Text retrieval from early printed books

Text Retrieval from Early Printed Books

Simone Marinai
Dipartimento di Sistemi e Informatica
University of Firenze
Firenze, Italy
marinai@dsi.unifi.it

ABSTRACT

We describe a text indexing and retrieval technique that does not rely on word segmentation and is tolerant to errors in character segmentation. The method is designed to process early printed documents and we evaluate it on the well known Latin Gutenberg Bible.

The approach relies on two main components. First, character objects (in most cases corresponding to individual characters) are extracted from the document and clustered together, so as to assign a symbolic class to each indexed object. Second, a query word is compared against the indexed character objects with a Dynamic Time Warping (DTW) based approach. The peculiarity of the matching technique described in this paper is the incorporation of sub-symbolic information in the string matching process. In particular, we take into account the estimated widths of potential subwords that are computed by accumulating lengths of partial matches in the DTW array.

Categories and Subject Descriptors

H.3.1 [**Content Analysis and Indexing**]: Indexing methods; H.3.3 [**Information Search and Retrieval**]: Clustering; I.7 [**Document and Text Processing**]: Document Capture

1. INTRODUCTION

Early printed books are among the most difficult documents to be indexed by machine reading systems. These books share some features with manuscripts and therefore techniques proposed for handwriting recognition can be adopted also for these documents. At a first look early printed books look very similar to medieval manuscripts, since they contain illuminated letters (hand painted) and several ligatures and abbreviations that were standard in manuscript writing and have been slowly abandoned in the technological progress of printing. Ligatures are present also in later works such as the Trévoux dictionary of the XVII-th Century [1] and are still in use in a limited number of cases such as the 'fi' ligature that merges 'f' and 'i'.

The Gutenberg Latin Bible, produced in 1455, is traditionally regarded as the first printed book in Western Countries even if printing with movable type in Korea pre-dates Gutenberg by several Centuries. Besides illuminated letters and painted capital letters, the Gutenberg Bible has a regular layout and each page is printed on two columns each with 42 lines (for this feature the work is also called "the 42-line Latin Bible").

Due to the emblematic role of this work some copies kept in Libraries around the world have been digitized and made available in Internet. We can mention a copy from Keyko in Japan [18], the British Library edition, and the copy that can be accessed from the Göttingen University Library (Germany)[1]. In the experiments described in this paper we used the Göttingen version since it is easier to access and the images have a higher resolution. From a practical point of view, recognizing the Gutenberg Bible is probably not very useful, since it is one of the most studied documents of that period. However, since its digital versions are widely available, it can be used as a shared benchmark in order to compare different algorithms and methods and this is the main motivation for using these data in our experiments.

Despite the regular layout of the pages, the automatic reading of the Gutenberg Bible is a very difficult task for several reasons: First, the image resolution and the file format available in Internet are not optimal, since each page is stored as a JPG file with a size of 965x1390 pixels (the average character size is 10x15 pixels). Second, typical aging problems that concern historical documents occur also in this document: support aging, bleed trough, and see trough. The latter problem is particular critical since the 'verso' page is observed in the 'recto' one both for the textual and for the painted regions. Third, the cost of printing supports (vellum and parchment) was very high and therefore every possible strategy was applied to pack the text. Therefore, many abbreviations and ligatures are used and there is a very limited spacing among words. The latter feature influences the reading capabilities of humans as well: for people aware of Latin writing, but not scholars of medieval manuscripts, it is really difficult to read the text without the help of a transcription.

From the user point of view, the most suitable tool should be

[1] http://www.gutenbergdigital.de

AND '09 July 23-24, 2009, Barcelona, Spain

a specific OCR designed to read these documents. However, the large use of ligatures and abbreviations significantly limits the applicability of this approach since many symbols should be considered and a suitable dictionary listing all the possible forms for a given lemma is not easily available. Since Gutenberg Bible is a milestone in the Western culture, it is possible that those information could be available. However, our aim is not to deal only with the Gutenberg Bible, but to design tools that can process early printed books, that can adopt different ligatures and abbreviations. We therefore designed a text retrieval tool that deals with the text in a printed document in a different way, trying to identify occurrences of query words rather than recognizing the whole text.

