP)

Problem defined:

Forensic graphology is handwriting analysis. It can be used to identify a forgery by disqualifying a piece of writing as not being written by a specific author. Forensic graphology can also be used to authenticate a piece of writing as being written by a specific author. In both, the shapes of letters and words in a written piece of unknown origin are compared to one or more pieces of known origin

This project is not trying to identify or disqualify authorship by analyzing word-choice, spelling, or grammatical style.

This project is specifically a comparison between two images. In this application they are of alphabetic letters, but the techniques explored here could be used for more general shape analysis in other images. It should also be clear that these techniques could be applied to other alphabets (e.g. Sanskrit or Cyrillic), even when the user is unfamiliar with that alphabet.

Once comparison techniques have been developed for any letter, they can be applied seamlessly to any other letters of an alphabet. With that accomplished one can then analyze a full written script by viewing it as a linear collection of individual letters. With the same techniques one could also analyze entire words as single images and extend this to an entire passage with words instead of letters. Or, more likely, one would use a combination of common words and individual letters.

The object of this project is to develop quantitative metrics for comparing two individual letter images. These metrics can then be measured for different comparisons. The metrics would be equal to unity for a perfect match only, and zero for no overlap at all.

Approach:

<u>Create a sample suite</u>

For a project primarily involving letter image comparisons, it is important that the first step be to create a collection of sample images. Most of my work was based on the uppercase letter ‘A’. I made samples based on available computer fonts as well as a number of handwritten samples using GNU Image Manipulation Program (GIMP) with a drawing tablet (see Figure 1). For simplicity I converted all images to an 8-bit gray scale, with the darkest parts of the letter having a bit value of 0 and the background being

white (bit value = 255). I did not opt for a binary color assignation (where pixel is either black (0) or white (255) only) since this approach would lose the nuance of writing pressure.

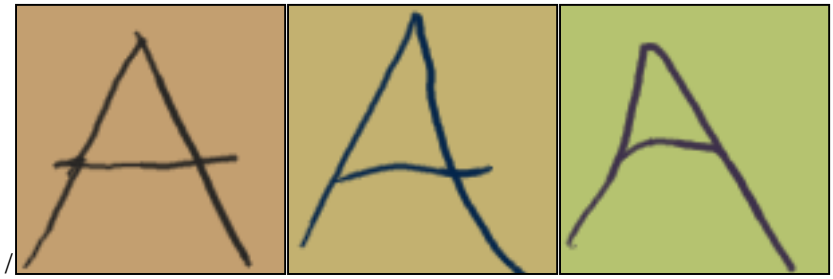

Figure 1. Various handwritten samples of the letter ‘A’ made using GIMP.

To allow for testing of barely visible effects, the suite I generated included slight variations of other samples I made. Figure 2 shows a sampling of font generated letters in the suite. The first is a normal ‘A’; the second is the normal ‘A’ rotated by 30 degrees; the third is the rotated version with barely detectable (to the reader) modifications.

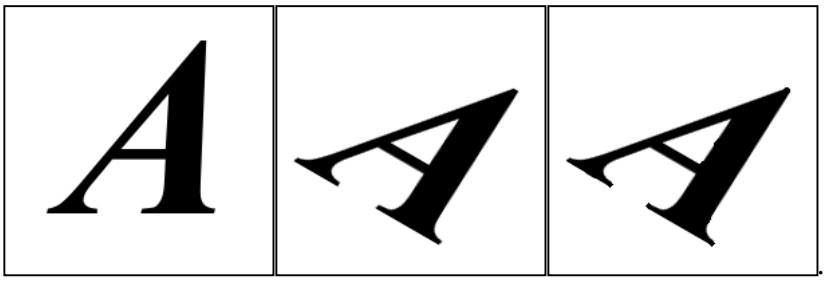

Figure 2. Variations of a standard font ‘A’ including very nuanced changes.

From the suite one sample is designated as the ‘unknown authorship’ (‘unknown’) sample. It is then compared to many other ‘known’ samples to see how well a match is obtained.

All the images are stored as 2-dimensional matrices with the matric element value, G_{ij}, being the grayscale value of the pixel in row i and column j.

I developed two methods for image comparison. The first is a pixel comparison; the second is a vector based slope comparison. The methods require different preparatory image manipulation so I will discuss each method separately from this point.

