Offline Signature Verification Using Critical Region Matching

Abhay Bansal, Bharat Gupta, Gaurav Khandelwal, and Shampa Chakraverty

Dept. of computer Science
Netaji Subhas Institute of Technology, Delhi University

abhaybansal.1988@gmail.com

Abstract

Signature can be seen as an individual characteristic of a person which, if modeled with precision can be used for his/her validation. An automated signature verification technique saves valuable time and money. The paper is primarily focused on skilled forgery detection. It emphasizes on the extraction of the critical regions which are more prone to mistakes and matches them following a modular graph matching approach. The technique is robust and takes care of the inevitable intra- personal variations. The results show significant improvement over other approaches for detecting skilled forgery.

1. Introduction

Signature Verification is the process of recognizing an individual's handwritten signatures. Signatures have been by far the most popular means for establishing the authenticity of individuals. Signature authentication offers a quick, simple and cost effective means for validating the authenticity of a document by determining the difference between an original signature and a counterfeit one.

The two most widely used approaches for the verification of the signatures are *static* and *dynamic*. Dynamic or Online Verification analyzes the behavioral biometric of the handwritten signature while it is being written with the aim of verifying its authenticity, thus establishing the identity of the user. The user's presence is vital for the real time verification of his/her signatures. In contrast, Static or Offline verification obviates the necessity of user's presence as it compares the various characteristics of a pre-recorded signature image on order to reach a desired conclusion

Unconventional writing styles and other factors such as mental state, illness, age etc. make the process of offline signature verification more complicated. On the other hand, Online Signature Verification can conveniently utilize a number of parameters associated with the stylus and electronic writing pad for determining the authenticity of the signature. These parameters include speed, direction, pressure of the stylus, number and order of the strokes, etc. These properties make the signature unique and nearly impossible to forge. However, it is not practical to implement, as a stylus and pad cannot be used everywhere for signature verification. For example, paper instruments such as checks and documents cannot use this method. Also, infrastructure costs are too high to implement a purely Online Signature Verification method.

The type of authentication method applied *i.e* offline or online can also vary depending on the type of application. For banks and other financial institutions, online signature verification is not feasible. When checks and other documents arrive for clearance at the banks' end, offline verification becomes mandatory, as the user is not present at the time. This requires a database of signatures to be present with the banks.

Check fraud detection is one of the largest challenges facing businesses and financial institutions today. IT contributes to the majority of the losses suffered by the bank. Fraudulent checks are too difficult to detect for skilled forgeries. However, with the presence of an offline method of authentication, this can be considerably reduced. Thus, of the two methods, Offline Signature Verification is more popular and widely accepted owning to its low costs and applicability.

Various other approaches [7][8][9][10][11][12] have been proposed for offline signature verification as However, most of these approaches provide good accuracy only for random forgeries, and to some extent, semi-skilled forgeries. A number of graph-based approaches have been proposed for automatic signature verification [2],[3]. These works have applied graph-matching approaches which compare the outer contour of the signatures based on the Hungarian method [7]. These approaches have two limitations: (1) they work on relatively small window sizes (32*64) and (2) they fail when the test signature is a superset of the original signature.

In this paper, we propose a graph-matching based automatic signature verification technique which is based on geometrical shape of the *critical regions of the signature*. Graph Theory finds extensive use in Image Processing areas such as Computer Vision, Pattern Recognition, and Segmentation as the comparison of two objects is reduced to the comparison of their respective graph representations [1]. This way, two abstract objects take mathematical forms that can be accurately compared by comparing the two graphs. The more similar the graphs are, the more similar are the corresponding objects. Once the problem is formulated into a relational structure, techniques in Linear Algebra, Statistics and Probability theory can be conveniently used to analyze the problem. Our method reduces the offline signature verification problem to a *modular* graph-matching problem, employing the Hungarian method to find the geometrical similarity. It scales down the complexity of Hungarian matching and precisely models different shapes in the signature to obtain a perfect match.

The rest of the paper is organized as follows. Section 2 describes signature preprocessing. Section 3 discusses in detail the major points of our proposed method that includes critical points extraction, critical region extraction, formulation of signature verification problem as a graph matching problem, training of the sample signatures and testing of the signature to be verified. Results are discussed in Section 4.

2. Preprocessing:

In the following, we adopt the following notations. Sets are denoted in bold and Cap face. Scalar quantities are denoted in regular (non-bold) font. Members of a set are denoted in small and bold letters of their respective sets, and are indexed. In addition, the set of x-coordinates and y-coordinates corresponding to a set of points are represented by subscripting the name of the set, *e.g.* X_A represents the set of X-coordinates of the point set A.

