Solving Captchas with an OCR system

Sebastian Agethen, D01944015
Lin Sheng-Ching, R99222030
Jeroen Dhondt, A01922201

June 19, 2013

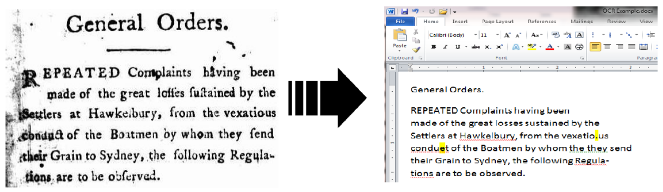
Contents

1 Introduction

Optical Character Recognition (OCR) has been around for quite some time as a research field in Image Processing. We found this idea interesting, and thus decided to base our project around OCR. As a new idea and modern application of this topic, we choose to try and solve Captchas. In this introduction we will first introduce the concept of Captchas as well as their motivation and modern day uses, an introduction to the topic of OCR is presented and followed by our motivation to start this project. In Section 2 we go in depth over our implementation, covering all different aspects in detail. Next, in Section 3 we've summed up our results followed by a discussion. Our conclusions are presented in Section 4, and the work division is covered in the final Section 5.

1.1 Optical Character Recognition

To start this introduction, we give more insight in the process called OCR. OCR is a term used to discribe computer applications that have as goal to recognize characters from a (digital) image. As such, these applications have a collection of image processing steps and decision making algorithms that together make it possible to find and correctly identify characters in an image. An example can be seen in Figure **??**. Traditionally, the most common application of OCR nowadays was digitazing paper resources, for instance old books in libraries or old files in companies. But nowadays a whole new array of applications has risen. Most noteworthy is the extraction of information from images and even video, for instance recognizing license plates on a traffic video feed or analyzing pictures taken Google Maps to recognize traffic signs, house numbers and others.



General Orders.

REPEATED Complaints having been made of the great loſſes ſuſtained by the Settlers at Hawkeſbury, from the vexatious conduct of the Boatmen by whom they ſend their Grain to Sydney, the following Regulations are to be obſerved.

General Orders.

REPEATED Complaints having been
made of the great losses sustained by the
Settlers at Hawkelbury, from the vexatio.us
conduet of the Boatmen by whom the they send
their Grain to Sydney, the following Regula-
tions are to be observed.

Figure 1: Example of OCR

1.2 Captchas

For our project, we decided to analyze captchas, and ultimately try to solve ('crack') these puzzels. Captchas were invented to solve the problem of so-called 'crawlers' or 'bots' that are found on the web. These automatic programs typically try to spam a website or forum with advertisements or do other unwanted actions. To stop these, captchas were introduced: they are a test to users to prove they are human. These captchas are images with characters hidden in them, in a way that they can be read by humans but are hard to distinguish by a computer. Some examples are shown in Figure **??**

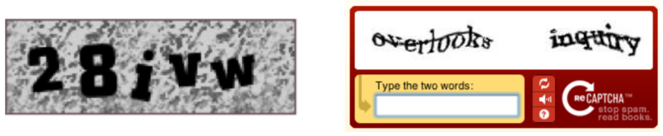


Figure 2: Different captchas

1.3 Motivation

The motivation for our project is to implement an application that can solve captchas, presented as an image file. To get this result, we will need to implement two parts. First, a preprocessing of the captcha will be done. Any distorted elements that obstruct the OCR application from recognizing the characters, must be removed. The second and most important part will be the OCR application. Our application must be able to recognize characters found in a given bitmap input image. Due to time constraints, we have only implemented OCR and let the preprocessing of captchas as future work.

2 Implementation

We first introduce the structure of our system. In the following sections we then present each step in greater detail.

2.1 General structure

Our OCR system runs in two stages, a training phase and a live phase. In the training phase, each our our features is evaluated on a set of 36 Bitmaps describing the characters we can recognize. Currently, this encompasses the 26 capital characters of the English alphabet as well as the digits 0 to 9. In the second phase, called live phase, we extract segments from an input file and then compute all features for each segment. Finally, training data and live data are passed together with a weight vector to a clustering method, which finds either the closest or the k-closest matching results.

2.2 Preprocessing

2.2.1 Quantization

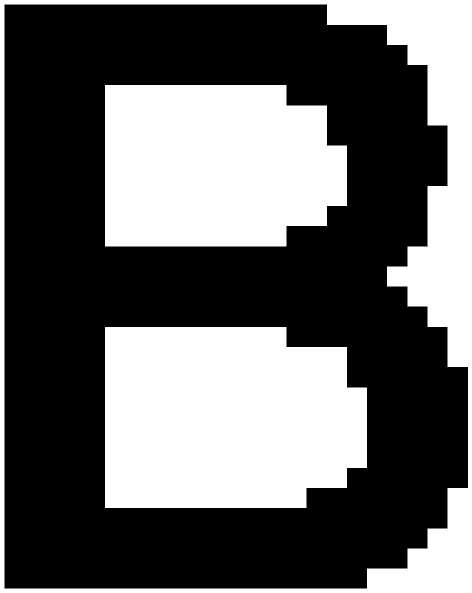
For our quantization, we first set a threshold value t, and then evaluate the Eq. **??**. If the term $v < t$, then we set the pixel $p = 0$, otherwise if $v \geq t$, we set the pixel $p = 1$.

$$v = \sqrt{r^2 + g^2 + b^2} \tag{1}$$

2.2.2 Skeletonization

After the image is quantized, we repeatedly use the conditional mark pattern and unconditional mark pattern until the nothing is changed. Finally, we use the bridge operation to recover the connectivity of the character. Conditional and unconditional mark patterns for Skeletonization can be found in Fig. **??** to **??**, while those for the Bridge operation can be found in Fig. **??**.