Method 1: Pixel comparison method

Preparatory image manipulation for pixel method

To compare two arbitrarily sized, oriented and positioned images one would need to translate, rotate and resize at least one of the images. I chose to manipulate the unknown image leaving the known image alone. The pixel method requires all three corrections. I handled translation by identifying the left-most, right-most, top-most and bottom-most non-background pixels and cropped the image.

To prepare for the rotation I copied the cropped image into a matrix of dimension equal to the diagonal length of the original matrix, essentially adding minimum sized buffer of white space around my original image. This is done so that when I perform the rotation I will not lose any part of the original letter.

Next I rotated the image in 1 degree steps re-cropping after each iteration. I recorded the height and width of the new matrix each time. After stepping through the entire range I looked for the best height to width ratio match with the known sample. This would be the rotation solution. This results in a patchy rotated image because continuous trigonometric functions don't fit smoothly into discrete positions, especially in small arrays. My solution to this problem was to interpolate by changing a white pixel to black if it had at least one black neighbor. I found this worked better than averaging the 8 nearest neighbors.

Allowing rotation all the way around to 360 degrees would produce more than one solution (correct height-to-width ratio) – including inverted letters. By restricting the range of rotation from vertical to 45 degrees to either the left or right one limits the number of possible wrong solutions. I chose this range to match the range to accommodate the slopes of most handwriting.

At this point I had a correctly positioned and rotated image. The letters may, however, not be the same size, meaning the matrices are different sizes. Resizing seemed to me to have a number of difficulties which I thought I would not be able to solve in the time I had. I have looked in the literature and seen that my assessment that this was a tricky problem was correct. I decided to avoid this issue by insisting my initial images be the same size. So, after translation and rotation I had two matrices of the same size (even if they were not square. I could now compare the matrices pixel by pixel (i.e. element by element).

Comparison metric for pixel method

I explored a number of different comparison metrics. For discussion's sake let the known pixel, p_{ij}, have the value 84. The first metric was of a binary nature: I scored a point if the gray scale values matched exactly (e.g. if value = 84 score 1 else score -1) but this was too sensitive a metric so I modified it to being a match if both pixels fell in the same half of the entire range (e.g. if value on [0;127] score 1 else score -1).

My second metric was based on the difference between the gray scale values at each pixel. Close values would count more than very different values. The exception being that if both the 'unknown' and 'known' pixels were white (background) I excluded them from the average. This places the emphasis, or weighting, on the portions of the images where the letter is. The match sensitivity is not diluted by the background matching (which is typically considerable).

I further refined the metric based on the difference by assigning weighting factors. I tried a number of different weighting factor maps. My first version was based inversely on the distance (r^{-1}) of the pixel from the center of mass of the letter (calculated from non-background pixels only) but I found the drop off too rapid. My second version allowed the user to choose the exponent of the distance dependence (r^{-e}) and I found e = 0.5 to be a satisfactorily gentle drop off at the larger distances. However, I preferred not to discriminate between points moderately near the center of mass, so implemented a cap to the weighting factors inside a specified radius. Table 1 shows variations in exponent and cap radius (in pixels). For comparison the metric was 91.39% without any weighting applied. This third map had the benefit of allowing the user to decide how much of the letter should be weighted fully and how fast the drop off in weighting should be beyond that. Note that in the table the best fit is obtained for a specific radius (5 pixels) and drop off exponent (0.5) not values would be easily predicted by the user.

Table 1. Fit metric (in percentage) for pixel method as function of exponent and cap radius.

	radius=2	radius=5	radius=10
exp = 0.3	91.37	91.39	91.39
exp = 0.5	91.2	91.44	91.39
exp = 1.0	90.0	90.4	91.1
exp = 2.0	65.7	71.5	76.4

Another weighting map idea I tried was to blur the ‘known’ image and base a weighting map on the blurred image. This attributes importance to where it matters most **for the ‘known, sample** on hand.

The code has the option to choose between these weighting map methods and can easily allow the programmer to change associated parameters.

Finally, I should point out that I chose to use the complement of the pixel grayscale difference, rather than the grayscale difference itself, since I wanted a metric that would be non-zero for a good match. Since my final metric was a ratio of current match to a perfect match I needed this non-zero denominator.