This paper is organized as follows. In Section 2 we briefly summarize some recent work related with the proposed technique. Additional information on the peculiarities of Gutenberg Bible is summarized in Section 3. Details on the proposed retrieval method are described in Section 4 whereas some experimental results are described in Section 5. Conclusions and future work are discussed in Section 6.

2. PREVIOUS WORK

Text retrieval methods have been widely studied in the last few years (e.g. [9] [13] [14]). One common feature of most systems is the assumption that individual words can be reliably extracted from the text. This feature holds in modern printed books, but in general is not realistic in handwriting and in early printed documents. For instance, word shape coding is considered in [13] where the document content is indexed by annotating each word image with a shape code. The codes are extracted at the word level, rather than at the character level, so as to minimize errors due to wrong character segmentations. The codes are computed starting from a set of topological shape features that include character ascenders/descenders, character holes, and character water reservoirs.

In our previous work [14] we dealt with modern printed books performing a word indexing and retrieval from XVIII - XIX th Century books. The indexing was based on character codes similarly to the approach described in this paper. However, the indexing granularity was at the word level, whereas in this work we do not attempt to identify word boundaries. Character object coding has been used also in [19] where text similarity between documents is considered.

Concerning the word matching, when the text is represented with characters codes, one possibility relies on the use of filtering strategies (e.g. [14]). An alternative approach is based on dynamic programming techniques that have been used to identify the occurrences of a query word in a document. The latter techniques are related to approximate string matching (also known as string edit distance or Levenshtein distance) when symbolic codes are assigned to characters. When feature vectors are associated to characters, the algorithms are most likely classified in the family of Dynamic Time Warping (DTW) approaches.

To deal with duplicate document detection, Lopresti and Zhou modified the approximate string matching algorithm considering *split* and *merge* editing operations in addition to the standard *deletion*, *insertion*, and *substitution* [10]. The edit distance is considered also in [5] where confused characters in erroneous words are located and editing operations are applied to create a collection of erroneous error-grams. Subsequently, query terms and error-grams are used to perform query expansion. More recently, the edit distance has been used for table identification [11]. Dynamic Time Warping is an algorithm for measuring the similarity between two sequences that has been initially applied mostly in speech recognition systems. In the field of document image retrieval it has been used for on-line handwritten document indexing and retrieval [7] and for handwritten word image matching [2] [16]. One modified DTW that takes into account under and over segmentations of characters has been described in [12] for Arabic documents transcription. More recently, one DTW-based partial matching scheme has been proposed to allow the control of morphological variants of a word [15].

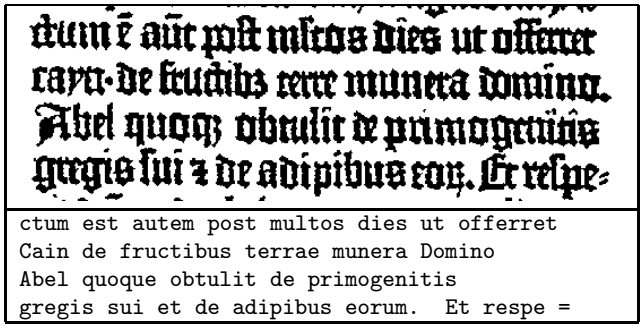


Figure 1: Portion of a page of Genesis 4:3,7. In the lower part we report the text transcription.

The Gutenberg Bible has been already used as a testbed for pre-processing algorithms designed to work on historical documents. For instance [17] describes an approach for pixel classification used for document image enhancement. The binarisation of historical documents is described also in [6]. Other papers addressed the retrieval of ornamental initials from historical documents [4] [8].