The comparison of the skeletonization can be seen in the Fig. **??**.



(a) Original B

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0
0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0
0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 0 0 0 0 0 0
0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0
0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0
0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0
0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0
0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0
0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0
0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 0 0 0 0 0 0
0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0
0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0
0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0
0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0
0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0
0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0
0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0
0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0
0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0
0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0
0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0
0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 0 0 0 0 0
0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0
0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0
0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0
0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

(b) Quantised B

Figure 3: Original B v.s. Quantised B

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0
0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0
0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 0 0 0 0 0 0
0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0
0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0
0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0
0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0
0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0
0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0
0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 0 0 0 0 0 0
0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0
0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0
0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0
0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0
0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0
0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0
0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0
0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0
0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0
0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0
0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0
0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 0 0 0 0 0
0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0
0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0
0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0
0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

(a) Quantised B

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 1 1 0 0 0 0 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

(b) Skeletonized B

Figure 4: Qunatized B v.s. Skeletonized B

2.2.3 Boundary extraction

The algorithm to extract the boundary of character can be seen in the listing **??**.

Listing 1: Boundary Extraction algorithm

```
1. Scan the pixel from left to right for
   each row.
2. When the gradient changes strictly , set
    the row number as the ceiling of the
   sentence.
3. After finding the ceiling , keep
   scanning the pixels. If we find a
   totally white row, set the row number
     1 as the bottom of the sentence.
4. Scan the pixels of each column from the
    ceiling to the bottom.
5. When the gradient changes strictly , set
    the column number as the left
   boundary of the sentence.
6. After finding the left boundary, scan
   the pixels from the rightmost pixel to
    the left , from ceiling to the bottom.
    As we find a strict gradient of the
   column, set it as the right boundary.
7. Keep doing the steps 1 ~ 6 until all
   the input file is scanned.
8. For each sentence, set the left
   boundary as the left boundary of the
   first chacter and scan the pixel from
   ceiling to the bottom, starting from
   the left.
9. If there is a column totally white, set
    the column number  1 as the right
   boundary of the chacter.
10. Keep scanning until a strict gradient
   occurs. Set the column number as the
   left boundary of the next chacter.
11. Keep scanning until another strict
   gradient column is found and the right
    boundary of the chacter is set.
12. Repeat step 9 ~ 11 until the sentence
   is totally scanned and scan the next
   sentence.
```

2.2.4 Orientation

The followings are the derivation of the orientation angle of an object: Define the (m,n)th spatial moments

$$M(m,n) \quad = \quad \frac{1}{J^n K^m}\sum_{j=1}^{J}\sum_{k=1}^{K}(x_k)^m (y_j)^n F(j,k) \quad (2)$$

where $x_k = k - \frac{1}{2}, y_j = J + \frac{1}{2} - j$ and

$$U(m,n) = \frac{1}{J^n K^m}\sum_{j=1}^{J}\sum_{k=1}^{K}\left(x_k - \frac{M(1,0)}{M(0,0)}\right)^m \left(y_j - \frac{M(0,1)}{M(0,0)}\right)^n F(j,k) \quad (3)$$

where $x_k = k - \frac{1}{2}, y_j = J + \frac{1}{2} - j$ and then find the maximum value of eigen value

$$E^T U E = \Lambda \quad (4)$$

$$E = \begin{bmatrix} e_{11} & e_{12} \\ e_{21} & e_{22} \end{bmatrix} \quad (5)$$

$$\Lambda = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} \quad (6)$$

By calculation,

$$\lambda_1 = \frac{1}{2}\left[U(2,0) + U(0,2)\right] + \frac{1}{2}\left[U(2,0)^2 + U(0,2)^2 - 2U(2,0)U(0,2) + 4U(1,1)^2\right]^{1/2} \quad (7)$$

$$\lambda_2 = \frac{1}{2}\left[U(2,0) + U(0,2)\right] - \frac{1}{2}\left[U(2,0)^2 + U(0,2)^2 - 2U(2,0)U(0,2) + 4U(1,1)^2\right]^{1/2} \quad (8)$$

$$\rightarrow \quad \theta \quad = \quad \arctan\left[\frac{\lambda_M - U(0,2)}{U(1,1)}\right] \quad (9)$$

2.2.5 Convex Hull

We define the *Convex Hull* of a character as follows: Given that the input is already quantized to black-and-white, where white is the background, any point that is on a line between any two black points has to be black, too. More formally, this can be expressed as:

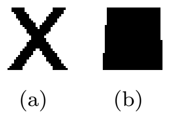


Figure 5: Convex Hull for the character 'X'. a) Original character, b) Convex Hull of character

$$\forall v_1, \forall v_2 \in \mathrm{B} : v_t \in \mathrm{C}, \quad v_t = (1-t)v_1 + tv_2, t \in [0,1] \quad (10)$$

where B is the set of black pixels and C is the Convex Hull. An example can be found in Figure **??** for the character 'X'.