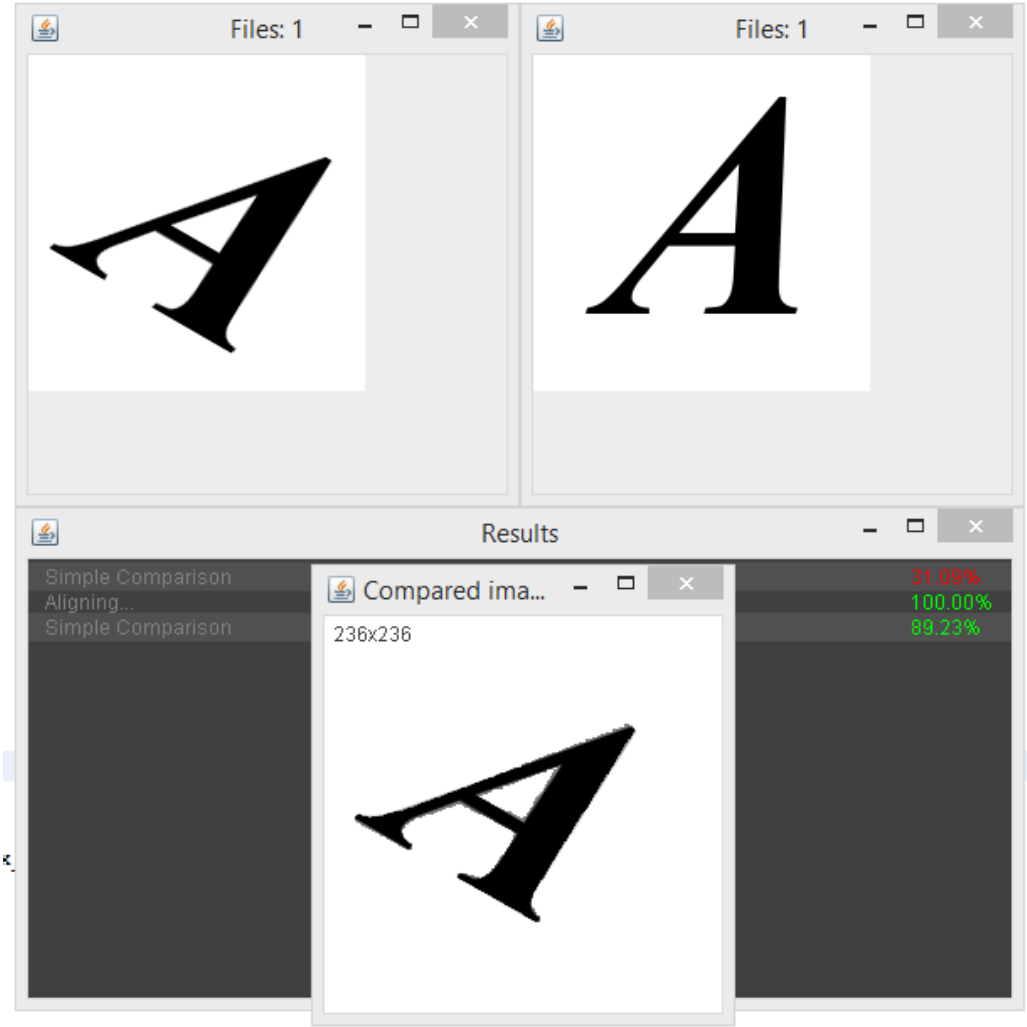

Figure 3. Comparison of an original and a rotated sample. The ‘Compared images’ window shows overlap as black, non-overlap as gray. Red score is before rotation; lower green score after rotation.

Results and observations for pixel method

Figure 3 shows a comparison of an original to a rotated version of itself. The comparison metric used here was based on the grayscale value difference with no weighting map applied. The red value is the percent match before rotation, the lower green value is the match after rotation. The original and rotation corrected versions are shown superimposed in the 'Compared images' window. Notice that the rotation correction is good but not perfect. Firstly, to save time I stepped through my angular range in one degree steps. If the rotated image was 37.5 degrees rotated relative to the original it cannot be undone by an integral number of 1 degree steps. Secondly, another source of error is discretizing trigonometric functions.

If I had much more computational power I would abandon the height-to-width comparison in favor of a direct image comparison looking for metric improvement trends as a guide to local minima. Or alternatively I would use the height-to-width comparison as a rough guide to focus on promising angles before doing direct image comparisons in smaller angular steps.

I noted that translations were more easily corrected but, like rotation, are limited by discretization of movements. All discretization effects are reduced if larger images are used but increased image size calls for significantly more storage and more computation time. I used images no bigger than 300x300 to keep space usage (less important) and processing time (more important) manageable.