3. THE GUTENBERG BIBLE

To illustrate the difficulties of the automatic recognition of the Gutenberg Bible we report in Figure 1 one excerpt with the corresponding transcription. In the image, some ligatures and abbreviations can be noticed. For instance, the second word is the verb 'est' that is printed with a unique symbol. Overall, the Gutenberg Bible contains more than 75 types of ligatures (15 having two or more corresponding strings) [3]. Some examples of ligatures with the corresponding meaning are shown in Figure 2.

From the page layout point of view the Gutenberg Bible has a fixed organization with two columns per page each containing 42 lines. Similarly to manuscripts, the spacing between lines is reduced and ascenders and descenders are very close to neighboring lines. In addition, most capital letters are painted and some pages begin with painted initials.

However, from the automatic recognition point of view, the most important feature is the very limited spacing between

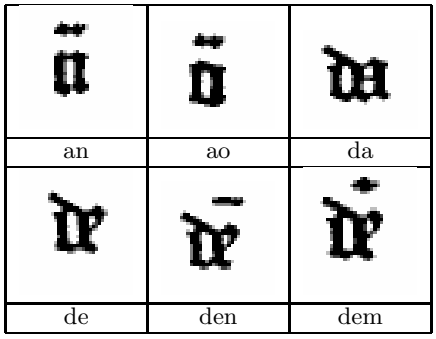


Figure 2: Examples of ligatures from the Gutenberg Bible.

contiguous characters in the same line. The method described in the next section performs the text indexing and retrieval without segmenting individual words.

4. THE TEXT RETRIEVAL METHOD

The proposed text retrieval method is based on the identification of Character Objects in the text and their subsequent clustering by means of Self Organizing Maps (SOM). The Character Objects (CO) [19] are connected components (or portions of connected components) that in general correspond to single characters. Once the SOM clustering is computed, each CO is labeled with the symbolic class corresponding to the nearest SOM neuron. In this way it is possible to speed-up the matching process with respect to a pure template matching working at the image level, since we can rely on approximate string matching. To improve the retrieval performance, we modified the approximate string matching algorithm so as to take into account the features of the SOM maps and the position of actual COs in the text lines. This matching algorithm is described in Section 4.3.

4.1 Pre-processing and layout analysis

With the pre-processing and layout analysis step we extract the columns, the text lines and the COs from each page with a top-down procedure based on projection profiles. Even if the quality of documents is not perfect, and the text lines are very close one to the other, we achieved good segmentation results by taking into account the information about the page structure (2 columns each with 42 lines).

The input images are in color, but the textual content is black and white since the colored parts have been added after printing in order to decorate the pages. We binarize a gray level image that is obtained by computing an average of the three color bands: $I = \frac{R+G+B}{3}$. Since we are interested in almost back pixels we remove many colored parts that should not be considered at all by our algorithm. The gray level image I is then binarized with a fixed threshold obtaining the black and white image BW.

Since the page structure is rather regular and the pages have been carefully aligned during the digitization, we implemented projection-profile based algorithms to identify columns, rows, and individual characters from each page. The text columns are extracted with the following steps:

1. Compute the vertical projection profile of the page: $V_j = \sum_{i=1}^{N} BW_{i,j}$ (N is the image height and $BW_{i,j}$ is the image value in position i, j);
2. Compute the "gradient" of the projection profile by considering a mobile window of 7 pixels: $G_j = \frac{\sum_{k=1}^{3}(V_{j+k}-V_{j-k})}{7}$;
3. Identify the columns positions by thresholding the G values (highest values correspond to text-background transitions, and vice-versa);
4. If two columns are not found, the step 3 is repeated by modifying the threshold used on G.

The previous procedure is used also for text lines segmentation, considering 42 as target value for the number of lines. Once the text lines are identified, we include also the pixels of the connected components that extend above or below the text line bounding box.