2.3 Features

In the following we present each feature we used in detail.

2.3.1 Geometrical features

Our geometical features consist of area, weight center, average and maximum distance from weight center.

Area Calculate the total number of pixels

Weight center

$$F_c = \frac{\sum F(i,j) x_{i,j}}{\sum x_{i,j}} \quad (11)$$

Average distance from weight center

$$d = \sqrt{\frac{\sum_{i,j}(x_{i,j} - F_c)}{\sum_{i,j} x_{i,j}}} \quad (12)$$

Maximum distance from weight center

$$d_M = MAX(\sqrt{x_{i,j} - F_c}) \quad (13)$$

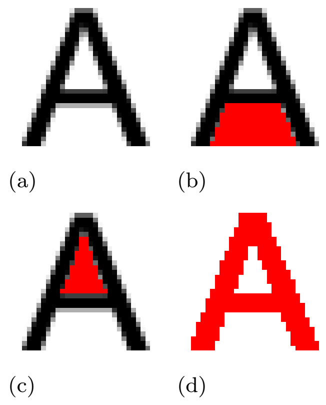


Figure 6: a) Original character, b) 'A' has a single bay (colored red), c) 'A' has one lake, d) 'A' has one Connected Component. Thereby 'A' has Euler number 0

2.3.2 Bays, Lakes and the Euler number

Each character posseses a property called the *Euler number*, which we define as the number of *Connected Components* minus the number of *Lakes*.

In a bitmap that consists of only two colors (e.g., black-and-white), we define a *Lake* to be a connected area which does not reach the borders of the *Convex Hull* (see Section **??**) and is colored with the background color (e.g., white). A related term, the *Bay* is defined to be a connected area of background color that does touch the borders of the *Convex Hull*. The last term is that of a *Connected Component*. For our purposes it will describe any connected area of non-background color (e.g., black). An example for each term is given in Figures **??** a) to d) with the capital character 'A' .

We compute these features in the following fashion: We sequentially scan the given image for white (Bays and Lakes) or black (Connected Components) pixels. Once such a pixel is found, we then execute a coloring algorithm that recursively visits neighbors of the same color and then colors each visited pixel in a third color (e.g., a greytone). If the coloring algorithm is run on a white pixel, we also determine whether we reached the border of the bitmap during execution by setting a flag `reachedBorder`. Furthermore, after each instance we increase a counter variable for the corresponding type, e.g., `numberOfLakes`.

This algorithm visits each pixel at most twice, since the coloring algorithm only visits pixels that have not been colored yet, and therefore requires linear time $O(n)$.

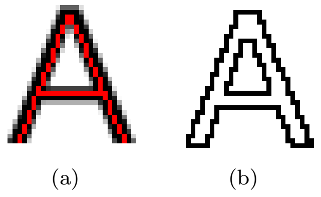


Figure 7: a) In Red: Skeleton of 'A' having one horizontal line, b) Boundary Extraction: Two horizontal lines

2.3.3 Line and Circle Components

Another important property of lating characters are the number of straight lines and the number of circles within a *Connected Component*. For an example, please refer to Figure **??**: The character 'A' is 0 vertical lines, 1 horizontal line and 0 circles. Note that we currently do not count diagonal lines. However, we reserve this feature for our Future Work.

We use a very simple method to count straight lines: We first search a pixel that is not of background color. Once it is found, we then visit the next pixel in one direction recursively and thereby determine the length of a possible line. For example, to compute horizontal lines and given that the current pixel is located at (x,y), we visit (x+1,y), (x+2,y) and so on until a different color is encountered. We then count the line if its length meets a predetermined threshold.

A major issue with this algorithm arises when lines have a thickness of more than one pixel: In that case, for varying sizes of characters, varying number of lines would be found! It is therefore imperative to first do preprocessing. We suggest to use either *Skeletonization* or *Boundary Extraction*. When using *Skeletonization*, each line of a character is shrunk to a thickness of 1, enabling easy parsing. When using *Boundary Extraction* however, two lines are (usually) created. In our implementation, we have used *Boundary extraction*.

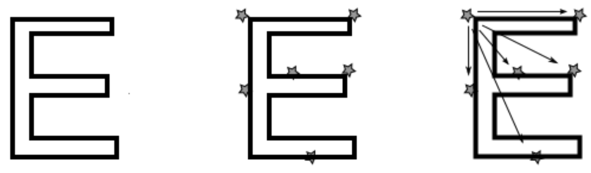


Figure 8: Sampling process: Left: Perform Boundary Extraction, Center: Choose a fixed number of random samples, Right: Compute log-distances between samples

We can also count the number of circles in a character. To do this, we find a *Lake* (see Sec. **??**) and determine the following formula:

$$C_0 = \frac{4\pi A_0}{P_0^2} \tag{14}$$

where A is the area of the Lake and P is the *Perimeter*, i.e., the number of neighboring pixels.

