Chapter 2

Document Segmentation : A Mathematical Approach

It is clear from the discussion in section 1.1 that once the document image is preprocessed it is subjected to segmentation. In this chapter we discuss application of mathematical techniques for document layout analysis and segmentation. As mentioned earlier, selection of most of these techniques are independent of the language, where as some others are script specific. This chapter covers discussion limited to application of mathematics in script independent techniques like Document layout analysis, Line / Word Segmentation, Connected Component extraction. In one of the sections we have discussed Gujarati script from the OCR perspective. A separate section is also devoted to justify the depth of segmentation required for Gujarati character recognition.

2.1 Document Layout Analysis

One of the expectations from a complete OCR system is that the output should retain the layout of document. In other words, the output should look same as the printed document. Therefore, it is necessary to analyze the image of printed document for its layout. Input to this process is a binarized document image and output will be coordinates of various blocks with their identification as text / non-text blocks. Figure 2.1 illustrates this.

XY-Cut and its variants are popularly used to carry out this task. In order to


સર્જક મધુ રાય

લાભશંકર ઠાકર

(a) A Typical Page for a Book

(b) Layout Labeling

Figure 2.1: Layout Analysis

explain XY-cut we need to define *Horizontal* and *Vertical projection.*

Horizontal Projection

Horizontal projection, say hp, of a binary image stores number of black pixels in each pixel row. Mathematically, for a binary image, I, of the size $M \times N$, horizontal projection

$$hp[i] = \sum_{j=1}^{N} I[i][j] \tag{2.1}$$

Vertical Projection

Similarly, *Vertical projection*, say vp, is defined as

$$vp[j] = \sum_{i=1}^{M} I[i][j] \tag{2.2}$$

Bounding Box

MBR

It is clear from the definitions that hp and vp will have as many elements as number of pixel rows and number of pixel columns in I, respectively. Another concept that is used frequently in the Document Image Analysis is of Bounding Box of a structural element of the document (text box, line, word etc.). *Bounding Box* is the smallest of all the rectangles enclosing the object under consideration. Hence it is also referred to as *Minimum Bounding Rectangle (MBR)*. Figure 2.2 illustrates this concept for words of document. The bounding boxes are normally identified by the coordinates of their top-left and bottom-right vertices.

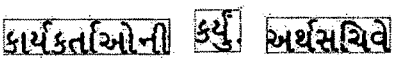


Figure 2.2: Bounding Box (MBR) of Words

The basic algorithm for XY-Cut starts with analysis of horizontal or vertical projection of a document image (see Fig. 2.3). It looks for comparatively large white space in the projection. Then the algorithm considers each block separated by the white space in the projection and finds other projection. The process continues till none of the projection results into new block. The output of this stage of processing is used to rearrange the recognized text and the image extracted non-text parts while reconstructing the document.