The last segmentation step is the character extraction that is based on the identification of connected components in each text line. Potential touching connected components are searched among the largest connected components. In this case we look for minima in the vertical projection profile computed from the pixels of the component to be split. Even if we carefully design and tune the segmentation algorithms it is not possible to obtain a perfect segmentation and under-over-segmentations can occur. For instance, in Figure 3 we show three different segmentations that are obtained from three instances of the word 'Adam'. The matching algorithm should therefore be tolerant to differences in the number of objects composing the words.

Figure 3: Different instances of the word 'adam'.

4.2 Character Clustering

Once the position of each CO is identified in the BW image we extract the corresponding character from the gray level image I and we scale it to fit an 8×10 grid. Each CO is therefore represented by an 80-th dimensional feature vector. These vectors are used to train an SOM with a size of 15×10 neurons. The trained SOM is then used to label each CO to be indexed with a pair of integers in the range $([0 \dots 14], [0 \dots 9])$.

Additional details of the SOM training and its use for CO labeling in the case of modern printed documents can be found in [14]. When dealing with early printed documents unsuitable maps can be obtained if the COs are not correctly extracted. In particular, if the last step of projection-based character segmentation is omitted, we can obtain many under- or over-segmented COs thus generating 'wrong' SOMs. For instance, Figure 4 shows two maps computed from an inaccurate CO segmentation.