Method 2: Vector comparison method

In comparing images of letters I developed a second, completely different method for characterizing the letter and then analyzing the data. This method is based on generating a set of gradients for the letters. I then compared the gradient sets for the 'known' and 'unknown' samples.

Preparatory image manipulation for vector method

The vector method has the advantage, over the pixel method, of requiring less preparatory image manipulation. Since the method is based on gradients translation and resizing are unnecessary. Preparatory rotation correction is helpful but not necessary as rotation can be done relatively easily later.

What does need to be done prior to comparison for this method is to generate a set of gradients associated with each letter. This is done by creating an axis-aligned square grid over the image and then recording the intersections between the grid lines and letter. There are two approaches to doing this.

The first is to find the grid intersections with the edges of the letter. This is done by comparing the grayscale values as one moves along a grid line. The grayscale values will decrease as one moves from the background (high grayscale value) into the darkened letter eventually reaching a minimum grayscale value. Values stay low until the grid lines moves back into the background. The edges for that grid line are defined as the boundaries of the letter stroke**.** These two edge points are stored before continuing along the gridline. A grid line may cross the letter more than once. The search then continues to the next gridline. Similarly identified values are recorded for all horizontal and vertical grid lines. This set of edge points forms the set of endpoints for calculating a set of vectors.

The second approach locates the grid line intersections with the spines of the letters. The spine points for each grid line are simply the midway points between the pairs of edge points that correspond to each crossing of the letter.

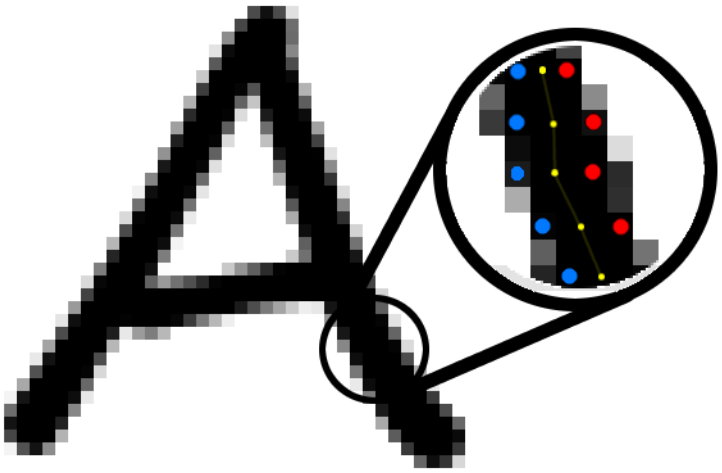

Figure 4. Method for determining the edges and spines of letters. Edge points (blue and red) are the locations of the grayscale halfway points (between background and letter colors). Spine points (yellow) are locations midway between edge points for each horizontal grid line.

Figure 4 shows an expanded view of a set of horizontal gridlines crossing a letter stroke. More edge points and more spine points are generated by vertical gridlines intersecting with the letter. The vertical and horizontal points are stored as one set.

There are only two sets: one of edge points and another of spine points. As with the edge points, a set of vectors is generated from the spine points.

Within each set, the vectors are formed between any pair of points. This leads to many vectors, some of which are not representative of the letter shape. Figure 5 shows two sample vector sets, one from edge points that concentrate along the outline of the letter is on the left; another from spine points on the right. It is not surprising that the highest number of uncharacteristic vectors occur at the endpoints, corners and intersections.