2.1. Binarization

All gray scale images are binarized with the help of modified Niblack algorithm [4]. This algorithm works very well for handwritten text images with complex backgrounds, as illustrated in figures 1a and 1b.

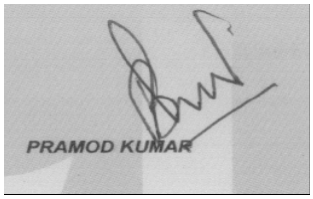


Fig.1a: Original image

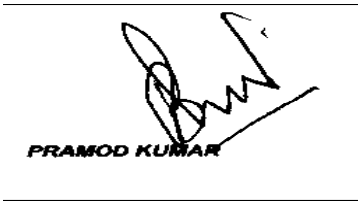


Fig.1b: Binarized image

2.2. Noise Removal

Once we have binarized an image, the noise components must be removed. Small components (pixel_size<5-10 pixels) are removed by using a simple morphological filter. Assumption that the signature content would be more prevalent in the image, the image is passed though a low pass filter to eliminate the low frequency noise components. The filtering is done by using 2-D convolution with a 5X5 Unity matrix. The results are illustrated in figure 2a and 2b. This process converts the binary image into a gray scale image. The image thus obtained is further binarized using a strict estimated threshold.

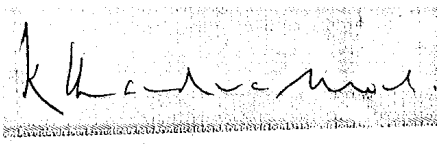


Fig. 2a: Noisy Image

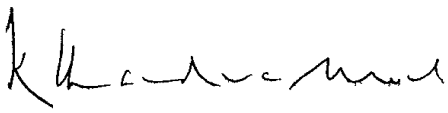


Fig. 2b: Image after removing noise

2.3. Rotation of Signatures

The accuracy of the results is largely dependent on the rotation algorithm used for orientation correction. The rotation algorithm should be robust and must produce the same results for images taken from the same user. The rotation algorithm ***rotate-image*** is given in Figure 3. The binarized and noise cleaned signature image is input to the algorithm. We use the bottom pixels of a signature image as a template to fit an *orientation line* through them using the *polyfit* function of Matlab® (Mathworks Inc Ltd) (lines 1-2). The *polyfit* function is further explained in Section 3.1. Finally, the *cp2tranform ()* function produces a projective transformation of the input image, using the slope of the orientation line as a guiding parameter (line 3).

Experimentally, we found that the above algorithm showed excellent parity between the rotated-corrected transformations of the sample signature and new input signature from the same user, when the rotation angle varied between -30 to +30 degrees.

2.4. Thinning of Signatures

The signatures are thinned in order to reduce the computations required by the graph matching algorithm. We employed the technique of Thinning by LOCAL coupling Points [5]. This algorithm works very well on signature images, as it is able to preserve their intricate details and other geometrical properties. Figure 6 shows the images before and after for a sample signature.

Algorithm rotate_image

Input: signature image *// Binarized and a noise cleaned signature image.*
Output: rotation corrected image
Rotation*(input_image)*
begin
1. Set ***A(X,Y)*** *= Set of x and y coordinates for lowermost pixels for each column in the image matrix.*
2. p = polyfit($\mathbf{X_A}$*,* $\mathbf{Y_A}$*, 1);*
//p is the equation of the best fit line passing through A[x,y]
*3. final_image=****cp2transform****(input_image, 'projective', slope(p))*
//final image: a result of projective transform
end

Figure 3: The Rotation algorithm

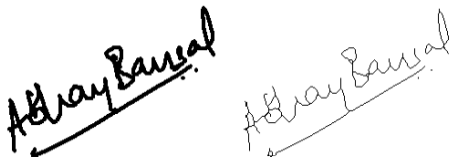


Fig. 4: Signature image before and after thinning

3. Critical Region based graph matching approach:

The proposed approach consists of extracting critical points on the input signature, locating the corresponding critical points among the sample signatures, extracting the critical regions centered around the critical points on the respective signatures, matching the corresponding critical regions using graph matching algorithm, training the sample signatures and finally, verifying the authenticity of the test signature. The entire approach is illustrated in the block diagram of figure 5.