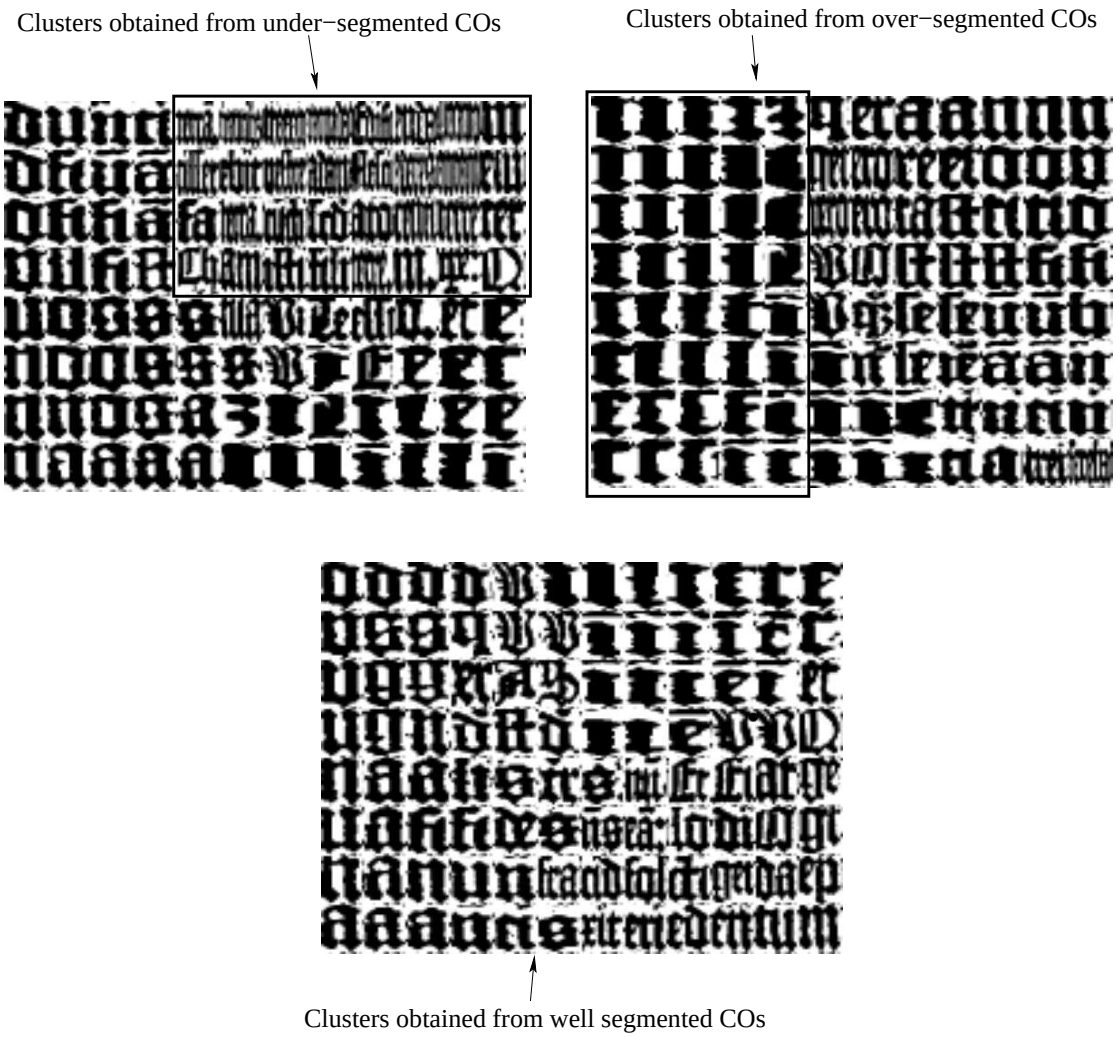


Figure 4: Examples of 12×8 SOM maps obtained with different segmentations of COs used for training.

For each CO_i extracted from the pages we index the following n-uple that only contains integer values and therefore can be quickly accessed with high compression rates:

$CO_i =< Pag(CO_i), Col(CO_i), Row(CO_i), BB_l(CO_i), BB_r(CO_i), S_x(CO_i), S_y(CO_i) >$

Pag, Col, and Row are the Page, Column and Row containing the CO. $BB_l(CO_i)$ and $BB_r(CO_i)$ are the left and right positions of the CO bounding box in its textline. $S_x(CO_i)$ and $S_y(CO_i)$ are the coordinates of the SOM neuron that is the closest to CO_i.

4.3 Matching Algorithm

The retrieval method that we implemented is based on a query by example approach. The user selects from the document collection one word image that is used as a query. The image is represented with its COs and the search for indexed words matching the query is performed as follows.

One solution to the approximate string matching problem is based on dynamic programming. This algorithm and some simple variants have been used in several contexts to compute a distance between two strings allowing three basic types of errors: *substitution*, *insertion*, and *deletion*.

Let us first summarize the basic algorithm that is used to compare a query string Q (having $|Q|$ symbols) with a text string T (having $|T|$ symbols). The basic data structure is a matrix $M_{0\ldots|Q|,0\ldots|T|}$ whose elements represent the minimum number of edit operations needed to match $Q_{1\ldots i}$ with $T_{1\ldots j}$. In other words $M_{i,j} = ed(Q_{1\ldots i}, T_{1\ldots j})$. The elements of the matrix are computed according to the following equations:

$$M_{0,0} = 0 \quad (1)$$

$$M_{i,j} = min(M_{i-1,j-1} + \sigma(Q_i, T_j), M_{i-1,j} + 1, M_{i,j-1} + 1) \quad (2)$$

In the basic formulation, $\sigma(Q_i, T_j) = 0$ if $Q_i = T_j$ and 1 otherwise. From Equation (2) it is clear that the value $M_{i,j}$ is computed taking into account only values in the previous

column or just above the current value. The matrix M can be computed starting from $(i, j) = (0, 0)$ evaluating one column after the other. Therefore, we only need to predefine the values of $M_{*,0}$ and $M_{0,*}$ to suitable values, for instance ∞, or with increasing values ($M_{i,0} = i$, $M_{0,j} = j$). When the whole matrix is computed, the last position provides the edit distance between the two strings: $M_{|Q|,|T|} = ed(Q, T)$. From the implementation point of view it is possible to use only two columns (the current and the previous one) that are updated during the computation, instead of the whole matrix.

This basic algorithm can be modified in several ways. We describe in the following only the changes that are considered in our approach.