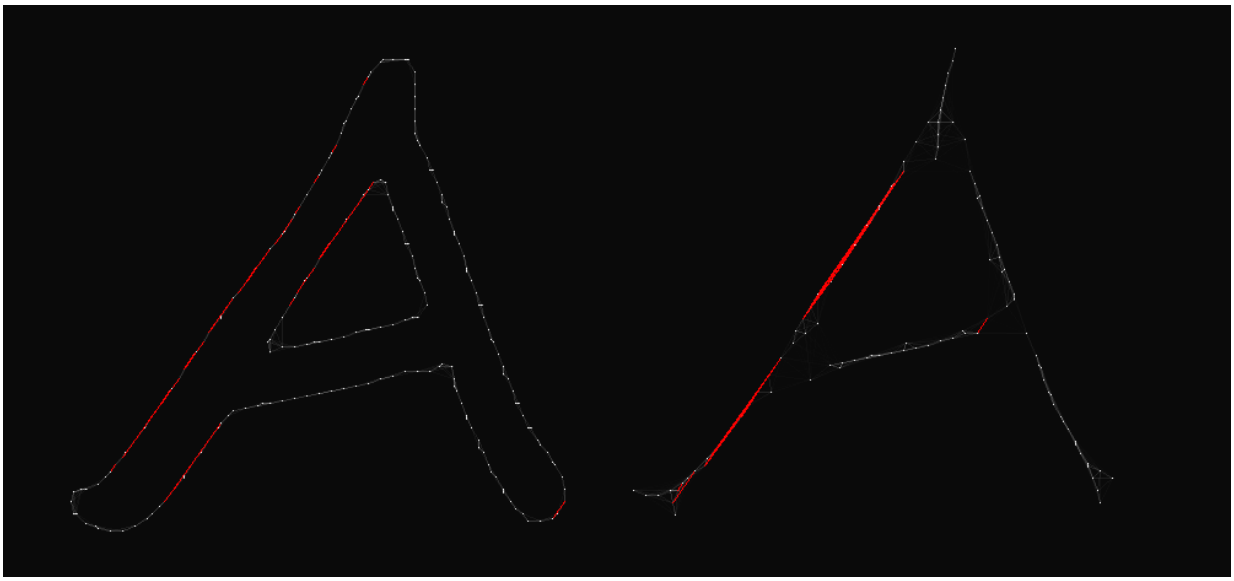

Figure 5. Examples of vector sets, one from edge points (left) the other from spine points (right). The red lines are only those vectors at a specified angle around 50 degrees relative to the horizontal.

To reduce the number of uncharacteristic vectors a connectivity range was set on how long any vector could be. However, the range can be set too short for any vectors to be formed. Even if one makes the range long enough to form vectors, the shorter vectors can still be uncharacteristic. I varied the connectivity range parameter to get the highest ratio of characteristic to uncharacteristic vectors. See Figures 6, 7 and 8 for three choices of connectivity range. The ratio of the peaks to the background for the letter ‘K’ is largest for smallest connectivity range but the statistics for each channel are poorer (lower number of vectors). The largest connectivity range can clearly be seen to introduce uncharacteristic vectors by bridging from one stroke to another. The optimal choice of connectivity range will depend on the size and shape of the letter.

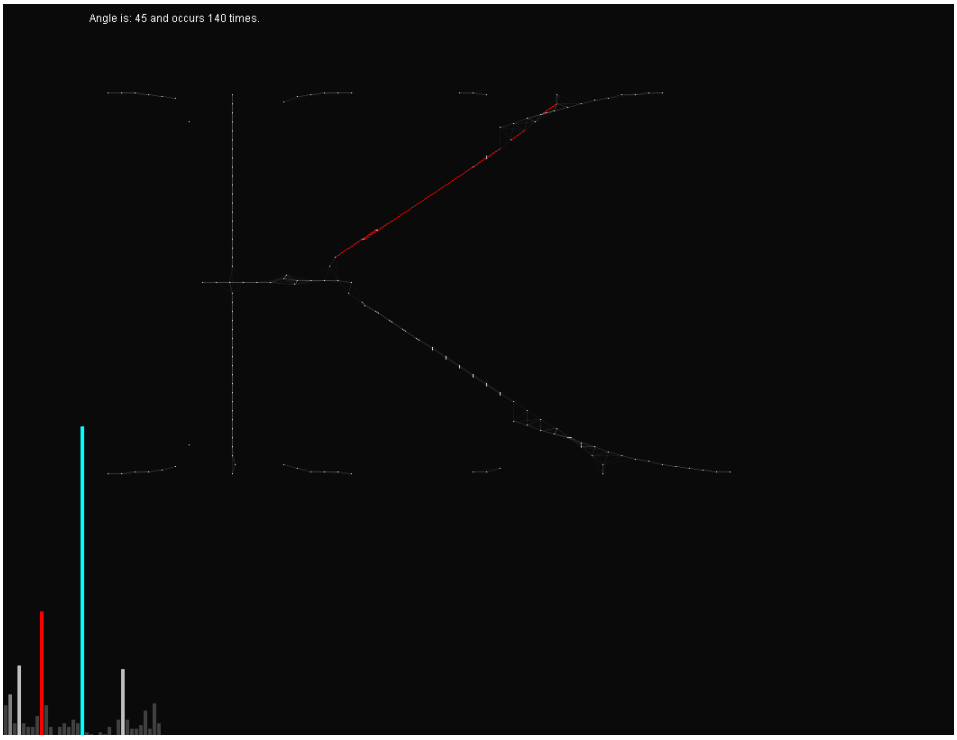

Figure 6. Spine vector set for small connectivity range.140 vectors at 45 degrees highlighted in red.