3.1 The *Polyfit* Function

The digitally scanned 2-D image of a signature gives a pixilated image. To overcome this limitation, Matlab's Image Processing Toolbox is used to estimate the continuous curve that best fits the image. The equation of any image can be estimated using the *polyfit* function:

[p,S] = polyfit(x,y,n) *(1a)*
[p,S, mu] = polyfit(x,y,n) *(1b)*

Where,
p: coefficients of a polynomial p(x) of degree n that fits the image data p(x(i)) to y(i) in a least squares sense.

$$p(x) = p_1 x^n + p_2 x^{n-1} + \ldots + p_n x + p_{n+1}$$

S: A structure that is used for getting error estimates i.e. deviation of predicted curve from the actual pixilated curve. Bounding the error value *S.normr* in a certain range i.e. (*0,Thresh*) gives the exact positions of the critical points, as depicted in the figure 6. The value of the threshold is experimentally tuned.

The function *[p,S, mu] = Polyfit(x,y,n)* finds the coefficients of a polynomial in $x = ((x-\mu_1)/\mu_2)$ *where* $\mu_1 = mean(X)$ and $\mu_1 = std(X)$. This centering and scaling transformation improves the numerical properties of both the polynomial and the fitting algorithm. The parameter *S.normr* keeps a track of the error.

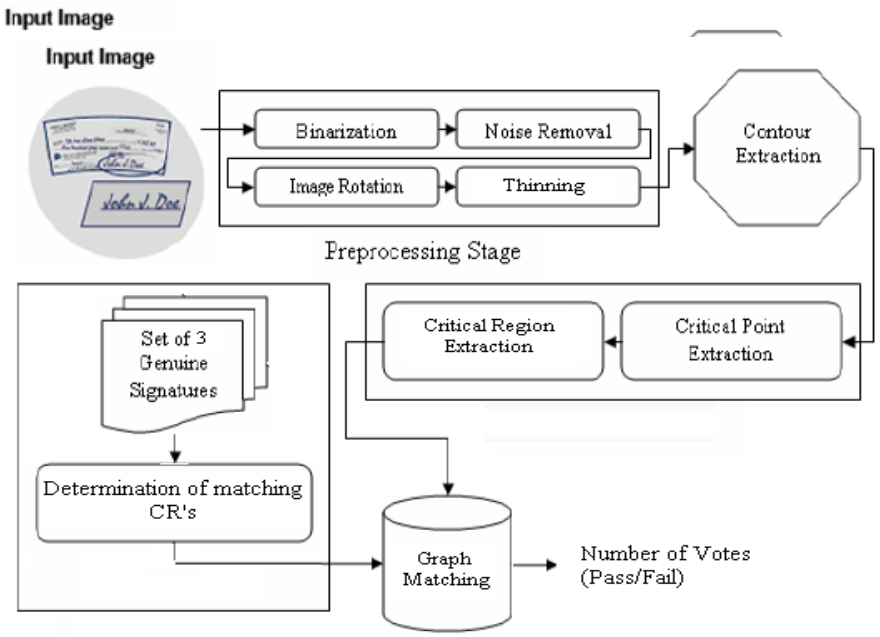


Figure 5: Block diagram of the proposed approach.

3.2 Extraction of critical points:

A contour based approach is followed to extract the critical points. In this approach the contour is traversed and any sharp change in the curve is marked as a critical point. Critical points can be best described as the set of points which model the basic structure of the signature. They are a minimum set of points to represent the shape of a signature.

A contour can be described as the outer boundary of a signature. To extract the same, the disconnected components in the signature are joined and the 'holes' inside the signature are filled. The set of all four-connected boundary pixels define a contour of the signature. The process undergoes the following steps. Figure 12 gives a pictorial illustration of the sequence of steps in the process.

As depicted in figure 12a, the contour image obtained is first thickened using a 5X5 morphological filter followed by thinning [5]. This is done to eliminate any sharp changes and to bring about uniformity in the curve. A unique point on the contour image (must occur in the same region for the same user set) is then selected as a starting point and the contour is traversed in the clockwise direction. Critical points are encountered during this traversal by an algorithm ***Critical_Points_Extract,*** described in the pseudocode in figure 8. A brief explanation follows.

The algorithm repeatedly segments the signature image into small curves using the *polyfit* function, taking care that at least 5 points are used. As the curve is extended to include newer points, the deviation of the curve as given by the error value *abs(S.normr)* indicates whether a peak is encountered. A critical point is identified when either the current peak exceeds an experimentally tuned threshold, or when the

number of peak points obtained past the last critical point exceeds a pre-determined number, as evaluated by the *track_error_peak* function. Figure 12b illustrates the critical points extracted for sample signatures.