2.3.4 Shape Context

We include a method proposed by Belongie et al. [**?**] as a feature: Shape Descriptors. A Shape Descriptor is a single value that describes how similar two shapes are, where a value is 0 when two shapes are identical, and > 0 otherwise.

The authors construct this value by sampling a picture, computing the distances between samples and then creating a histogram for each sample. When comparing two shapes, the Chi-Square distance

$$C_S = \frac{1}{2}\sum_{k=1}^{K}\frac{[g(k)-h(k)]^2}{g(k)+h(k)}$$

is computed for each pair of histograms of shape A (g in the formula above) with histograms of shape B (h). This would result in n^2 values! The suggested solution is to find a bipartite matching between samples of shape A and B and then compute the sum of all matched chi-square distances.

2.4 Decision mechanism

Once every feature has been evaluated on all segments, we need to decide which character a segment resembles. In our system, we use a simple clustering algorithm.

2.4.1 Weight vector

An important point to notice is that above features are not normalized. The *Euler number* of a character may range between -1 and 1, while the total black area of a segment may easily have 300 pixels for letter of font size 32. Normalizing all values would improve results greatly. However, another problem occurs: In reality, the Euler number distinguishes a character far more than the area. We would like to **weigh** the Euler number higher than the area. To achieve this, we must therefore include a weight-vector.

2.4.2 Clustering

Once this is done, the clustering algorithm is very straightforward: For each segment a vector is created, where each dimension of the vector resembles another feature. Each component of a vector is furthermore multiplied by the corresponding entry in the weight-vector.

We then choose the vectors found in our training phase to be the cluster centroids. For all other vectors, namely those found in the live phase, we compute the distances to all cluster centroids and then assign it to the closest centroid. The advantage of this method over, e.g., a decision tree is that we can also support k-next-neighbor request, which is useful for post-processing. As we currently do not do any post-processing, we keep this matter for our Future Work.

3 Experimental Results

We evaluate our program on the text seen in Figure **??**. The text contain Capitals form the English alphabet as well as the digits 0 to 9.

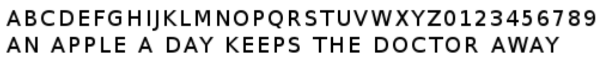

ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789
AN APPLE A DAY KEEPS THE DOCTOR AWAY

Figure 9: Input Bitmap for our evaluation

Listing 2: Result of our evaluation

```
Detected:
ABZDEFDHG2SLNNORORETKXQWYZD5ZB4ZB5BB
Diff:
  C    G IJK M   PQ S UVWX   0123  56789
(22 Errors)

Detected:
ANARRLEADAYSEERETHEDOZTORAQAY
Diff:
    PP          K    PS         C       W
(7 Errors)
```

The output of our OCR system can be seen in Listing **??**. From the total of 65 characters, we did not correctly recognize 29, giving an overall error-ratio of 44.6%. We notice that with our current features, the recognition is far better for alphabetical characters than for numbers. From ten digits only one was correctly identified (Error ratio: 90%), while of the 55 alphabetical characters 35 were correctly identified (Error ratio: 36.3%).

3.1 Discussion

We now discuss the known flaws of our system leading to the low recognition rate seen above. First of all, some of the features are not well implemented resulting in incorrect values, e.g., our Number of Bays feature. The problem here is that during the computation of the Convex Hull some pixels are falsely included. These pixels then form small Bays that distort our results.