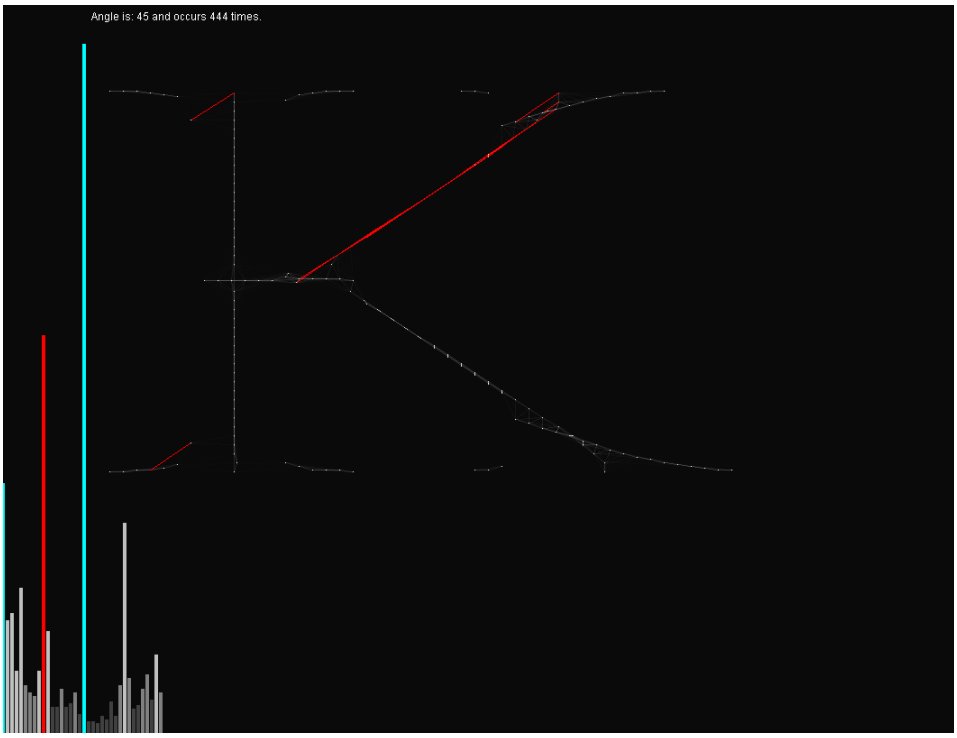

Figure 7. Spine vector set with medium connectivity range. 444 vectors at 45 degrees highlighted in red.

