

Improved Geometric Feature Graph: A Script Independent Representation of Word Images for Compression, and Retrieval

Gaurav Harit Richa Jain Santanu Chaudhury
Electrical Engineering Department
Indian Institute of Technology, Delhi, New Delhi, India
gharit@ee.iitd.ac.in, richaajain10@rediffmail.com, santanuc@ee.iitd.ac.in

Abstract

In this paper, we discuss a new representation scheme for word images which exploits the structural features. The word image features are represented in the form of a graph called as the Geometric Feature Graph (GFG). The GFG is encoded in the form of a string which serves as a compressed representation of the word image skeleton. We demonstrate reconstruction, and retrieval of word images for 3 different scripts using the GFG string.

1. Introduction

Word spotting [4] is widely used for indexing and retrieval of word (printed or hand-written) images in documents. Many of the recent works [3, 5] have devised features and matching measures for word spotting and analysis. However none of these schemes allow synthesis of the word images from feature based representations. Our contribution lies in our novel strategy, an improvement of our earlier work [2], to represent a word image in the form of a graph so as to provide compression as well as efficient matching for indexing and retrieval tasks. In the subsequent sections, we provide the complete algorithmic framework to generate the compressed representation and matching of word images in a retrieval algorithm. We have compared the compression results offered by our scheme with the JBIG compression [1]. Our retrieval results establish the efficacy of our matching algorithm.

2. GFG based representation of Word Images

The word image skeleton is broken down into smaller segments by identifying a set of critical points on the skeleton such that they would form end points of the possible segments. A graph is the most informative way to represent the set of critical points (as the nodes) and the set of

segments (as the links connecting the nodes) thus identified from the word skeleton. A GFG is a representation of a word image structure (in the form of geometric features) expressed as a graph where the nodes represent certain key points identified on the word image and the links represent shape primitives that best characterize the structural segments connecting pairs of neighboring nodes. Representing the word image using a GFG makes it possible to achieve compression because the number of shape primitives required to approximate different possible segments in a word image is very less compared to the total number of pixels in the word image.

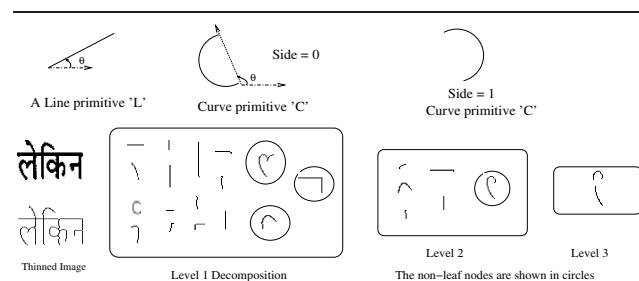


Figure 1. Shape primitives, Word Decomposition

2.1. Shape Primitives Used

In this work we use two types of shape primitives: A line segment, and a curve segment. A line segment drawn from a given end point is characterized by its length and its orientation. We model a curve segment as a semicircle. Thus a curve drawn from a given point is characterized by the orientation and the length of its diameter. Since there are two possible semicircles around a diameter, we need to include the information as to whether the semicircle is the one to the left side of the diameter (when the diameter is being viewed

as starting from the given point) or to the right side. The shape primitives are shown in Fig 1. In this paper, a line segment is referred to as the shape primitive ‘L’ and described as ‘L-len-orient’, where the attributes length and orientation are also included as parameters, and thus uniquely identify a line segment to be drawn from a given point. A curve segment is referred to as the shape primitive ‘C’ and described as ‘C-side-len-orient’, where the attributes are the side of the diameter which the curve takes (0 for left and 1 for right), and the length and the orientation of the curve diameter respectively.

2.2. Marking the Critical Points and the Segments

The word image is binarized and then preprocessed with a closing operation. The skeleton is obtained by a thinning algorithm. An initial set of critical points are identified as the junction points and the dead end points of the word skeleton. A junction point is the point which has three or more branches of skeleton emanating from it. Thus, if a pixel has 3 non-contiguous neighboring (8×8 neighborhood) skeleton pixels, it is a junction. A skeleton end point has exactly one contiguous set of neighboring skeleton pixels. The skeleton (denoted as \mathcal{W}) is divided into segments, with the critical points serving as end points of the segments. Word segments that connect such critical points are contiguous sets of pixels without any branching (see Fig 1, Level 1 decomposition). However the curve segments may still have a complex shape even though they do not exhibit branching. We denote the initial set of segments as $\{S\}$ where S is one of the segments.

$$\mathcal{W} \longrightarrow \{S\} \quad (1)$$

If the skeleton \mathcal{W} is considered to be a root element of a tree, the segments $\{S\}$ are its children. The idea is to recursively subdivide each of these segments to form smaller segments, the subdivision being stopped if a segment can be approximated (within tolerable errors) by one of the shape primitives, in which case it would be a leaf segment. A leaf segment is denoted as F_L (or F_C) if its being represented by a line (or curve) primitive.

