ENGLISH ALPHABET RECOGNITION USING CHAIN CODE AND LCS

Debi Prasad Bhattacharya

Assistant Professor, Department of Computer Science, West Bengal State University,
B.R.S.College, Barrackpore
North 24 Parganas, Kolkata-700120, India
debiprasad.9831@gmail.com

Susmita Koner

Student, M.Sc. in Computer Science, Department of Computer Science, West Bengal State University,
B.R.S.College, Barrackpore
North 24 Parganas, Kolkata-700120, India
susmita.koner@gmail.com

Abstract

Since several decades OCR system is getting more and more useful in daily life for various purposes. Many researches have been done on many types of characters by using different approaches. However very little investigation have been performed by chain code base approach. In this paper, a new kind of algorithm, based on that chain code along with another significant characteristic, number of vertices (end or junction point) and edges (chain code, from one vertex to another) existing in each alphabet has been used to identify English Capital Characters. We have built a technique that is independent of size and many styles of input characters. Experimental results show relatively high accuracy.

Keywords: OCR; Histogram; Skeleton; Chain code; Longest Common Subsequence.

1. Introduction

The English language consists of 26 characters of which 5 are vowels and 21 are consonant and this language is written from left to right. The set of English characters or alphabets are shown here. Though English letters can be of 2 cases viz. Uppercase and Lowercase, here we consider text only in upper case. Example of the uppercase character of one style of English language is shown in the figure 1.

Fig 1. Character set of English language.

OCR (Optical Character Recognition) plays the main role to translate images of typewritten text or pictures of characters into machine editable text. OCR began as a field of research in artificial intelligence and machine vision. The task of OCR consists of pattern recognition, image processing and natural language processing.

The application of OCR can be viewed in commercial, educational as well as government and security sectors.

In this paper we introduce a new characteristics i.e. number of vertices (end or junction point which exists in each letters), to categorize them into groups. Then chain code has been employed for the

edges. Then some other added features have been employed to make this proposed technique independent of size and style of the letters.
In the rest of the paper the process and all will be structured as follows::The overall approach will be discussed in section 2. The Design and implementation algorithm will be described in section 3. After that in section 4 we will discuss about the testing of the algorithm with the results. Finally the paper will be concluded in section 5.

2. Overall Approach

Printed document of English characters ⟶ scanning of and transformation of the document segmentation into lines ⟶ segments lines into letters ⟶ skeletonize letters ⟶ generate chain code sequence by traversing ⟶ modification of the chain code and vertices(squeezing) ⟶ count no of vertices n ⟶ select group according to no. of vertices n ⟶ If no of vertices n> 6 then match the chain code of all 26 letters one by one else match the chain code of the letters of the group containing the letters with n vertices one by one ⟶ count the no of match (hit) in the chain codes of each edge ⟶ recognize the letter according to the threshold(h) of hits.

3. Design and Implementation

In this paper we are describing an algorithm to recognize English capital characters. To implement it we have used Matlab as tool. The total work done can be divided into 3 phases-Preprocessing, Feature extraction, and finally recognition.

3.1 *Preprocessing*

This stage covers those functions that are needed to be performed on the characters so that feature extraction is possible on it. This stage consists of the following substages :-

3.1.1 *Scanning and transformation*

A printed document of English language is scanned by scanner. This document can be in any file format. First of all we convert it to monochrome binary image. Now the image is ready to be processed further.

3.1.2 *Segmentation*

This step deals with the partitioning of lines from the total document by thresholding histogram values gathered horizontally i.e. row wise. Then letters are segmented from each line by using values of vertical histogram i.e. column wise [4]. Now we will process these individual letters. The example of this process result is shown in the following figure 2

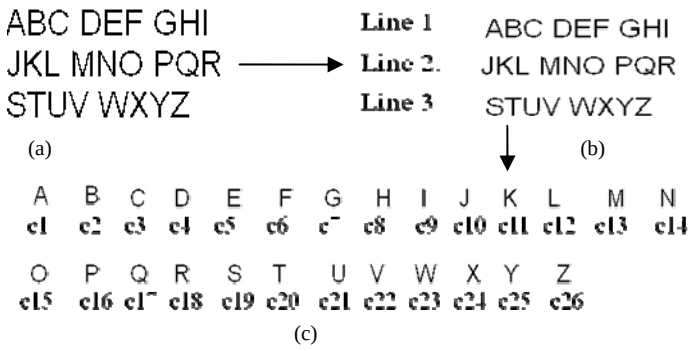

Fig. 2. (a) Original image of the text document (b) Segmentation of lines from the document (c) segmentation of letters from the lines.