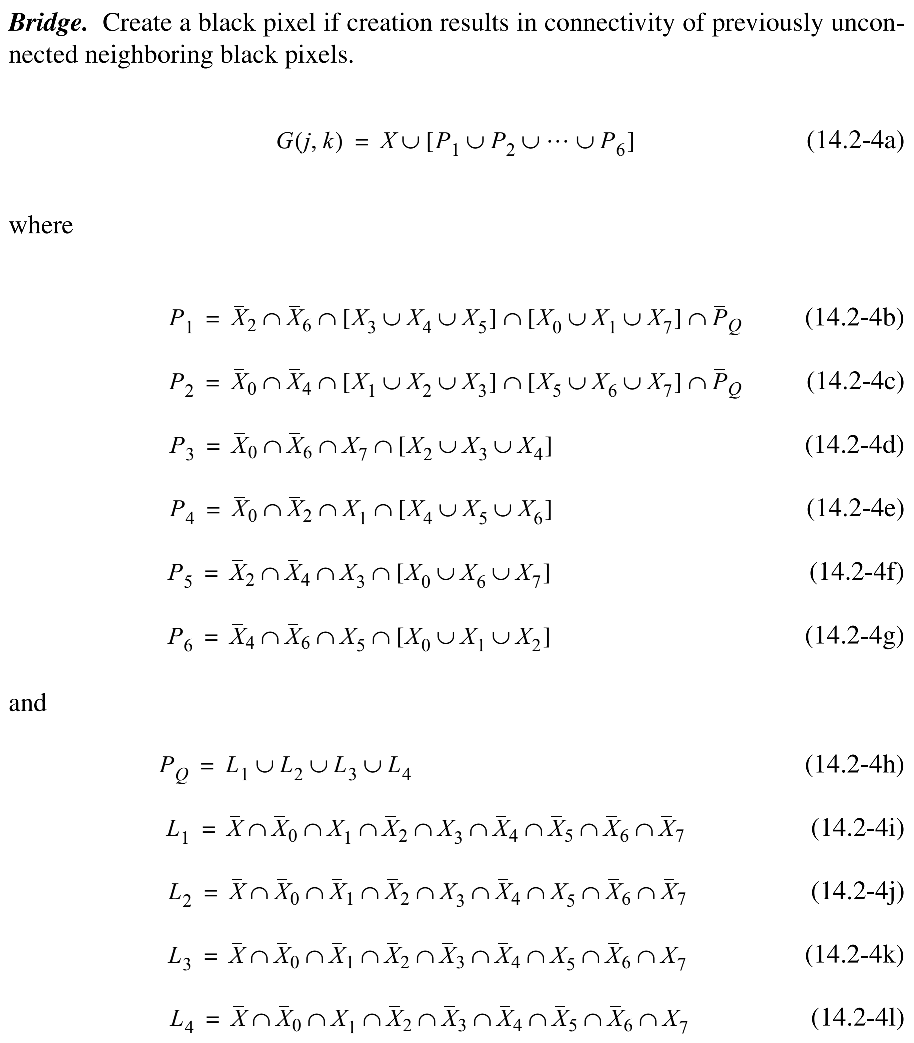
Another problem we met can be revealed from closer study of Fig. **??**: Notice that the spacing between each character is quite wide. If this is not the case, it may happen that two characters are joined during segmentation. One problem is that characters may overlap, another one is the quantization. Characters in the bitmap are not necessarily black-and-white, but diffuse. It is therefore important to find a good Quantization threshold: Is the threshold too high, characters maybe joined due to diffusion. Is the threshold too low, some characters might be split up, especially for small font sizes.

Finally, we also note that adjusting the weights is critically important for a good result. This issue is a keypoint for any Future Work on this project.

4 Conclusion

When we look at our results of the OCR application, we see that some improvements are needed if we want the program to work perfectly. The error rate should be reduced significantly, and this can be tried by different approaches. We can do this by increasing the test phase, using more variations as input for our program. Different, and more decisive features can be implemented, further decreasing the error rate. It is questionable though that very similar characters like 0 and D can ever be 100% correctly recognized. Finally, we could optimize the digitization process of text input even further by implementing a dictonary. This could be seen as a 'second opinion' on the decision made by our application. If OCR is rather doubtfull about the character, we can lookup the possible combinations in the dictionary and pick the most plausible one.

However, we can conclude that OCR is a very possible application (as has been proved by commercial applications): letters and numbers can be recognized based on their characteristics and in very reasonable time. This observation has lead to a series of applications found nowadays in our daily life. With the increase of computation power and portability of our devices, we can only predict these applications to become even more popular.



Bridge. Create a black pixel if creation results in connectivity of previously unconnected neighboring black pixels.

$$G(j,k) = X \cup [P_1 \cup P_2 \cup \cdots \cup P_6] \qquad (14.2\text{-}4a)$$

where

$$P_1 = \bar{X}_2 \cap \bar{X}_6 \cap [X_3 \cup X_4 \cup X_5] \cap [X_0 \cup X_1 \cup X_7] \cap \bar{P}_Q \qquad (14.2\text{-}4b)$$

$$P_2 = \bar{X}_0 \cap \bar{X}_4 \cap [X_1 \cup X_2 \cup X_3] \cap [X_5 \cup X_6 \cup X_7] \cap \bar{P}_Q \qquad (14.2\text{-}4c)$$

$$P_3 = \bar{X}_0 \cap \bar{X}_6 \cap X_7 \cap [X_2 \cup X_3 \cup X_4] \qquad (14.2\text{-}4d)$$

$$P_4 = \bar{X}_0 \cap \bar{X}_2 \cap X_1 \cap [X_4 \cup X_5 \cup X_6] \qquad (14.2\text{-}4e)$$

$$P_5 = \bar{X}_2 \cap \bar{X}_4 \cap X_3 \cap [X_0 \cup X_6 \cup X_7] \qquad (14.2\text{-}4f)$$

$$P_6 = \bar{X}_4 \cap \bar{X}_6 \cap X_5 \cap [X_0 \cup X_1 \cup X_2] \qquad (14.2\text{-}4g)$$

and

$$P_Q = L_1 \cup L_2 \cup L_3 \cup L_4 \qquad (14.2\text{-}4h)$$

$$L_1 = \bar{X} \cap \bar{X}_0 \cap X_1 \cap \bar{X}_2 \cap X_3 \cap \bar{X}_4 \cap \bar{X}_5 \cap \bar{X}_6 \cap \bar{X}_7 \qquad (14.2\text{-}4i)$$