In the first stage, we mark out the straight line segments (if present) in each of the segments in $\{S\}$. Thus identified straight line segments in S form the leaves $\{F_L\}$ and need not be subdivided. After removing the straight line segments, the remaining pixels in S are organized into segments S_C (subscript c indicating that the segment does not have a straight line) by identifying the connected components. This is shown by transitions:

$$S \longrightarrow \{S_C\} \quad (2)$$

$$S \longrightarrow \{F_L\} \quad (3)$$

In the second stage, each of the segments in the set $\{S_C\}$ is subdivided (if necessary) into further curve segments at the points where the curvature is changing. However if the segment S_C can be represented by the shape primitive ‘C’ then it forms a leaf F_C and is not considered for further subdivision.

$$S_C \longrightarrow \{S_C\} \quad (4)$$

$$S_C \longrightarrow F_C \quad (5)$$

In the third stage, we are left with segments $\{S_C\}$ arising after transition (4). There can be the following possibilities.

$$S_C \longrightarrow F_C \quad (6)$$

$$S_C \longrightarrow F_L \quad (7)$$

$$S_C \longrightarrow \{S_C\} \quad (8)$$

Here transitions 6 and 7 indicate that the curve segment S_C can be represented by a shape primitive ‘C’ and ‘L’ respectively. Transition 8 is recursive and implies the need to subdivide S_C since it cannot be reasonably represented by any shape primitive.

2.2.1. Detection of Line Segments Detection of straight lines is done in the first stage (for transitions 2 and 3). Line detection is organized into 3 steps: detection of horizontal lines, then vertical lines, and finally the slant lines. Detection of horizontal lines is done by horizontal profiling, ie, counting the segment pixels in each row. The row for which the count is large indicates the presence of a horizontal line(s) in that row. The horizontal line(s) is thus identified as the connected set of segment pixels in that row. Different horizontal lines may exist in different rows and they are all marked out as leaves F_L . The remaining pixels which are not part of any line are organized into child segments S_C . The vertical lines are detected by doing a vertical profiling in a similar way.

Detection of slant lines is computationally costly and hence we carry out this step after detecting the horizontal and vertical lines so that the leaf segments marked by the first two steps need not be processed by this step. Consider a parametric representation of the segment $C = \{x(t), y(t)\}$ where t denotes the pixel number (index) as we traverse the segment from one end. Let N be the total number of pixels in the segment. The algorithm for straight line detection is outlined as follows:

Define the following 4 flags for each pixel:

- 1 LA and LB as the line Ahead and line Behind flags
- 2 SnA and SnB as the sign Ahead and sign Behind flags
- 3 For each pixel ($t = 0 \rightarrow N$) DO lines 4 to 18
- 4 Initialize each of flags LA, LB, SnA, SnB to false
- 5 L^+ be the line joining $x(t), y(t)$ with $x(t+\sigma), y(t+\sigma)$

```

6   $L^-$  be the line joining  $x(t), y(t)$  with  $x(t-\sigma), y(t-\sigma)$ 
7   $m(L^+)$  be the slope of  $L^+$ 
8   $m(L^-)$  be the slope of  $L^-$ 
9   $d_{avg}^+(t) = \frac{1}{\sigma} \sum_{t'=t+1}^{t'+\sigma} d_{\perp}^+(t')$  where  $d_{\perp}^+$  is perp dist from  $L^+$ 
10  $d_{avg}^-(t) = \frac{1}{\sigma} \sum_{t'=t-1}^{t'-\sigma} d_{\perp}^-(t')$  where  $d_{\perp}^-$  is perp dist from  $L^-$ 

11  if ( $d_{avg}^-(t) < \gamma$ )    LB = true
12  if ( $d_{avg}^+(t) < \gamma$ )    LA = true
13  if (LB is true AND LA is true) {
14  if ( $fabs(m(L^+) - m(L^-)) > \Delta_{slope}$ ) LB = false
15  else    LA = false
16  }
17  if ( all pixels from  $t$  to  $t + \sigma$  lie towards
        clockwise direction of  $L^+$ )    SnA = true
18  if ( all pixels from  $t$  to  $t - \sigma$  lie towards
        clockwise direction of  $L^-$ )    SnB = true