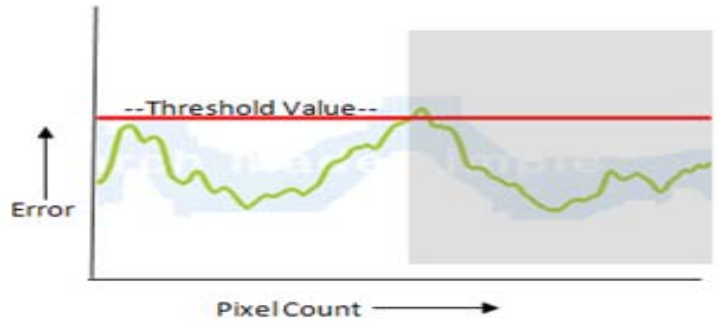


Figure 6. Bounding error estimate with an error threshold

Fig. 7a. Original signature

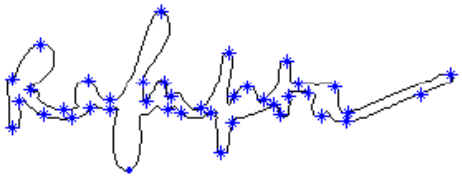


Fig. 7b. Extracted Critical points

3.3 Finding the corresponding critical points among the sample signatures:

After extracting the critical points from the sample signatures, the next step requires finding out the correspondence among the critical points in these signatures. The aim is to find out which critical point in a signature A corresponds to which critical point in the signature B.

The algorithm for matching the critical points ***Match_Criical_Points*** is given in figure 9. The procedure is explained below.

First, each critical point on signature *SigA* and signature *SigB* is masked with a 21 × 21 *black block* centered on the critical point. Every pixel within a block is set to the value '1' (black). The remaining pixels in both images are marked '0' (white). Thus we obtain two new *images SigA'* and *SigB'* respectively containing only the black boxes at their respective positions. Let N_{SigA} and N_{SigB} be the total number of critical points on SigA and *SigB* respectively.

The algorithm next finds the common portion of the every block of *SigA'* with respect to each of the blocks extracted from the *SigB'* by using a simple AND gate function between corresponding locations, operating on the binary values 0 (white) and 1(black).

Finally, for each block in *SigA'* the maximally overlapping block in *SigB'* is located. In effect, the algorithm traverses an *Overlap_matrix* with N_{SigA} rows and N_{SigB} columns, where cell(i,j) is set equal to the number of overlapping pixels between the i^{th} block of *SigA'* and j^{th} block of *SigB'* (refer figure 10). The highest value cell in row k indicates the matching critical point of *SigB* for the k^{th} critical point on *SigA*. Figure 12c depicts the result of the process of matching critical points on a pair of sample signatures.

```
Algorithm: Critical_Points_Extract.
Input: Contour Image I(X,Y).
Output: C(X,Y) Set of extracted  critical points.
Begin
        Initialize: I_count=0, C_count=0 , Error_vector = ϕ;
        for i in 1 to size-of-contour
                increment I_count;
                .if I_count >  5
                        [P S mu]  ←polyfit(X_I( i: i+I_count),   Y_I( i: i+I_count), 2);
                        Error_vector = Error_vector ∪ S.normr.
                else     continue;
                endif
                if abs(S.normr) <  10
                        Check ←track_error_peak(Error_vector);
                         if Check = 1 then peak is encountered. Set:
                                 x_C(C_count)  ←  x_I(I_count);  // x  coordinate  :critical
        point
                                 y_C(C_count) ←y_I(I_count); //y coordinate :critical point
                                 Increment C_count ;
                                Clear I_ count;
                                Error_vector = ϕ;
                                Continue;
                         endif
                 else
                        x_C(C_count) ←x_I(I_count); // x coordinate :critical point
                        y_C(C_count) ←y_I(I_count); //y coordinate :critical point
                        Increment C_count ;
                        Clear I_ count;
                        Error_vector = ϕ;
                endif
         endfor
end
```

Figure 8. Algorithm for Critical Point Extraction

Algorithm: Match_Critical_Points
Input : $C_A(X,Y)$, $C_B(X,Y)$ *are sets of critical points for signature images SigA and SigB*
Output : $M(X_1,Y_1,X_2,Y_2)$ *is a set of Matched critical points for sets* C_A *and* C_B