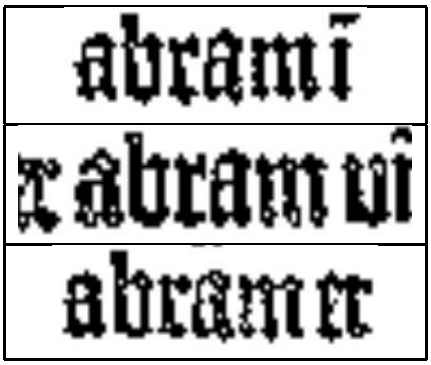


Figure 5: Three occurrences of the word 'Abram': the space with neighboring words is very narrow.

4.3.1 Text searching

In our application, and in general when the word segmentation is not reliable, a text searching approach is more appropriate with respect to string matching. As an example, we show in Figure 5 three words that are very close to adjacent words and it is very difficult to identify the word boundary. In text searching, we search the pattern Q in text T allowing an occurrence to begin at any position in T. This is obtained by setting $M_{0,j} = 0$ for each j. After computing the whole matrix, the values in the last row correspond to edit distances of potential occurrences of Q in T. An example is shown in Figure 6 where the pattern "adam" has one occurrence with one error (a substitution) in the text "ademad".

		a	d	e	m	a	d
	0	0	0	0	0	0	0
a	1	0	1	1	1	1	0
d	2	1	0	1	2	2	1
a	3	2	1	1	2	2	2
m	4	3	2	2	1	2	3

Figure 6: Matrix M for comparing searching the word "adam" in the text "ademad".

4.3.2 Weighting differences between COs

The previous formulation is typical of a symbolic domain, where objects are represented by tokens belonging to a finite number of classes. Typical examples are textual documents. In syntactic pattern recognition, we deal with numerical feature vectors rather than with strings. However, in image text retrieval it is possible to cluster the character images (see Section 4.2) and then consider a substitution when the query character and the test one belong to different clusters. In the opposite, the characters are assumed to coincide when they belong to the same cluster. In our case we use the SOM to cluster the character objects and therefore we can take into account the topological organization of the SOM map when comparing characters. In the dynamic programming formulation the edit distance needs to be slightly modified. First, we consider Q and T to be sequences of COs instead of sequences of symbols. Second, we weight the similarity between symbols according to the position in the SOM map of the corresponding clusters:

$$\sigma_S(Q_i, T_j) = \frac{\sqrt{(S_x(Q_i) - S_x(T_j))^2 + (S_y(Q_i) - S_y(T_j))^2}}{MaxSomDist} \quad (3)$$

where $MaxSomDist$ is the maximum distance between pairs of neurons in the SOM.

Taking into account this value of $\sigma(Q_i, T_j)$ it is now possible to recompute the matrix M with Equation (2). The elements of M are real values, but the overall interpretation is the same: lowest values in the last row of M correspond to best matching occurrences. It is also possible to consider variable costs for insertions and deletions and this yields to the general DTW framework that has been already used in some word retrieval approaches.

4.3.3 Considering the sub-word width

The DTW framework is very elegant, however one significant limitation is that we implicitly consider all the COs as having the same size. In other words, in this framework we have poor results when character segmentation errors occur. For instance, if one character is split into two sub-components, then the matching cost with a perfectly segmented word will be at most 2 (or more generally the cost of one deletion and one substitution). One possible solution to this problem relies on the use of *split* and *merge* editing operations that are more likely to occur in OCR [10]. In a recent paper [12] the DTW algorithm has been modified to take into account split and merged characters in handwritten Arabic text.

In our approach, we deal with this problem by taking into account the width of the query image and the expected width of the partial match in the searching text. The width of the query image can be computed by: $W(Q) = BB_r(Q_{|Q|}) - BB_l(Q_1)$ that is the difference between the rightmost point of the last character and the leftmost point of the first character in Q. Similarly, the width of a generic sub-query (composed by the first i characters in Q) can be computed by: $W(Q_i) = BB_r(Q_i) - BB_l(Q_1)$.