3.1.3 *Skeletonization*

The characters segmented in the previous step are skeletonized i.e. thinned to unit pixel thickness. This process deletes the extra pixels which do not belong to the backbone of the letter. Therefore the skeletonized thin lined character will be taken as input to the feature extraction step.

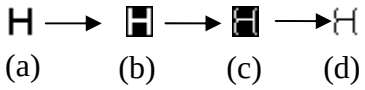

Fig. 3. (a) Original H in the document (b) Negation of the letter H (c) Skeleton of negative form of H(d) Negative of skeleton in negative form to get original skeleton of H.

3.2 ***Feature Extraction***

This phase will describe the process that extracts the number of vertices and chain codes between vertices by the help of the following subprocesses :-

3.2.1 *Vertex and code sequence generation from skeleton of a letter*

In order to extract those features, first of all we have started traversing the pixel intensity matrix (S_j) of the character from bottom left corner to find the first occurrence of a black pixel B. From B another traversing will be started for a vertex by checking 8 way neighborhood pixels Fig 3. Finally from that pixel (stored as the first vertex in the vertex array) again by the help of 8 way neighborhood checking, the letter is traversed and accordingly the chain code (direction of the next pixel to be traversed from its previous one) will be stored in an array(sequence array) [2].If more than 2 neighbor is encountered then put all the co-ordinates of the neighbors in a stack and take the last found neighbor as next pixel to be traversed if it is not already traversed previously. Also while traversing if a vertex is found then that co-ordinate will be stored in vertex array and a divide marker will be inserted to the chain code sequence to denote an end of the previous edge. Fig 4 shows the code generation process. The pseudo code for this task is described bellow

1. FOR each skeleton matrix s_j, where 1<=j<=m, [where m=total no.of letters in the document] REPEAT step 2 to 21.
2. Count total no. of black pixels =P
3. Find the occurrence of the first black pixel from bottom left pixel with its co-ordinate St_{ij}
4. Set staring_point=St_{ij}
   current_pixel=starting_point
5. Count no. of neighbour of current_pixel except previously traversed pixel (N)
6. WHILE N= =1 [Not a vertex] OR current_pixel!=starting_point [Reached to staring point i.e cycle] REPEAT step 7
7. Set current_pixel=neighbour of current_pixel.
8. END WHILE
9. Put the current_pixel co-ordinate into a stack C, set array code[]=0 head=1, vertex[]=0 v=1, t=1;[t=no.of pixel traversed]
10. WHILE C[top]!=0 AND top!=1 REPEAT step 10 to 19
11. Count no. of neighbours including the previously traversed pixel (N), Mark current_pixel as traversed i.e set it to 1 in its 3rd dimension of s_j.
12. Push the co-ordinates of all neighbours in C[] with the code of its position direction Sequence.
13. IF N>2 [current_pixel=junction point]
    I. Code[head]=99, head++, Vertex[v]=current_pixel, v++
    II. Current_pixel=C[top]
    III. IF current_pixel is traversed i.e 3rd dim of current_pixel = 1
       Top- -
       IF C[top]!=0
          i. Current_pixel=C[top]
          ii. t++

END IF

IV. ELSE
t++
Code[head]=Seq(current_pixel), head++

V. END IF

14. ELSE IF N= =1 [current_pixel=end point]

I. Code[head]= 99, head++, Vertex[v]=current_pixel,v++
II. Current_pixel=C[top]
III. IF current_pixel is traversed i.e 3rd dim of current_pixel = 1
Top- -
IF C[top]!=0
i. Current_pixel=C[top]
ii. t++
END IF
IV. ELSE
t++
Code[head]=Seq(current_pixel), head++
V. END IF

15. ELSE [current_pixel has only one neighbour]

I. Current_pixel=(C[top])
II. IF current_pixel is traversed i.e 3rd dim of current_pixel = 1
Top- -
IF C[top]!=0
Current_pixel=C[top]
t++