Begin
Initialize **Block_CR_signA = zeros** *(800,800),* **Block_CR_signB** = **zeros** *(800,800)*
1. ***for*** *i* ***in*** *1 to number-of-critical-points in* C_A
Block_CR_signA$(x_{CA}(i)$ *to* $x_{CA}(i)+20$, $y_{CA}(i)$ *to* $y_{CA}(i)+20)$ ← ***ones****(21,21);*
for *j* ***in*** *1 to number-of-critical-points in* C_B
Block_CR_signA$(x_{CB}(j)$ *to* $x_{CB}(j)+20$, $y_{CB}(j)$ *to* $y_{CB}(j)+20)$ ← ***ones****(21,21);*
Block_CR_overlap ← ***Block_CR_signA*** *AND* ***Block_CR_signB****;*
Overlap_matrix*(i, j) = number-of-1's in* ***Block_CR_overlap****;*
Clear ***Block_CR_overlap, Block_CR_signB;***
endfor
Clear ***Block_CR_signA;***
endfor
Initialize Match_count=0;

2. ***for*** *i* ***in*** *1 to number-of-critical-points in* C_A
max_i ← *max(****Overlap_matrix****(i,1 to number-of-critical-point in* C_B*))*
*//the function 'max(****V****)', returns the maximum value in vector* ***V***
if *max_i* ≥ *200* ***then*** *//a match is found*
j ←*find(max_i,****Overlap_matrix****(i,1 to number-of-critical-point in* C_B*))*
*// the function ' find(a,****V****)', returns the index of '****V****' where the value =a*
$M(x_{1M}(count),y_{1M}(count),x_{2M}(count),y_{2M}(count))=[x_{CA}(i),y_{CA}(i), x_{CB}(j),y_{CB}(j)]$;
Overlap_matrix*(i,1 to number-of-critical-point in* C_B*)=0;*
Increment Match_count;
endif
endf or
End

Figure 9: Matching critical points in a pair of signatures images *SigA and SigB*

	1	2	3	…	N_{SigA}
1	118	210	56		0
2	12	32	116		0
3	5	21	23		0
…					
N_{SigB}	0	0	43		281

Figure 10: The *Overlap Matrix*

3.4 Extraction of critical regions and comparing them using graph matching:

After determining the one-to-one corresponding matched critical points in the sample signatures, their respective critical regions are extracted. Critical regions serve as a sound basis for modular graph matching. Instead of using the entire signature

image, its critical portions are extracted and corresponding regions are compared to judge the overall similarity between the input and sample signatures. Figure 12d illustrates graph matching. The algorithm ***Ext&Mat_Critical_Regions*** for extracting and matching corresponding critical regions in a pair of sample signatures, is described in the pseudo code given in figure 11. Its working is outlined below.

A critical region is a 31× 31 block extracted from a signature image and containing a critical point at the center. For every critical point on signature A, the algorithm extracts a 31×31 block ***CR1***, taking the critical point *cp1* as centre. The corresponding 31× 31 block ***CR2*** from signature B is also extracted, taking the matching critical point *cp2* as centre.

Each critical region is represented by a undirected graph in which every black pixel in the critical region signifies a vertex. The *x, y* coordinates of all black pixels in ***CR1*** and ***CR2*** represent vertex sets *S1* and *S2* respectively.

Matching two graphs measures the similarity of the two corresponding critical regions based on their geometrical shapes. The distance-matrix ***W*** is a (m × n) adjacency matrix whose rows represent vertices of ***S1*** and whose columns represent vertices of ***S2*** (where |***S1***|>=|***S2***|). We calculate the Euclidean distance each pair of vertices in ***S1*** and ***S2*** using the x-y co-ordinates of their corresponding pixels.

The formulated assignment problem is solved using the Hungarian method [6]. This returns the optimal cost/distance *min_dist* between critical regions ***CR1*** and ***CR2***. The *min_dist* is divided by |***S1***| to get a normalized minimum distance per pixel. It is further divided by a factor α which is a measure of the percentage of vertices matching in ***S1*** and ***S2***.

The above procedure is repeated for every pair of matching critical regions to yield the array of optimal distances ***Optimal_Distance*.**

3.5 Signature verification:

Once we get the optimal distance vector we compare it against a threshold (opt_thresh~=15). For each value less than the threshold the vote number is incremented. For a set of N values any signature giving more than two third votes is considered a genuine signature. In case a signature gives consideration results with the first genuine sample but is not good enough to be tagged as genuine, in that case the signature is tested against the second genuine signature sample.

If it still does not pass the test, then is tested against the third sample. If the signature gives average results (2N/3>votes>N/3) with the entire genuine signature-set, then it is tagged as a probable forgery. If at any stage the input signature gives unacceptable results (votes<N/3) with any of the genuine signature samples, then it is straightaway rejected.