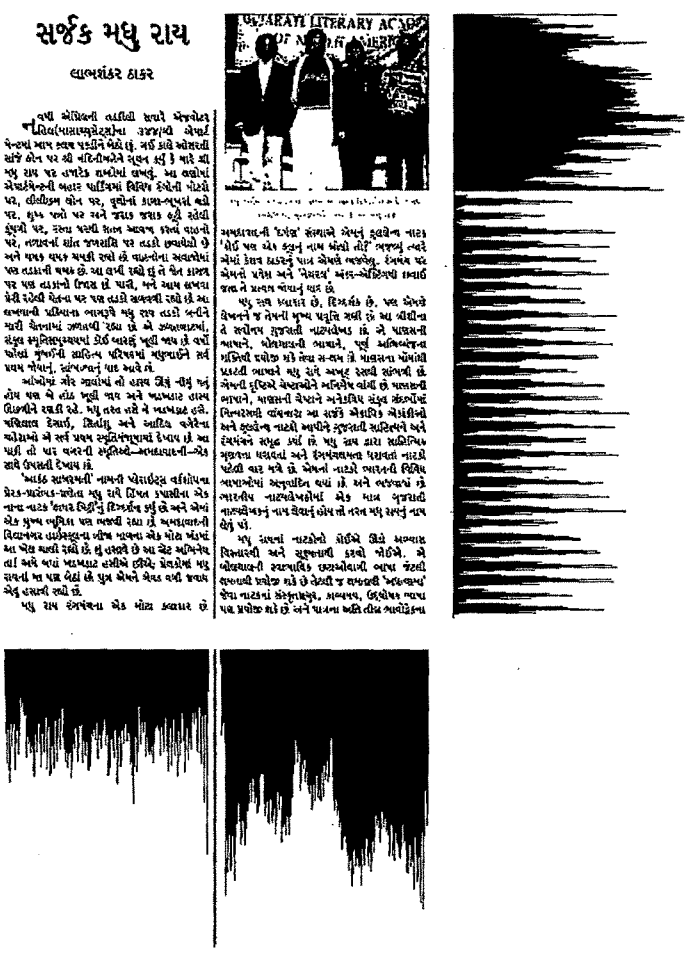


Figure 2.3: A Page with Its Vertical and Horizontal Projections

Once the text blocks in the documents are identified then the next task is

to analyze them for other building blocks like lines, words, connected components etc.. Some of the researchers select Top-down approach and the others go for Bottom-up approach for this task (Table 2.1). Top-down approach, as the name suggests, starts breaking the bigger blocks of text into smaller ones. That is, a block of text is analyzed for the line, segmented lines are analyzed for words and words for connected components. The bottom-up approach starts by finding all the connected components of a text block which are then group together(according to some criteria) to form bigger blocks like word and then words are grouped to form line.

Table 2.1: Segmentation Approaches

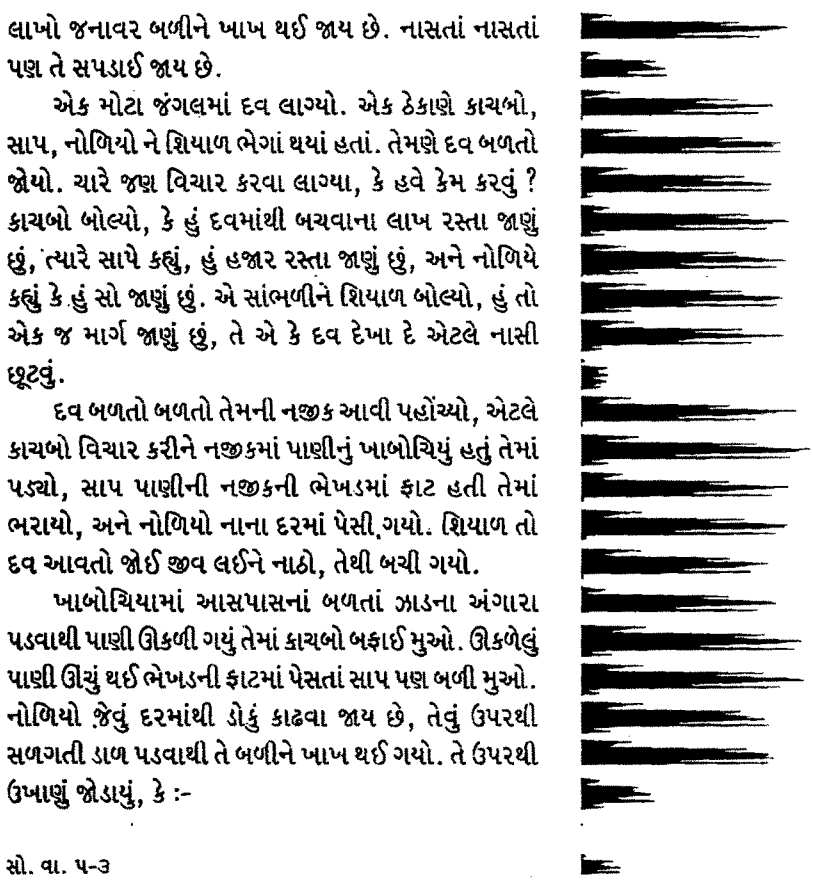
Top Down	Bottom UP
Text block	
↓	↑
Line Segmentation	Line Formation
↓	↑
Word Segmentation	Word Formation
↓	↑
Connected Component	Connected Component

Here, we have followed top-down approach and following sections describe mathematics used for carrying out various tasks in this approach.