END IF
III. ELSE
t++
Code[head]=Seq(current_pixel), head++
IV. END IF

16. END IF(13)
17. If top==1 AND C[top]==0 AND t<P
compare $s_j(l,m,1)$ and $s_j(l,m,2)$
IF $s_j(l,m,1)$==0 and $s_j(l,m,2)$!=1
C[top]=$s_j(l,m,1)$, code[head]=99 , head++
END IF
18. END IF(17)
19. END WHILE(10)
20. Write code[] and vertex[] in 2 different .gif files.
21. END FOR (1)

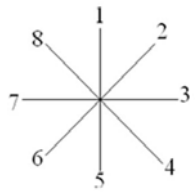

Fig. 3. code used for 8- way neighboring.

code: 99- 2- 99- 3- 3- …

Fig. 4. Matrix for the letter E. B is the first black pixel. In code , 99 is the end marker of one edge or get a vertex.

3.2.2 *Chain code modification (Achieving size independence)*

After creating the chain code, to make it general for letters of different sizes we have replaced a sequence of same code numbers with a single occurrence (squeezing)[3]. Also for the vertices that are very near to one another (distance between them is less than a threshold value t), second one of them is deleted from the vertex array. Accordingly the end markers in the chain code of the deleted vertices are also omitted from the code sequence.

Hence we can write as,

For a vertex v[x,y] remove all v[l,m] from vertex array such that

` |x-l|<|t|

And

|y-m|<|t|

3.2.3 *Counting of number of vertices*

Once the chain code sequence is modified completely, by the help of the end marker of edges in the sequence, number of vertices for that letter is count. Then the recognition process will be started depending on that number of vertices.

3.3 ***Recognition or identification***

In this phase each letter will be recognized and will be identified to a predefined character. The steps of that stage are group identification, matching of chain code sequence with all characters of that group one by one, accordingly number of hit are count, count the threshold of hit(h) to identify the letters, finally announce the letter to its identified character(with hit = h).

3.3.1 *Identification of group*

In the last stage number of vertices is count for the letter. Now depending on that count number the group is identified. Previously we have set 7 groups of letters where if number of vertices is more than 6 then it will fall on the group 7 otherwise it will fall in the group number according to its vertex count number.

if no_of_vertices >6 then
    Search in the superset of English alphabet i.e. from 26 letters
else search in the group containing letters of equal to the no_of_vertices.

3.3.2 *Matching the code sequence and threshold measuring of hit number and recognition of the letter*

Now depending on the group the code sequence of new input character is matched with the code sequence of the letters that belongs to that group. If the no of vertices is greater than 6 then the sequence will be matched with all 26 characters one by one. This matching will be done by extracting the sequence of one edge of predefined code and same from newly generated code. Accordingly matching number i.e. hit number will be count.

The code sequence matching process is done by using longest common subsequence (LCS)[1] counting i.e. sequence of one by one edge will be compared and the number of LCS will be stored. Also the length of the sequence of that edge of the predefined character and the new input character will be measured by which with the help of greater one the LCS out of that size will be stored as fractional hit number. Then the same process will be done next edge sequence and the hit number of previous edge will be added to new hit number. Finally after comparing with one total sequence a total hit number and a total fractional hit number will be count and it will be stored. Then the code sequence of new input character will be compared to the code sequence of the letter that is next in that group. Accordingly all the total hit numbers that are get by comparing with all the characters of that group are stored in an array sequentially.

From the array where we have stored the hit numbers of the selected group the maximum hit number of that array is count as hit threshold (h). This will be stored as hitrate i.e Hitrate = Max(hit). Now the

total hit number of matching with each letter is fetched one by one and it will be compared to the hitrate. If it is matched then that new letter can be said as identified to that predefined alphabet. Now if the hitrate matched with more than one letter then we have considered the fractional hit count. And depending on that hitrate and fractional hitcount we can announce the new input letter as one of the predefined character.