$$L_2 = \bar{X} \cap \bar{X}_0 \cap \bar{X}_1 \cap \bar{X}_2 \cap X_3 \cap \bar{X}_4 \cap X_5 \cap \bar{X}_6 \cap \bar{X}_7 \qquad (14.2\text{-}4j)$$

$$L_3 = \bar{X} \cap \bar{X}_0 \cap \bar{X}_1 \cap \bar{X}_2 \cap \bar{X}_3 \cap \bar{X}_4 \cap X_5 \cap \bar{X}_6 \cap X_7 \qquad (14.2\text{-}4k)$$

$$L_4 = \bar{X} \cap \bar{X}_0 \cap X_1 \cap \bar{X}_2 \cap \bar{X}_3 \cap \bar{X}_4 \cap \bar{X}_5 \cap \bar{X}_6 \cap X_7 \qquad (14.2\text{-}4l)$$

Figure 10: Conditions for Bridging a Pixel, taken from [?], p. 406

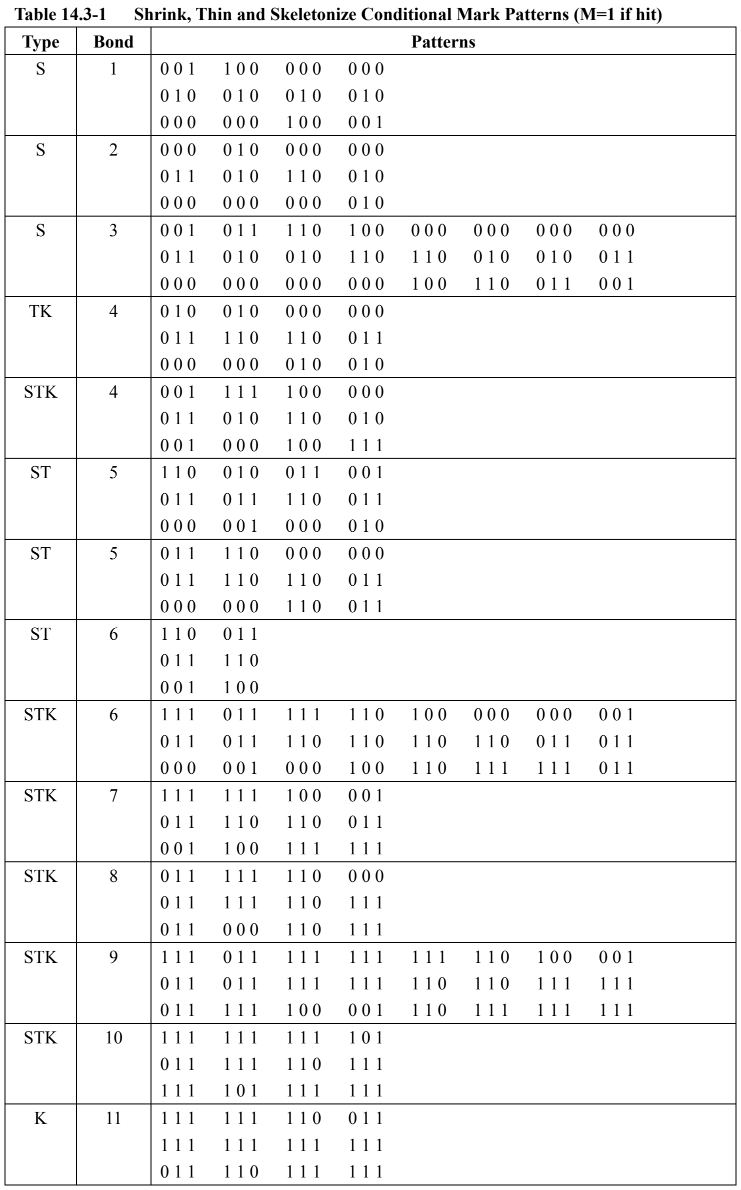


Table 14.3-1 Shrink, Thin and Skeletonize Conditional Mark Patterns (M=1 if hit)