```

```

19  Initialize  Label = 1
20  For each pixel ( $t = 0 \rightarrow N-1$ )  DO lines 21 to 25
21  LB and LA are the flags for this pixel  $t$ 
22  Let LB' and LA' be the flags for pixel  $t + 1$ 
23  if (LB is true AND LB' is false)  Label ++
24  else if (LB is false AND LA' is true)  Label ++
25  Assign Label to this pixel

```

In steps 5 and 6, σ is a smoothing parameter. The threshold σ is chosen to be half the minimum length (in terms of number of pixels) of straight line desired to be identified. In step 9, $d_{\perp}^+(t)$ denotes the perpendicular distance of point t' from line L^+ . The threshold γ in step 11 is used to decide whether the σ pixels behind t lie on a straight line. We have chosen γ as 1.5. For pixels which lie on a straight line, both the flags LA and LB evaluate to be true. Condition in step 14 is satisfied by a pixel which lies on the intersection of two straight lines. Δ_{slope} is a threshold (chosen 20 degrees) to constrain that, for an intersection point, the line ahead and the line behind must have a slope difference. For step 17, we assume L^+ to be a directed line from t to $t+\sigma$. The flag SnA is true if majority of pixels on the curve between t and $t+\sigma$ lie on the clockwise direction from the line L^+ , and is false if majority of pixels lie on the anticlockwise direction. In steps 19 to 25, each pixel is assigned a label. Consecutive pixels which lie on a straight line segment get the same label and now form a leaf F_L . Pixels that lie on a curved segment also get a common label. The curved segments are assigned as S_C and are easily distinguished since they have pixels with both LA and LB flags as false.

2.2.2. Splitting and Identification of Curve Segments

In order to represent a curved segment with shape primitive 'C' it is essential that the curve should have either a positive or a negative curvature. Complex curves have pos-

itive curvature subsegments as well as negative curvature subsegments, and hence need to be splitted at the points of zero crossings of the curvature. In our scheme for detecting slant straight line, as we compute the perpendicular distance for the pixels t' , we also keep a note of the side of the line the pixels lie. This is done using the flags SnA and SnB. The segment is split at the points where SnA and SnB are both true, or both false. Such points symbolize a change in the curvature sign. Thus our algorithm has one more step listed after line 24.

```

25  if ((SnA is true AND SnB is true) OR (snA is false AND
SnB is false))  Label ++

```

Referring to section 2.2 this kind of curve splitting is carried out for transition (4). Transition (5) corresponds to the curve segments which have a constant sign curvature, but the curve is better approximated by a straight line joining its end points, rather than a semicircle joining its end points. Transition (8) is recursive, but here the segment subdivision takes place at the point which is at the maximum perpendicular distance from the base line (ie, the line joining the end points of the segment). The segments which qualify for transition (8) usually have the perpendicular from the farthest curve point (from the baseline) intersecting the base line at a point which happens to be sufficiently close to one of the ends (of the baseline), rather than being somewhat middle of the baseline. Such curves need to be split (transition 8) since they cannot be reasonably approximated by the shape primitive 'C'. Though transition (8) is recursive, the recursion is stopped if the length of a segment falls below a threshold, and thereafter either of the transition (6) or (7) is evoked. The algorithm for line detection has the advantage of providing smoothing proportional to the skeletonizing error. It provides for detection as well as identification of line and curve primitives.

2.3. Generating a Representation of GFG

The Geometric Feature Graph (GFG) is formed using the critical points as the nodes and the segments as the branches connecting the nodes. Each branch carries a label which indicates the shape primitive used to approximate it and the values of parameters for the primitive. We need a way to encode the GFG in the form of a string such that the reconstruction of the original GFG and thus the word skeleton, is possible given just the encoded GFG string. To do this, we select a node (typically the left most node in the skeleton) and do a depth first traversal (DFT) of the GFG. The labels of the branches are concatenated in the sequence in which the branches are visited. The concatenated string is called as the GFG string for the given word skeleton. During DFT, as we reach a dead end of a branch, we need to back-track to a previous node so that any unvisited branches emanating from it may now be visited. If DFT is implemented us-

ing a stack, backtracking calls for popping nodes out of the stack. When a node is popped from the stack, we insert a special symbol '\$' to the GFG string.