Algorithm: ***Ext&Mat_Critical_Regions***

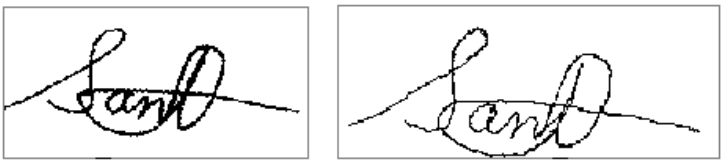
Input: Set of all matched critical points $\mathbf{M(X_1,Y_1,X_2,Y_2)}$*, contour image* $\mathbf{I_A(X,Y)}$ *and* $\mathbf{I_B(X,Y)}$
Output: Optimal distance matrix ***Optimal_distance*** *of size |M|.*

Begin
Initialize CR1=zeros*(31,31)****,CR2=zeros****(31,31)*
for *i* ***in*** *1 to no-of-matching-points in* ***M***
$\mathbf{CR1} \leftarrow \mathbf{I_A}(x_{1M}(i)-15 \text{ to } \mathbf{x}_{1M}(i)+15, \mathbf{y}_{1M}(i)-15 \text{ to } \mathbf{y}_{1M}(i)+15)$
SI(X1, Y1) ← *find(****CR1****)*

*//the function 'find(**V**)' returns the set of x,y coordinates for each member of //**V** whose value is equal to ' 1'.*
CR2 ← $I_B(x_{2M}(i)-15$ *to* $x_{2M}(i)+15$, $y_{2M}(i)-15$ *to* $y_{2M}(i)+15)$
S2(X2, Y2) ← *find(**CR2**)*

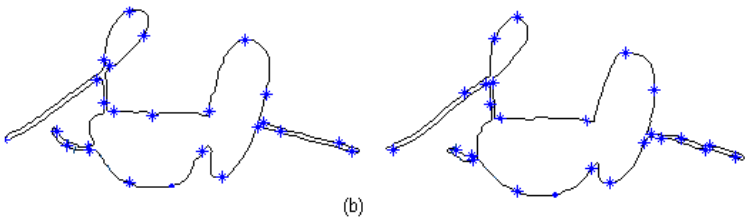
Initialize Distance_matrix=zeros*(no-of-black-pixels in **CR1**, no-of-black-pixels in **CR2**)*
Distance_matrix ← *dist(**S1,S2**)*
*//the function 'dist(**A,B**)' returns a matrix where rows are the //indices(ind1) of **S1** and columns are indices(ind2) of **S2** and the value at //**D**(ind1,ind2) is the Euclidean distance between **S1**(**x1**(ind1),**y1**(ind1)) and //**S2**(**x2**(ind2),**y2**(ind2)).*
Min_Dist ← *Hungarian(**Distance_Matrix**) ;*
Optimal_Distance*(i)* ← ***Min_Dist****/ no_of_black_pixels_CR1 ;*
endfor
End

Figure 11. Extraction and matching of critical regions.



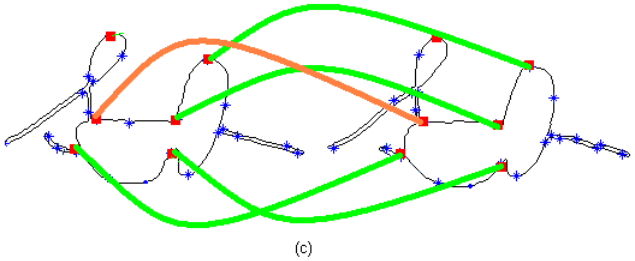
(a)

Figiure 12a. Thickened and Thinned signature images



(b)

Figure 12b. Extraction of critical points



(c)