Type	Bond	Patterns							
S	1	0 0 1	1 0 0	0 0 0	0 0 0				
		0 1 0	0 1 0	0 1 0	0 1 0				
		0 0 0	0 0 0	1 0 0	0 0 1				
S	2	0 0 0	0 1 0	0 0 0	0 0 0				
		0 1 1	0 1 0	1 1 0	0 1 0				
		0 0 0	0 0 0	0 0 0	0 1 0				
S	3	0 0 1	0 1 1	1 1 0	1 0 0	0 0 0	0 0 0	0 0 0	0 0 0
		0 1 1	0 1 0	0 1 0	1 1 0	1 1 0	0 1 0	0 1 0	0 1 1
		0 0 0	0 0 0	0 0 0	0 0 0	1 0 0	1 1 0	0 1 1	0 0 1
TK	4	0 1 0	0 1 0	0 0 0	0 0 0				
		0 1 1	1 1 0	1 1 0	0 1 1				
		0 0 0	0 0 0	0 1 0	0 1 0				
STK	4	0 0 1	1 1 1	1 0 0	0 0 0				
		0 1 1	0 1 0	1 1 0	0 1 0				
		0 0 1	0 0 0	1 0 0	1 1 1				
ST	5	1 1 0	0 1 0	0 1 1	0 0 1				
		0 1 1	0 1 1	1 1 0	0 1 1				
		0 0 0	0 0 1	0 0 0	0 1 0				
ST	5	0 1 1	1 1 0	0 0 0	0 0 0				
		0 1 1	1 1 0	1 1 0	0 1 1				
		0 0 0	0 0 0	1 1 0	0 1 1				
ST	6	1 1 0	0 1 1						
		0 1 1	1 1 0						
		0 0 1	1 0 0						
STK	6	1 1 1	0 1 1	1 1 1	1 1 0	1 0 0	0 0 0	0 0 0	0 0 1
		0 1 1	0 1 1	1 1 0	1 1 0	1 1 0	1 1 0	0 1 1	0 1 1
		0 0 0	0 0 1	0 0 0	1 0 0	1 1 0	1 1 1	1 1 1	0 1 1
STK	7	1 1 1	1 1 1	1 0 0	0 0 1				
		0 1 1	1 1 0	1 1 0	0 1 1				
		0 0 1	1 0 0	1 1 1	1 1 1				
STK	8	0 1 1	1 1 1	1 1 0	0 0 0				
		0 1 1	1 1 1	1 1 0	1 1 1				
		0 1 1	0 0 0	1 1 0	1 1 1				
STK	9	1 1 1	0 1 1	1 1 1	1 1 1	1 1 1	1 1 0	1 0 0	0 0 1
		0 1 1	0 1 1	1 1 1	1 1 1	1 1 0	1 1 0	1 1 1	1 1 1
		0 1 1	1 1 1	1 0 0	0 0 1	1 1 0	1 1 1	1 1 1	1 1 1
STK	10	1 1 1	1 1 1	1 1 1	1 0 1				
		0 1 1	1 1 1	1 1 0	1 1 1				
		1 1 1	1 0 1	1 1 1	1 1 1				
K	11	1 1 1	1 1 1	1 1 0	0 1 1				
		1 1 1	1 1 1	1 1 1	1 1 1				
		0 1 1	1 1 0	1 1 1	1 1 1				

Figure 11: Unconditional and Conditional Mark Patterns (part 1), taken from [?], p. 413