2.3.1. Compression using GFG string We represent each branch label with 1 byte. The first bit indicates the type of shape primitive (whether 'L' or 'C'). The second bit is used to indicate the *side* parameter value for the case of 'C' primitive. The next 3 bits store the quantized *length*, and the last 3 bits store the quantized orientation. Thus, the length gets quantized to 8 levels and the orientation is approximated using one of the 8 prominent directions. A word image often comprises of disconnected alphabets giving rise to more than one connected components. Since the complete word image needs to be represented as a single GFG, a dummy branch (straight line) is introduced between the nearest nodes of two connected components of the word skeleton. The dummy branch serves as a link so that tree traversal can take place and the position of the connected component can be localized. While reconstructing the word image, the dummy branch is not shown. For the 'L' primitives, the second bit is set to 0 to indicate a dummy segments. The number of bytes used to store the complete GFG string is equal to the number of branches in the GFG and the number of times the backtracking symbol '\$' (represented in 1 byte with all zero bits) is inserted in the string. We have achieved high compression rates with the GFG string representation of word images. For the word images (Fig 2(a to f)) of three different languages (Hindi, Bengali and Telugu), the size of the GFG strings were 54, 51, 36, 43, 27, 56 bytes respectively. The size of the corresponding thinned images in JBIG format were: 158, 163, 199, 196, 192, 137 bytes respectively. This shows that our GFG string representation offers compression of around 60 to 80% over the JBIG compressed thinned word images. For more accurate representation of branches, we can use 2 bytes for the label, which offers a more fine quantization of length and orientation.

2.3.2. Reconstruction of Word Images using GFG The GFG is decoded from the GFG string by traversing the string in Depth First order. A stack is used to store the end points (nodes) of the branches. A new branch is drawn from the node which is at the top of the stack. The other end point of drawn branch is pushed into the stack. The backtracking symbol '\$' is used to pop out nodes from the stack. Once all the branches have been drawn, we have the reconstruction of the word image. The reconstruction can be done at various sizes by multiplying the values of the primitive length with a chosen scale factor. The shape primitives are drawn using a rasterization algorithm with antialiasing. We have achieved a fairly reasonable reconstruction of word images

with one byte representation for branch labels. The results of word image reconstruction from GFG string are shown in Fig 2. The results show satisfactory reconstruction for word images from 3 languages (Hindi, Bengali and Telugu). Despite minor artifacts because of mis-recognition of certain noisy branches, almost all the reconstructed word images were quite legible. Though the GFG string depends on the choice of the start node, the reconstruction of the word skeleton from the GFG string is not dependent on the choice of the start node.

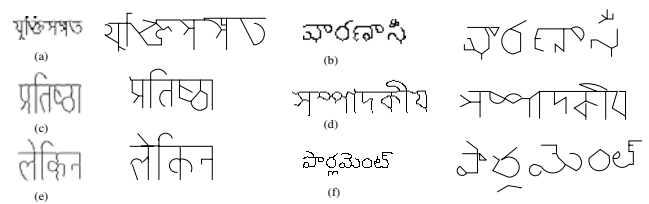


Figure 2. Reconstruction using GFG string

3. Retrieving Word Images using GFG

The GFG string can serve as the index of the word images in a database of documents. A query word image is represented as a GFG and is matched with the GFG of each word image in the database. The word images whose GFGs best match with the given GFG are retrieved from the database. In this section we describe how the matching between two GFG strings is carried out. This is not a problem of straightforward string matching because the GFG string of a word image is not unique.

The GFG string is traversed in a depth first order to identify all the junction points. The branches emanating from a junction may comprise of a single shape primitive or a concatenation of multiple shape primitives. In the former case it is a simple branch and in the latter case it is called as a complex branch. Since the GFG is traversed in depth first order and we have the lengths and orientations encoded for the shape primitives, it is a simple matter to compute the coordinates of the junction points. Next we form a grid by laying a horizontal and a vertical line through each of the junctions. Note that the grid cells are not necessarily of the same size. A rectangle is laid on the grid such that it just encompasses all the junction points. The top left corner of this rectangle is assigned as the origin. The grid lines are numbered relative to the origin. Each junction point is thus assigned grid coordinates depending on the numbering of the grid lines that intersect on it.