2.2 Line Segmentation

There are several references available in the literature for line segmentation of documents of Indic script, see for example [6], [10]. In [26] Jindal *et. al.* present technique to segment horizontally overlapped lines. In this work we have selected one of most used methods for line segmentation where in the white space in the horizontal projection of the text block is used as a text line delimiter.

Figure 2.4 shows a page with its horizontal projection. it can be noticed that the inter line gaps in the printed text produces space in the horizontal projection of the text block. Analysis of the horizontal projection gives row coordinates of the bounding box for lines. The **Algorithm 2.1** describes the computational procedure for line segmentation. We are initializing y-coordinate of top left vertex of the MBR as 0 and that of bottom right vertex with n, the width of the text block in pixels.



લાખો જનાવર બળીને ખાખ થઈ જાય છે. નાસતાં નાસતાં પણ તે સપડાઈ જાય છે.

એક મોટા જંગલમાં દવ લાગ્યો. એક ઠેકાણે કાચબો, સાપ, નોળિયો ને શિયાળ ભેગાં થયાં હતાં. તેમણે દવ બળતો જોયો. ચારે જણ વિચાર કરવા લાગ્યા, કે હવે કેમ કરવું ? કાચબો બોલ્યો, કે હું દવમાંથી બચવાના લાખ રસ્તા જાણું છું, ત્યારે સાપે કહ્યું, હું હજાર રસ્તા જાણું છું, અને નોળિયે કહ્યું કે હું સો જાણું છું. એ સાંભળીને શિયાળ બોલ્યો, હું તો એક જ માર્ગ જાણું છું, તે એ કે દવ દેખા દે એટલે નાસી છૂટવું.

દવ બળતો બળતો તેમની નજીક આવી પહોંચ્યો, એટલે કાચબો વિચાર કરીને નજીકમાં પાણીનું ખાબોચિયું હતું તેમાં પડ્યો, સાપ પાણીની નજીકની ભેખડમાં ફાટ હતી તેમાં ભરાયો, અને નોળિયો નાના દરમાં પેસી ગયો. શિયાળ તો દવ આવતો જોઈ જીવ લઈને નાઠો, તેથી બચી ગયો.

ખાબોચિયામાં આસપાસનાં બળતાં ઝાડના અંગારા પડવાથી પાણી ઊકળી ગયું તેમાં કાચબો બફાઈ મુઓ. ઊકળેલું પાણી ઊંચું થઈ ભેખડની ફાટમાં પેસતાં સાપ પણ બળી મુઓ. નોળિયો જેવું દરમાંથી ડોકું કાઢવા જાય છે, તેવું ઉપરથી સળગતી ડાળ પડવાથી તે બળીને ખાખ થઈ ગયો. તે ઉપરથી ઉખાણું જોડાયું, કે :-

સો. વા. ૫-૩

Figure 2.4: Line Segmentation

It is clear from the algorithm that it does not take care of the cases where in there is an overlap between lines i.e. the some part of a line goes into the bounding box of the line below or above. In this case the horizontal projection of the text block around those lines will not have zero frequency run between the region corresponding to two overlapping lines. For this work, we have considered documents with clear line separation.

2.3 Word Segmentation

Once the bounding box of the line is identified the pixels falling in that box will be extracted as line pixels and processed further. Mathematically, it is a sub matrix of the pixel matrix of the text box i.e. if $TB_{m \times n}$ is pixel matrix

Algorithm 2.1 To Find Line Boundary

Input: Text block from a binarized image, TB_{ij} of a Gujarati document with m pixel rows and n pixel columns

Output: A 2D array $line[1, 2, \ldots, noLines][2]$ where each row stores pixel row numbers of the two lines forming x-coordinates of the top left and bottom right coordinates of its bounding box, where *noLines* represents number of text line in the text block.