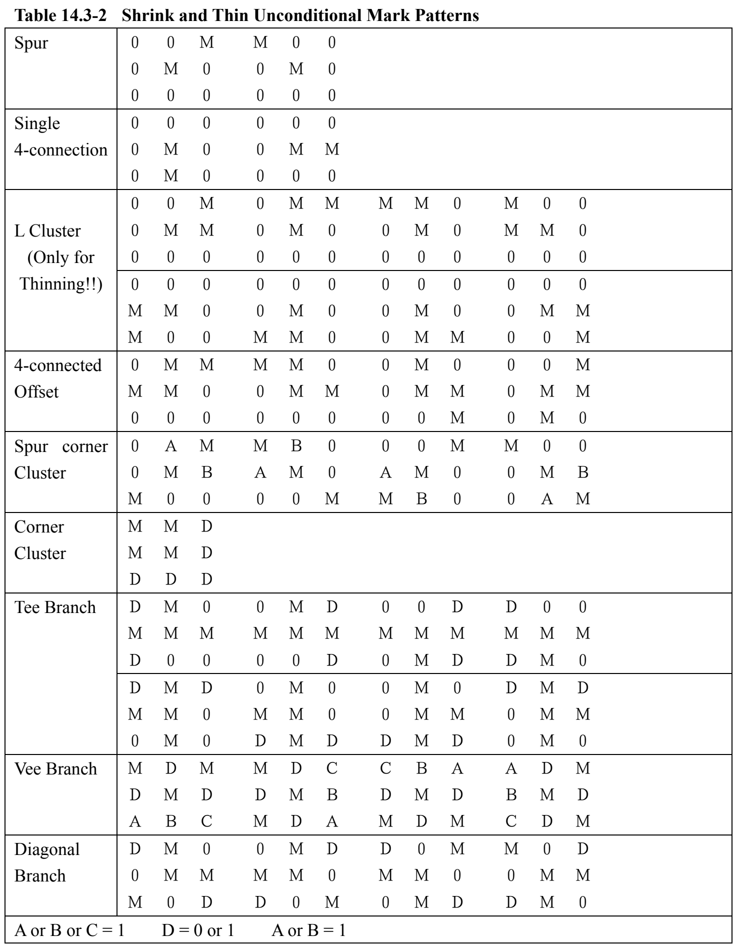


Table 14.3-2 Shrink and Thin Unconditional Mark Patterns

Spur	0	0	M	M	0	0						
	0	M	0	0	M	0						
	0	0	0	0	0	0						
Single 4-connection	0	0	0	0	0	0						
	0	M	0	0	M	M						
	0	M	0	0	0	0						
L Cluster (Only for Thinning!!)	0	0	M	0	M	M	M	M	0	M	0	0
	0	M	M	0	M	0	0	M	0	M	M	0
	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0
	M	M	0	0	M	0	0	M	0	0	M	M
	M	0	0	M	M	0	0	M	M	0	0	M
4-connected Offset	0	M	M	M	M	0	0	M	0	0	0	M
	M	M	0	0	M	M	0	M	M	0	M	M
	0	0	0	0	0	0	0	0	M	0	M	0
Spur corner Cluster	0	A	M	M	B	0	0	0	M	M	0	0
	0	M	B	A	M	0	A	M	0	0	M	B
	M	0	0	0	0	M	M	B	0	0	A	M
Corner Cluster	M	M	D									
	M	M	D									
	D	D	D									
Tee Branch	D	M	0	0	M	D	0	0	D	D	0	0
	M	M	M	M	M	M	M	M	M	M	M	M
	D	0	0	0	0	D	0	M	D	D	M	0
	D	M	D	0	M	0	0	M	0	D	M	D
	M	M	0	M	M	0	0	M	M	0	M	M
	0	M	0	D	M	D	D	M	D	0	M	0
Vee Branch	M	D	M	M	D	C	C	B	A	A	D	M
	D	M	D	D	M	B	D	M	D	B	M	D
	A	B	C	M	D	A	M	D	M	C	D	M
Diagonal Branch	D	M	0	0	M	D	D	0	M	M	0	D
	0	M	M	M	M	0	M	M	0	0	M	M
	M	0	D	D	0	M	0	M	D	D	M	0

A or B or C = 1 D = 0 or 1 A or B = 1

Figure 12: Unconditional and Conditional Mark Patterns (part 2), taken from [?], p. 414

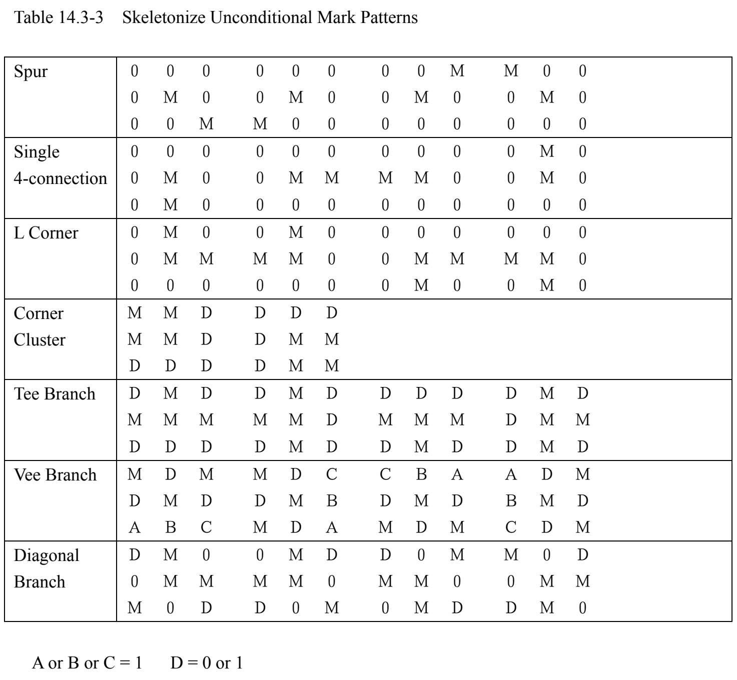


Table 14.3-3 Skeletonize Unconditional Mark Patterns

Spur	0 0 0	0 0 0	0 0 M	M 0 0
	0 M 0	0 M 0	0 M 0	0 M 0
	0 0 M	M 0 0	0 0 0	0 0 0
Single	0 0 0	0 0 0	0 0 0	0 M 0
4-connection	0 M 0	0 M M	M M 0	0 M 0
	0 M 0	0 0 0	0 0 0	0 0 0
L Corner	0 M 0	0 M 0	0 0 0	0 0 0
	0 M M	M M 0	0 M M	M M 0
	0 0 0	0 0 0	0 M 0	0 M 0
Corner	M M D	D D D		
Cluster	M M D	D M M		
	D D D	D M M		
Tee Branch	D M D	D M D	D D D	D M D
	M M M	M M D	M M M	D M M
	D D D	D M D	D M D	D M D
Vee Branch	M D M	M D C	C B A	A D M
	D M D	D M B	D M D	B M D
	A B C	M D A	M D M	C D M
Diagonal	D M 0	0 M D	D 0 M	M 0 D
Branch	0 M M	M M 0	M M 0	0 M M
	M 0 D	D 0 M	0 M D	D M 0

A or B or C = 1 D = 0 or 1

Figure 13: Unconditional and Conditional Mark Patterns (part 3), taken from [?], p. 415

5 Division of Labor

Sebastian Agethen	Program Framework, Decision Clustering, Features: Lakes, Euler number, Lines and Circles, Shape Coefficient
Lin Sheng-Ching	Extraction of characters, skeletonization of characters geometrical features (Sec. **??**), orientation
Jeroen Dhondt	Project Proposal, Bays, Presentation/Final Report: Introduction, Motivation, Conclusion