The dissimilarity between two junctions is essentially a mismatch measure of the junction attributes: the branches emanating and the grid coordinates. As-

sume that the match cost of two completely mismatched junctions is assigned a value η . If two junctions have their grid positions differing by $(\Delta x, \Delta y)$ then this contributes to a match cost $\mathcal{G}_p = \max(\Delta x, \Delta y) * 0.2\eta$. A branch is best characterized by specifying the direction to which it progresses and the direction in which it shows convexity (ie where its center of curvature lies). For this purpose, we quantize the directions into 8 prominent directions (N,S,E,W,NE,NW,SE,SW). A complex branch may comprise of segments progressing in several directions. Hence the progress direction of a branch is stored in the form of an 8 binned histogram denoted as H_p . Each segment makes a contribution (proportional to its length) to the histogram bin corresponding to its quantized progress direction. For curve primitives, the progress direction is considered to be the orientation of its diameter. A branch having multiple segments may have multiple convexity directions. Again, we store the information related to the convexity direction(s) of a branch in an 8 binned histogram denoted as H_c . The convexity direction for shape primitive 'C' is defined as the direction of the line joining the mid point of the curve to the center of curvature. For a single branch comprising of a line primitive, the convexity direction is not defined. However in a complex branch with multiple straight line segments, the successive line primitives with different orientations give rise to convexity whose direction is computed using their orientations. For two given junctions, we compare their branches using the features H_p and H_c . Any mismatch in the values of histogram bins is penalized with a cost $0.2*\eta$. The cost assigned to a missing branch is $0.5*\eta$. The total match cost of a junction with junction node is the sum of the match costs for the branches and for its grid coordinates. We define the neighborhood of a junction as the set of grid positions for which $\mathcal{G}_p < \eta$. We outline the algorithm to compute match cost between two GFGs.

Let \mathcal{W}_c be the match cost for two GFG strings. $\mathcal{W}_c \leftarrow 0$ GFG1 is the string with the smaller number of junctions. GFG2 is the other one. Let J, J' be the set of junctions of the two GFGs respectively. Let η be the match cost between two completely mismatched nodes. Let C_{ki} be the match cost between $J_k \in J$ and $J'_i \in J'$. Flag all junctions in J' as *unmatched*. Let d be the difference in the number of junctions of the two GFGs.

1. $\mathcal{W}_c = \eta * d$
2. For each junction J_k , such that $J_k \in G$, DO lines 3 to 5
3. Compute $C_k = \min(C_{ki}) \quad \forall J'_i \in \text{Neighborhood}(J_k)$
the min is computed for J'_i which are flagged as *unmatched*. Let the min occur for J'_v
4. $\mathcal{W}_c = \mathcal{W}_c + C_k$
5. Mark J'_v as matched.

Note that for computing \mathcal{W}_c we do not do the reconstruc-

tion of the word image. Rather we only compute the coordinates of the junction points. We consider only the junction points for matching since it is observed that the number of junction points and their features remain invariant even if the skeleton is computed from word images with different font sizes and styles. Also, we have observed that any kind of noise or font variation is less likely to affect the correct identification of junction points, though it does affect the approximation of branches. GFG of a word image is not unique. It depends on the node from where the depth first traversal is started.

4. Results

The retrieval precision of our algorithm was 86% with a recall rate of 90% averaged over a whole database of 10693 word images in 310 documents in three different scripts: Hindi(3915 words), Bengali(3890), and Telugu(2888). The legibility of the words reconstructed from the GFG-string representation was found to be acceptable for around 91%, 85%, and 74% of the word images in Hindi, Bengali, and Telugu respectively. The unusable word GFGs are not considered for the retrieval task. We found that the reconstruction was unusable mostly when the word skeleton itself had noise because of degraded quality and poor resolution of the input word image. To make the retrieval of relevant word images fast, we abort further matching between two word GFGs if there is a mismatch in the number of junctions. Thus most of the word images in the database get pruned out at the first step. Also complete node matching is aborted as soon as a branch mismatch cost exceeds η .

5. Conclusions

We have introduced a Geometric Feature graph based representation of the structural features of the word image. The scheme offers a compressed representation of the word images, while simultaneously offering reconstruction and indexing.

References

- [1] JBIG. <http://www.cl.cam.ac.uk/mgk25/jbigkit/>.
- [2] S. Chaudhury, G. Sethi, A. Vyas, and G. Harit. Devising Interactive Access Techniques for Indian Language Document Images. In *ICDAR*, pages 885 – 889, 2003.
- [3] A. K. Jain and A. M. Namboodiri. Indexing and Retrieval of On-line Handwritten Documents . In *ICDAR*, 2003.
- [4] C. H. R. Manmath and E. Riseman. Word Spotting: A new approach to Indexing Hand Writing . In *IEEE CVPR*, pages 631 – 637, 1996.
- [5] T. M. Rath and R. Manmatha. Word Image Matching Using Dynamic Time Warping . In *CVPR, Madison, WI*, volume 2, pages 521 – 527, June 18-20, 2003.