Step 1: Compute horizontal projection

$$hp[i] = \sum_{j=1}^{n} TB[i][j], \quad \forall i = 1, 2, \ldots, m. \tag{2.3}$$

Step 2:

```
in ← false
out ← true
cLine ← 0
for i = 1 to m do
    if hp[i] > 0 then
        if out then
            cLine ← cLine + 1
            line[cLine][1] ← i
            out ← false
            in ← true
        end if
    else
        if in then
            line[cLine][2] ← i
            out ← true
            in ← false
        end if
    end if
end for
```

of text box and $line_i$ is the matrix representing the pixels of the i^{th} line then

$$[line_i]_{p \times q} = TB[line[i][1] \cdots line[i][2]][n] \tag{2.4}$$

Obviously, $p = line[i][2] - line[i][1]$ ($line[\cdot][\cdot]$ as in **Algorithm 2.1**) and $q = n$.

The algorithm for word segmentation works on similar lines as that of line segmentation. Here, vertical projection of the line is used to detect the inter

word gap and hence, for deciding the word boundary. Figure 2.5 shows three lines with their vertical projections.

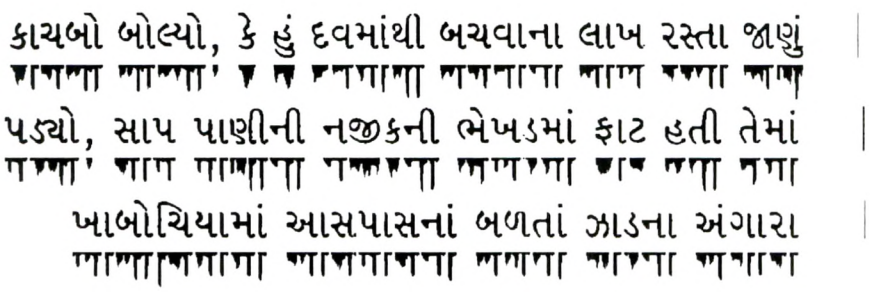


Figure 2.5: Word Segmentation

As shown in Fig. 2.5 inter character spacing in the printing also gives rise to zero frequency point in the vertical projection.However, it can also be noticed that the run of such points are much lesser than the points in vertical projection which belong to an inter word spacing. This property is used to differentiate between two types of spaces. Run Length Smearing Algorithm (RLSA) [44] is one of the techniques to remove the space between the characters / glyphs of the same word.

Figure 2.6: Result of Run Length Smearing Algorithm

RLSA extends the run of the black pixel to next black pixel in the same pixel row of a binarized image if the number of white pixel between them is less than a predetermined threshold. Figure 2.6 shows result of application of this algorithm on a text line.

The out put of the word segmentation algorithm is bounding box information for word of the line under consideration.

2.4 Connected Component Extraction

Connected component labeling (alternatively connected component analysis) is an algorithmic application of graph theory, where subsets of connected components are uniquely labeled based on a given heuristic. Some of the graph theoretic concepts used in this algorithm are *Adjacency* and *Connectivity*. These concepts are defined in the context of digital images. The definitions

are taken from [19].

Let V be the set of intensity values used to define adjacency. In a binary image, $V = \{1\}$ if we are referring to adjacency of pixel with value 1. In a gray scale image, the idea is the same, but set V typically contains more elements. For example, in the adjacency of pixels with a range of possible intensity values 0 to 255, set V could be any subset of these 256 values. There are three types of adjacency which are considered :

1. Two pixels p and q with values from V are *4-adjacent* if q is in the set $N_4(p)$ — 4-adjacency
2. Similarly all pixels in $N_8(p)$ are called *8-adjacent* to p.
3. Two pixels p and q from V are called *mixed adjacent (m-adjacent)* if
   a. q is in $N_4(p)$, or
   b. q is in $N_D(p)$ and the set $N_4(p) \bigcap N_4(q)$ has no pixels whose values are from V.

A *(digital)path* from a p with coordinates (x, y) and q with coordinates (s, t) is the sequence of distict pixels with coordinates — Path

$$(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)$$

where $(x_0, y_0) = (x, y)$, $(x_n, y_n) = (s, t)$ and pixels (x_i, y_i) and (x_{i-1}, y_{i-1}) are adjacent for $1 \leq i \leq n$. In this case, n is the *length* of the path. if $(x_0, y_0) = (x_n, y_n)$, the path is said to be a *closed* path. We can define 4-,8- or m-path depending on the type of adjacency specified.