The pseudo code for this stage is described as follows:-

1. For the input character cj and group found k set top=0, hitarr[top]=0
2. For all letter matrix lki of the group k
3. Set hit=0, frachit=0;
4. For all edges cei of cj and edges ei of the i^{th} letter of group k [i=1 to no_of_letters in that group k]
5. Read code sequence seq1 of ei and seq2 of cei
6. Measure size of seq1 and seq2
7. C= LCS (seq1, seq2).
8. Count hit number hit1 and fractional hit for that edge and update hit and frachit as-
   I. hit1=max (C)
   II. hit=hit + hit1+1
   III. frachit= frachit+ (hit1/ maxsize(seq1,seq2)
9. End for(4)
10. hitarr[top]= hit ,
    top++
11. Store hit, frachit with i to identify hit of i^{th} letter.
12. End for (2)
13. Count hitrate =max(hitarr) and set match=0, top1=0, letter[top1]=0
14. For all lki
15. Read its hit by the help of i
16. If hit(lki)==hitrate
    I. match++
    II. letter[top1]=i
    III. top1++
    End if
17. End for(14)
18. If match ≤ 1
    Announce cj as lk_p [where p =letter[top1-1]]
19. Else
20. For all lk_p in letter[]
21. Read frachit of lk_p and lk_{p+1} from letter[top1] and letter[top1-1].
22. Compare and store the lk_p with greater frachit , top1- -
23. End for(20)
24. Announce the letter cj as lk_p having maximum frachit in letter[].
25. End for(1)

An example of the process is given below.

Hit=LCS(ei(I),ei(P)) [Where $1 \leq i \leq$ no_of_edges, ei = i^{th} edge, I= input character, P= predefined character]

TotalHit = $\sum$Hit(j) [where $1 \leq j \leq$ no_of_edges]

Fractional Hit = Hit/(Max(ei(I)))

For eg. Let sequence1 = 2 5 4 7 3 99 1 6 2 1 5 99 3 8 99
And sequence2 = 2 3 4 6 7 3 99 2 6 2 1 6 99 3 1 8 6 99

edge1.1= 2 5 4 7 3 edge1.2= 1 6 2 1 5 edge1.3= 3 8 (where edge1.i= i^{th} edge of sequence1)
edge2.1= 2 3 4 6 7 3 edge2.2= 2 6 2 1 6 edge2.3= 3 1 8 6(where edge2.i=i^{th} edge of sequence2)
lcs1= 2 4 7 3 lcs2= 6 2 1 lcs3= 3 8
Hit=4 Hit=3 Hit=2
frachit=0.66 frachit=0.6 frachit=0.5

TotalHit = 4+3+2= 9
Totalfractionalhit = 0.66+0.6+0.5 = 1.76

4. Experimental Results

We have run the algorithm to many styles of English characters that can be printed and get near about 95% accuracy on those styles. Also we have tested on different sizes of characters. As we have stated earlier that we Matlab is used to develop the code of the above stated algorithm. Table 1 shows the experimental results where after eight tests, average recognition accuracy of 95 % is obtained. Even in two cases, the correct recognition rate of 100% is achieved. We have set the reference set font as Times New Roman type with size 20.

TABLE 1. Results obtained from implementation of the proposed method.

Type of input document	Size of input document	Total characters in that document	Number of correctly identified characters	Recognition accuracy
Arial	20	26	26	100%
Times New Roman	20	26	26	100%
Arial	24	26	25	96.15%
Arial	16	26	24	92.31%
Tahoma	20	26	23	88.46%
Century Gothic	20	26	24	92.31%
Verdana	20	26	25	96.15%
Microsoft Sans Serif	18	26	24	92.31%
Average				95%

5. Conclusion

A chain code based approach with vertex set separation and LCS is used in this paper to identify printed English characters from a text document. The most important part of the algorithm is the vertices identification from a character and depending on that grouping of all 26 letters differently that makes the identification process easier and less time taken one. Experimental results are quite promising. In the future work we will try to make our algorithm workable for handwritten English text document. Also parallel approaches may be explored and can reform the document with recognized characters.

6. References

[1] Coreman T. H. ,*et al. Itroduction to Algorithms*, 2nd edn.
[2] Gonzalez. R. C, *Digital Image Processing* 3rd edn.
[3] Izakian H. *et al.(2008). Multi-Font Farsi/Arabic Isolated Character Recognition Using Chain Codes*, World Academy of Science, Engineering and Technology 43, pp. 67-70
[4] Pal .U, Dutta.S.(2003). *Segmentation of Bangla Unconstrained Handwritten Text* , ICDAR,7th ,pp.1128-1132.