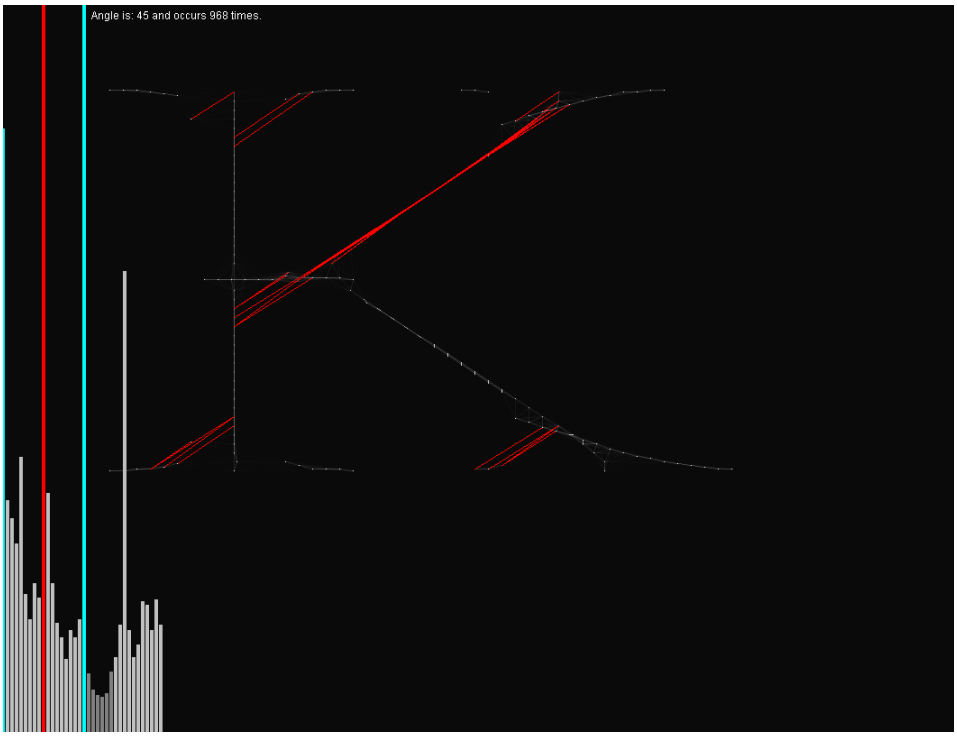

Figure 8. Spine vector set for large connectivity range. 968 vectors at 45 degrees highlighted in red.

To guide me in my explorations of the connectivity range (and as a diagnostic tool while I wrote the code) I built a live display window showing all the vectors within the connectivity range specified, superimposed on the letter image.

The next stage towards generating a metric for goodness-of-fit involved histogramming the gradients of the vectors in the set. I then added this histogram to my display window. Figures 6, 7 and 8 show examples of the display window with the histogram in the lower left corner. The histogram starts at 0 (horizontal), and ends at 180 rather than 360 due to the duplicity of slope direction. The vertical axis is the number of occurrences of a vector with a specific gradient or slope. Notice that for this letter (uppercase ‘K’) which has three distinctive slopes (corresponding to the three strokes), there are three significant peaks in the histogram at about 45, 90 and 135 degrees. A fourth peak occurs at zero degrees corresponding to the horizontal portion in the middle of the letter.

An additional feature I added to the histogram display in the GUI was the highlighting of the bar corresponding to an angular channel which I could select from the keyboard. At the same time the contributing vectors in that angular channel are highlighted.

Any histogram can be shifted horizontally by adding a constant angle to all the data. This is a much simpler way of addressing letters that are rotated relative to each other than doing the matrix rotation on the image before applying the square grid and determining the edge and spine points.

Once the histograms were lined up to start at the same angle, I normalized both the spectra.

Comparison metric for vector method

The difference between the normalized spectra for the 'known' and 'unknown' samples was quantified by doing a channel by channel subtraction and finding the average difference. Basing the metric on this quantity has the advantage over the method in the next paragraph of allowing adjacent channel differences to cancel each other - which would lead to a less noisy metric. To keep the metric from going negative, the absolute value of the average difference is preferable.

Other variants on this difference idea include finding the average absolute difference where no channel differences can cancel others. This is a more sensitive basis for a metric for detecting differences so would be more appropriate for very similar images. One can imagine basing the metric on the difference to any exponent before summing for the average.

The metric I chose is based on the linear difference but since I want the metric to be 100% for a perfect match I used the complement of the absolute value of the average linear difference between normalized spectra. In normalized spectra with a maximum of unity the channel by channel difference, and also the absolute value of its average, cannot exceed unity or be negative. The complement is then also bound between 0 and 1.

A results display window indicates the value of the comparison metric expressed as a percentage.

Results and observations for vector method

Making comparisons with this method involve far less preparatory work than the pixel method since no translation, resizing or rotation of images is required. This method is a relative method rather than an absolute method. This means that different-sized and different-shaped images are easily compared.

The biggest issue with this method is the generation of vectors that are uncharacteristic of the letter. They

most often occur near discontinuities in the letter. Their contribution is in the form of noise, or background, in the histogram. This background clouds the identification of peaks. It also lowers the quality of the fit since noise is random so is not likely to match background (noise) in another histogram.

The connectivity range can be used to control the uncharacteristic vectors. Table 2 shows how varying the connectivity range affects the goodness-of-fit between two very similar images (the center and right images in Figure 2). Notice that the goodness-of-fit improves as longer vectors are allowed to be considered in the calculation. For comparison when viewing Table 2 it is useful to know that the method gave a 76% match in a ‘A’ to ‘B’ comparison (using connectivity range of 5 pixels).

Table 2. The effect of connectivity range on the vector goodness-of-fit metric.

Connectivity range (pixels)	Goodness-of-fit (%)
3	90.5
5	99.1
10	99.5

A finer grid will generate more points and therefore also more vectors but his does not necessarily make for a less noisy histogram or a better fit.