Let S represent a subset of pixels in an image. Two pixels p and q are said to be *connected* if there exists a path between them consisting entirely of pixels in S. For any pixel p in S, the *set* of pixels that are connected to it in S is called *connected component* of S. — Connected Pixels; Connected Component

Here, in this work S is set of all pixels of a word and we find connected component from that sub image. The algorithm we used for this work is a two pass algorithm.One of the two passes traverses the array from top to bottom and the other traverses it from left to right. In the first pass pixels are labeled following certain scheme and some of the labels are identified as equivalent. In second pass Redundancies are removed, equivalent labels are made equal and thereby identify different connected components as sets having all pixels

with same labels.

The broad outline of the procedure to find connected components is described below and the details are given in **Algorithm 2.2**. Fig. 2.7 shows an example of the typical input to connected component extraction and Fig. 2.8 shows the result after applying **Algorithm 2.2**.

To Find Connected Components

Input: A pixel array, say $pixel_{h \times w}$, of a word sub-image (binary). Where $pixel[i][j] = 1$ for the pixels part of connected component corresponding glyphs of the word and it is equal to 0 for background pixels. Labels start from 2.

Output: Same array with connected components labeled.

Process: First Pass : Labeling

i. For each pixel check if the pixel above and before is labeled.

ii. If only one of them is labeled then the current pixel takes that label.

iii. If both of them are labeled then give current pixel any one of the labels and record the two different labels as equivalent labels (labels of the same connected component).

Second Pass : Processing Equivalents

i. Find the distinct pairs of equivalents by removing the redundancy.

ii. Make equivalent labels equal.

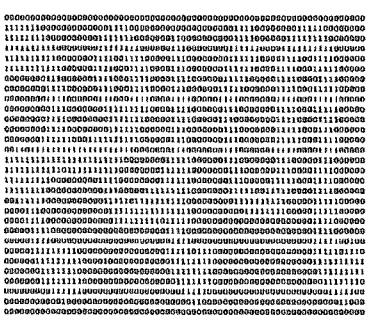


Figure 2.7: Input to Connected Component Labeling

Algorithm 2.2 To Extract Connected Components

Step 1: Labeling

```
IN ← 1
OUT ← 0
status ← OUT
Initialize equivalents
noEquals ← −1
currentLabel ← 2
for i = 0 toh do
    for j = 0 to w do
        if tpixels[i][j] = 1 then
            if status = IN then
                if i = 0 then
                    tpixels[i][j] ← currentLabel
                else
                    tpixels[i][j] ← tpixels[i][j − 1]
                    if tpixels[i−1][j] ≠ tpixels[i][j−1] and tpixels[i−1][j] >
0 then
                        noEquals ← noEquals + 1
                        if tpixels[i][j − 1] < tpixels[i − 1][j] then
                            equivalents[noEquals][0] ← tpixels[i − 1][j]
                            equivalents[noEquals][1] ← tpixels[i][j − 1]
                        else
                            equivalents[noEquals][0] ← tpixels[i][j − 1]
                            equivalents[noEquals][1] ← tpixels[i − 1][j]
                        end if
                    end if
                end if
            else
                status ← IN
                if i = 0 then
                    tpixels[i][j] ← currentLabel
                else
                    if tpixels[i − 1][j] ≠ 0 then
                        tpixels[i][j] ← tpixels[i − 1][j]
                    else
                        tpixels[i][j] ← currentLabel
                    end if
                end if
            end if
```

```
        else
            if status = IN then
                status ← OUT
                currentLabel ← currentLabel + 1
            end if
        end if
    end for
    if status = IN then
        currentLabel ← currentLabel + 1
        status ← OUT
    end if
end for

Step 2: Processing Equivalents
pureEquals ← noEquals
for i = 0 to noEquals − 1 do
    if equivalents[i][0] ≠ 0 then
        for j = i + 1 to noEquals do
            if equivalents[i][0] = equivalents[j][0] and equivalents[i][1] =
equivalents[j][1] then
                equivalents[j][0] ← 0
                equivalents[j][1] ← 0
                pureEquals ← pureEquals − 1
            end if
        end for
    end if
end for
for i = 0 to noEquals do
    if equivalents[i][0]=0 then
        continue
    end if
    if equivalents[i][0] = equivalents[i][1] then
        continue
    end if
    fe ← equivalents[i][0]
    se ← equivalents[i][1]
    for j = 0 to h do
        for k = 0 to w do
            if tpixels[j][k] = fe then
                tpixels[j][k] ← se
            end if
```

```
        end for
    end for
    for p = i + 1 to noEquals do
        if equivalents[p][0] = 0 then
            continue
        end if
        if equivalents[p][1] = fe then
            equivalents[p][1] ← se
        end if
        if equivalents[p][0] = fe then
            equivalents[p][0] ← se
        end if
        if equivalents[p][1] = equivalents[p][0] then
            equivalents[p][1] ← equivalents[p][0] ← 0
        end if
        if equivalents[p][1] > equivalents[p][0] then
            t ← equivalents[p][0]
            equivalents[p][0] ← equivalents[p][1]
            equivalents[p][1] ← t
        end if
    end for
end for
```