The computation of the width of a sequence of characters

in T is more complex. When dealing with the character T_j in matrix M the rightmost point of the sub-string in T is $BB_r(T_j)$. However, the leftmost point depends on the path followed in the computation of M.

Let us first remind the rationale at the basis of the standard edit distance computation: when computing each element $M_{i,j}$ we assume inductively that all the edit distances between shorter strings have already been computed, the best matches are summarized in M and we then try to convert $Q_{1\ldots i}$ into $T_{1\ldots j}$.

We can therefore associate to each element $M_{i,j}$ a value that corresponds to the leftmost point of the sub-word $T_{1\ldots j}$. These values are stored in a matrix $L_{0\ldots|Q|,0\ldots|T|}$ that is updated and read in parallel with M (therefore it is possible to use also in this case two columns instead of the whole matrix).

$M_{i,j}$ and $L_{i,j}$ are updated according to the following steps. First, we compute $SubCost$, $DelCost$, and $InsCost$ as follows:

$$\sigma_W(Q_i, T_j) = \frac{(BB_r(T_j) - L(i-1, j-1)) - W(Q_i)}{AvgW} \quad (4)$$

$$SubCost = \sigma_S(Q_i, T_j) + \sigma_W(Q_i, T_j) + M(i-1, j-1) \quad (5)$$

$$\sigma_W(Q_i, T_j) = \frac{(BB_r(T_j) - L(i-1, j)) - W(Q_i)}{AvgW} \quad (6)$$

$$DelCost = \sigma_S(Q_i, T_j) + \sigma_W(Q_i, T_j) + M(i-1, j) \quad (7)$$

$$\sigma_W(Q_i, T_j) = \frac{(BB_r(T_j) - L(i, j-1)) - W(Q_i)}{AvgW} \quad (8)$$

$$InsCost = \sigma_S(Q_i, T_j) + \sigma_W(Q_i, T_j) + M(i, j-1) \quad (9)$$

In general $AvgW$ is set to the average width of characters in the collection.

We then compute the minimum cost among $SubCost$, $DelCost$, and $InsCost$ and update $M_{i,j}$ and $L_{i,j}$ as follows (for instance, if condition (10) holds we update $M_{i,j}$ and $L_{i,j}$ with (11) and (12) respectively):

$$if(SubCost == min(SubCost, DelCost, InsCost)) \quad (10)$$

$$M_{i,j} = SubCost \quad (11)$$

$$L_{i,j} = L_{i-1,j-1} \quad (12)$$

$$if(DelCost == min(SubCost, DelCost, InsCost)) \quad (13)$$

$$M_{i,j} = DelCost \quad (14)$$

$$L_{i,j} = L_{i-1,j} \quad (15)$$

$$if(InsCost == min(SubCost, DelCost, InsCost)) \quad (16)$$

$$M_{i,j} = InsCost \quad (17)$$

$$L_{i,j} = L_{i,j-1} \quad (18)$$

To begin the algorithm we need to set suitable values for $L_{*,0}$ and $L_{0,*}$: $L(i,0) = BB_l(T_1)$ for $(i = 1\ldots|Q|)$, $L(0,j) = BB_l(T_j)$ for $(j = 1\ldots|T|)$

When the whole text T is processed, the values in the last row of the matrix M correspond to errors of possible occurrences of Q in T. The lower values identify potential matches whose starting coordinate is indicated by the corresponding value in L.

Depending on the application, it is possible to reset M and L when a new text line begins or to continue with the current values in the next row so as to allow the matching of hyphenated words.

To summarize, in the proposed method two factors contribute to the weight computation at each step: the distance between cluster centers in the SOM map and a comparison of the matching word lengths. In the next Section we discuss the experiments that we performed to evaluate the performance of the method.