Figure 12c. Matching critical points on a signature pair

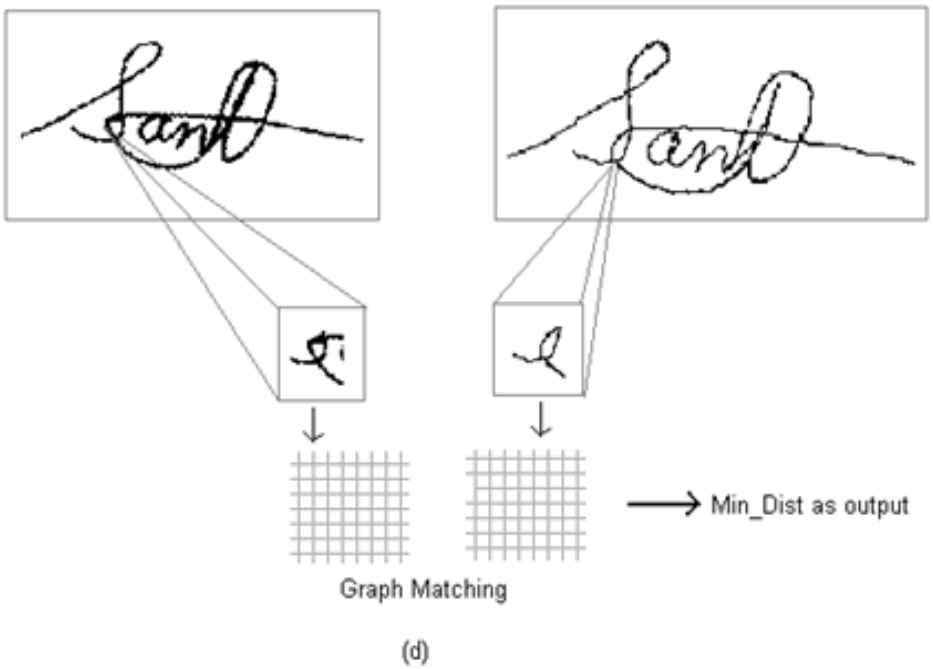


Figure 13. Description of the whole process

4. Results

Datasets: 5 genuine signatures of each of 76 persons were taken. Therefore, total 5 x 76 = 240 genuine signatures were collected.

For each of the 76 persons, 4 semi-skilled and 4 skilled forgeries were collected. After collecting signatures, they were scanned as 8-bit gray images using a scanner with 200 dpi resolution. The program was run on Pentium IV PC, 1.70 GHz with 512 MB RAM.

Tests were conducted on Authentic, Semi-skilled and Skilled forgery signatures for each person.

1. Authentic test: 3 genuine signatures were taken as sample signatures. The remaining 2 genuine signatures are tested against these 3 samples. Total number of tested signatures is 76X2 = 152. This is used for the calculation of False Rejection Ratio (FRR).
2. Semi-skilled Forgery test: For every person, 4 skilled forgeries are tested. This is used to find the FAR skilled. Therefore, total 4X76 = 304 signatures were tested.
3. Skilled Forgery test: For every person 4 skilled forgeries are tested. This is used to find the FAR skilled. Therefore, total 4X76 =304 signatures were tested.

The test results are compiled in Table 1.

5. Discussions and Conclusions

Traditionally, graph matching is used on the whole image and each pixel is compared with every other pixel in the other image, thus incurring a large computational overhead. For images of resolution of $P \times Q$ *and* $A \times B$, *P, Q, A* and *B* being typically in the range 300-1200, we compute a non-negative $n \times m$ matrix, where the element in the *i*-th row and *j*-th column represents the cost of assigning the *i*-th pixel to the *j*-th pixel in considered images *1* and *2* respectively. The complexity of the algorithm amounts to be $O(n^3)$ *where* $(n \sim = m)$.

In our approach, we identify isolated, smaller critical portions of the signature images.

These critical regions contribute significantly to the shape of the original image and therefore serve as accurate model of the signature. These critical regions are utilized as a basis for graph matching, thus reducing the computational overhead by a large amount. Critical regions of size 31×31 are constructed and compared using Hungarian method. And this computation is done for only some identified points, say N_{crit}. Thus the computational time is reduced to (N_{crit} $O(p^3)$) *where (p<<n).* For a typical value of N_{crit}=*20,* this can amount to around 10^6 times less the the time overhead incurred in the previous approach. In fact, this approach is efficient and even comparable with online signature verification methods in terms of results.

We have proposed an algorithm that not only works better than the similar graph-based offline verification approaches but also works on a sample base of just three authentic signatures, which is closer to the real world requirements. In this paper we demonstrate that it is possible to achieve very low error rates even for skilled forgeries. The approach is computationally faster as compared to other graph matching techniques. It introduces the concept of modular graph matching.