Peak search algorithms abound in spectroscopic sciences and one could use them to identify peaks in the histogram and determine the areas of the peaks. Matched peak areas and shapes would indicate a good fit. This would be an alternative approach to my difference metric but a lot more complicated.

Letters made up solely of straight lines such as ‘A’, ‘K’ and ‘L’ give histograms with narrow peaks. Curved letters such ‘C’, ‘O’ and ‘S’ give histograms with broad peaks. Letters with straight and curved strokes give a mix of narrow and broad peaks. Figure 9 shows histograms for the letters ‘A’ and ‘B’. Notice that the letter ‘A’ histogram is dominated by three peaks at 0, 50 and 90 degrees (the three strokes of the letter). The ‘B’ spectrum is dominated by peaks at 0 and 90 degrees but clearly has many vectors at other slopes corresponding to the curved portions.

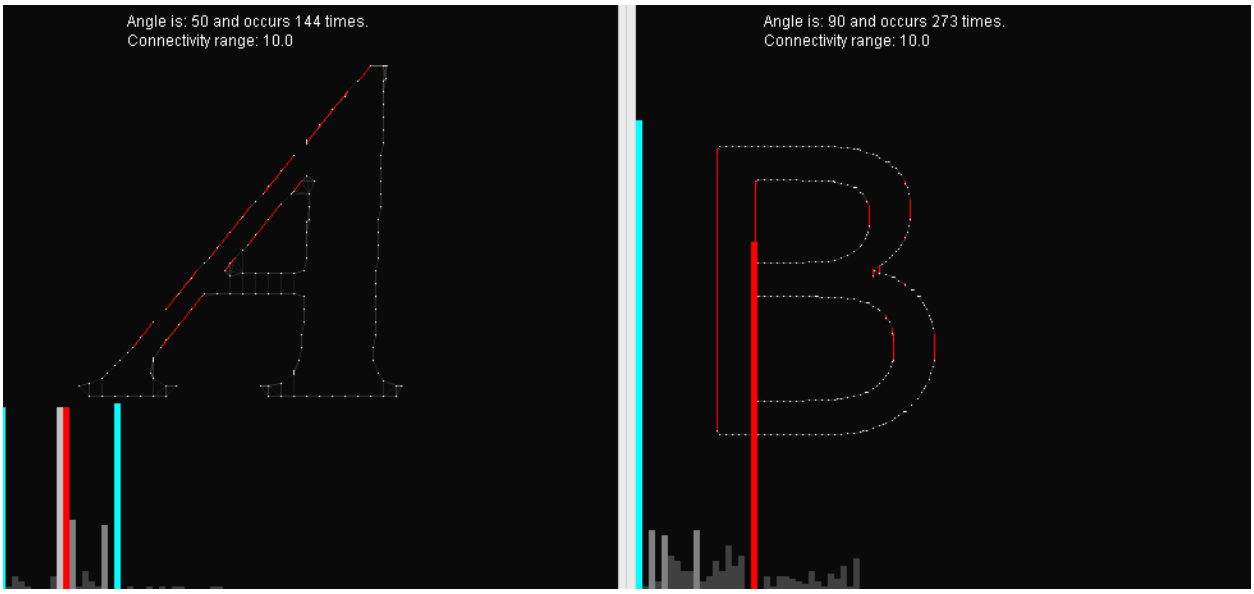

Figure 9. Histograms for edge point vectors for letter 'A' (left) and 'B' (right).

Concluding remarks:

This project was meant to take the first steps in a potentially much bigger project. I developed two methods for doing forensic graphology using software. They both have their merits and difficulties.

The first method was a pixel by pixel comparison. It needed substantial preparatory work to ready the images for comparison but once that was done became a very powerful method for comparison. I developed a goodness-of-fit metric and explored ways of optimizing it through the use of weighting maps.

The second method was a vector based approach in which sets of characteristic gradient vectors were generated and then histogrammed. Comparison of the histograms led to a goodness-of-fit parameter. This method needs little preparatory work on the images and so is much easier to do. The noise in the histograms generated by uncharacteristic vectors is the biggest weakness but is manageable enough to make this still a useful method. Another weakness of this approach is that two letters with strokes at the same angles but in different relative locations (for example, 'X', 'W' and 'V' will be hard or impossible (if stroke lengths are equal) to distinguish. In such cases in forensic graphology we would be comparing different versions of the same letter rather than different letters so this becomes less of an issue.

In conclusion. I would like to thank both my mentors, Drs. Monique Morin and Andrew Green, for their help and guidance on this project.