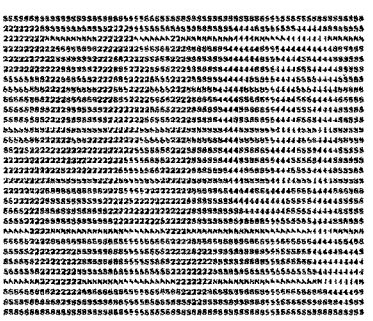


Figure 2.8: Output of Connected Component Labeling

2.5 Depth of Segmentation

Earlier sections described various mathematical techniques used commonly in the document segmentation. We define *depth of segmentation* as the extent up to which segmentation is done. In other words, it will determine where the segmentation process will stop - at line level or word level or at connected component level or any other level of segmentation. This decision will also determine the unit of recognition (the symbol which will be fed to recognition system). It is important to note here that the decision about depth of seg-

mentation is also driven by selection of unit of recognition. Hence, we analyze Gujarati script to decide about the unit of segmentation and hence depth of segmentation.

There may be two different approaches to recognition :

1. Recognizing a (C)CV combination as a whole (as explained in previous section)
2. First segmenting the consonants from dependent vowel modifier and then recognizing them separately. The recognized text has to be composed by reconstructing the (C)CV cluster based on combining the recognized glyphs.

(a) CV combination

(b) Symbols in Three Zones

Figure 2.9: Two Possible Approaches For Recognition

Selecting first option will result into exorbitantly large number of symbols to be recognized. Total number of symbols to be recognized in this case would be:

34 consonant + 5 vowel symbols + 34 consonants × 12 vowel modifiers + approximately 250 conjunct × 12 vowel modifiers = 3500 approximately symbols

Such a large number of symbols poses a great challenge of designing large class classifier and it is known that designing a classifier to handle such a large number of classes is a very difficult task. Now, if we choose to recognize components in each zone separately, the total number of components to be identified are **middle zone components**(consonants + conjuncts + vowel modifier corresponding to /AA/ + half forms of consonants + half form of conjunct) + **upper zone symbols** (upper parts of the vowel modifiers corresponding to /i/, /ii/, /e/, *Anuswar*, *repha*, *Chandrabindu*) + **lower zone symbols** totaling to not more than 350.

Hence, It is clear that second option is more feasible as the number of classes reduces significantly. That means, if we want to avoid known complexities

of designing classifier with large number of classes, we will have to carry out slightly complex process of zone boundary detection and then recognizing symbols in each zones. Since this is the choice we make, it is then necessary to have a robust zone boundary separator which can reliably identify zone separation boundaries. Further, before we start connected component extraction from a word, zone separator algorithm identifies and disconnects zones of the word. As a part of this work we have devised an algorithm for this task using some of the mathematical techniques. Following two chapters are devoted to discussion of these techniques.

2.6 Conclusions

We have discussed various mathematical techniques like projection profiles and graph theoretic connected component extraction that are applied for various subtasks of document segmentation. Algorithms used for each of these subtasks are presented. A thorough analysis of Gujarati script is also presented for deciding the unit of recognition. The analysis revealed that it is better to recognized components in each of the three logical zones separately. This justifies the need of a robust zone boundary identification technique.