Table 1. Test Results for the experimental data-set

Type of Forgery	**No. of Signatures**	**Accuracy(%)**	**Error(%)**
Authentic	152	98.68	1.32
semi-skilled	304	95.69	4.31
Skilled	304	89.09	10.81

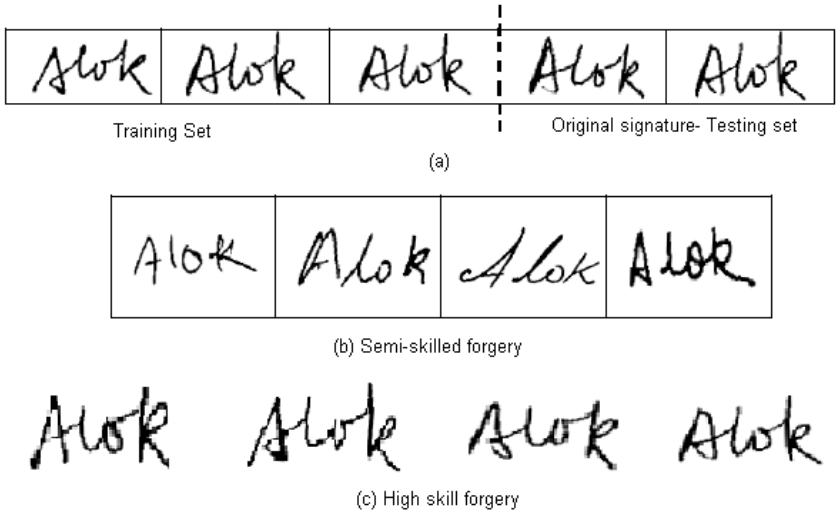


Figure 13. A depicts the training set of three signatures being taken and the rest are the sample signatures used for testing.

References

[1] N. Christofides, *Graph theory: an algorithmic approach* (New York, Academic Press Inc., 1977).
[2] Ibrahim S.I. Abuhaiba, Offline Signature Verification Using Graph Matching, *Turk J Elec Engin, VOL.15, NO.1* 2007.
[3] Siyuan Chen and Sargur Srihari, A New Offline Signature Verification method based on Graph Matching , *18th International Conference on Pattern Recognition Volume 2,* Issue , 2006.
[4]. Lal Chandra, Puja Lal, Raju Gupta, Arun Tayal,Dinesh Ganotra: Improved adaptive binarization technique for document image analysis. VISAPP (1) 2007: 317-321.
[5]. Lalit Kumar, Abhay Bansal, Dinesh Ganotra, Neeraj Jain: An improved approach for Thinning by preserving local coupling points: selected for VISAPP 2008.
[6]. http://en.wikipedia.org/wiki/Hungarian_algorithm
[7]. J. J. Igarza, I.Goirizelaia, K. Espinosa, I. Hernáez, R.Méndez and J.Sánchez, "Online Handwritten SignatureVerification Using Hidden Markov Models", CIARP 2003, LNCS 2905, Springer-Verlag , A.Sanfeliu and J. Ruiz-Shulcloper (Eds.), 2003, pp. 391-399.
[8]. J. Edson, R. Justino, F. Bortolozzi, and R. Sabourin, "An Off-Line Signature Verification Using HMM for Random, Simple and Skilled Forgeries," Proc. Sixth Intl Conf. Document Analysis and Recognition, pp. 1031-1034, Sept. 2001.
[9]. N. Papunzarkos, and H. Baltzakis, "Off-Line Signature Verification Using Multiple Neural Network Classification Structures", IEEE 1977.
[10]. Maan Ammar, Yuuji Yoshida, Teruo Fukumura, "Description of Signature Images and its Applications to their classification", IEEE 1988.
[11]. B. Fang, C.H. Leung, Y.Y. Tang, K.W. Tse, P.C.K. Kwok, Y.K. Wong, "Offline signature verification by the tracking of feature and stroke positions", Pattern Recognition Society (2002).
[12]. E.J.R. Justino, A. El Yacoubi, F. Bortolozzi, R. Sabourin ""An Off-Line Signature Verification System using HMM and Graphometric Features", DAS 2000, Rio de Janeiro, Dec. 2000, pp. 211-222.

Authors

Abhay Bansal is studying B.E in Computer Science from Netaji Subhas Institute of Technology. His research areas are Computer Vision and Pattern Recognition. He has publications in various International conferences in the field of Image Processing.

Bharat Gupta is studying B.E in Computer Science from Netaji Subhas Institute of Technology. His research areas are Soft Computing, parallel algorithms and Mathematics.

Gaurav Khandelwal is studying B.E in Computer Science from Netaji Subhas Institute of Technology. His research areas are Graph Theory, Computer Vision and Pattern Recognition.

Shampa Chakraverty is Professor in the Department of Computer Engineering of Netaji Subhas Institute of Technology. She obtained her B.E. degree in Electronics and Communication from Delhi College of Engineering, M-Tech in Integrated Electronics and Circuits from I.I.T. Delhi and Ph.D. from the Faculty of Technology, Delhi University. Her research interests are in the areas of VLSI, hardware software co-design and soft computing.