5. EXPERIMENTAL RESULTS

The numerical validation of text retrieval in the Gutenberg Bible is a difficult task since an accurate ground truth of the text is not available for our experiments. The content is obviously well known since Gutenberg used the so called *Biblia Vulgata* as source. However, the textual version of the *Biblia Vulgata* that we used contains a transcription of the text that does not take into consideration ligatures and abbreviations. In the current system implementation the is searched with a query by example mechanism and it is not possible to find occurrences of a query word that are printed with a different ligature. For instance, one query is performed with the word "*dominus*", that is printed with various abbreviations. In this case we made the query with one of the most common abbreviations ("dn̄s"). To evaluate the retrieval performance we therefore needed to visually check all the images in the answer set in order to compute the achieved Precision. However, since we do not know the number of occurrences of a given abbreviation we cannot measure the Recall level obtained.

The experimental framework has been set as follows. We first indexed 20 pages from the Genesis. Subsequently, we identified some common words and used ten query words and sorted the retrieved words according to three methods (Table 1).

The proposed method (denoted as $M1$) takes into account the SOM clustering and the estimation of word width by combining σ_S and σ_W (in Equations (5),(7),(9)). In $M2$ we set $\sigma_S(Q_i, T_j) = 1$ if and only if Q_i and T_j belong to the same SOM cluster. In so doing the SOM structure is not considered (Section 4.3.2). On the opposite in $M3$ we considered only the SOM contribution by setting always $\sigma_W = 0$

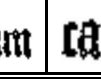
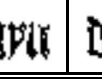
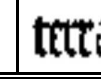
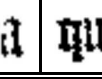
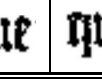
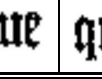
		abram	cain 	deus 	dominus 	terra 	que 	que 	quia 	quo 	quod 	Avg
M1	5	5	3	5	5	3	2	3	1	4	2	**3.3**
	10	8	3	7	10	3	2	7	2	7	3	**5.2**
	20	11	3	11	19	5	6	14	3	12	3	**8.7**
M2	5	4	0	2	5	0	2	2	1	3	2	2.1
	10	5	0	4	5	0	3	3	1	4	3	2.8
	20	5	1	8	10	1	4	4	1	6	3	4.3
M3	5	5	0	3	5	0	3	2	2	4	3	2.7
	10	8	0	6	10	1	4	4	2	6	4	4.5
	20	11	1	6	18	2	7	11	4	8	4	7.2

Table 1: Comparison of the three methods.

and excluding the width contribution (Section 4.3.3).

For each method, and for each query word, we computed the number of positive results in the top 5, 10, and 20 positions. In the majority of cases the $M1$ method recognizes more correct words in the top-x ranks and also the average values (in the last column) confirm the good performance of the proposed method. We computed also the standard string edit distance, but the values are not reported in the table since it performed significantly poorly than the others.

Even if only ten query words are considered, we tried to take into account several types of situations. In particular, *dominus* and *quod* are words having a significant abbreviation and the latter word is composed by a single CO. It is interesting to notice that in the case of *quod*, $M3$ performs better than the other methods since the information on word length has no influence.

The results obtained with the query *Deus* are very interesting. We show the query word and some positive answers (obtained with $M1$) in Figure 7. Each CO is annotated with some indexing information. For instance, the first CO has the following values: 12 is the number of CO in the text line; 96 is the leftmost point of CO ($BB_l(CO)$); 9 is the CO width; 2/1 denotes the SOM neuron ($S_x(CO) = 2$, $S_y(CO) = 1$). By comparing the SOM neurons of corresponding characters in different words we can notice that the same characters are clustered in neighboring SOM neurons, and this allows us to have lower values in σ_S with correct matches of characters. It is important to notice also that the query word is over-segmented since it is composed by five COs instead of the expected four characters. However, the proposed algorithm is able to identify also words with four COs and also one word that is under-segmented (the last one).

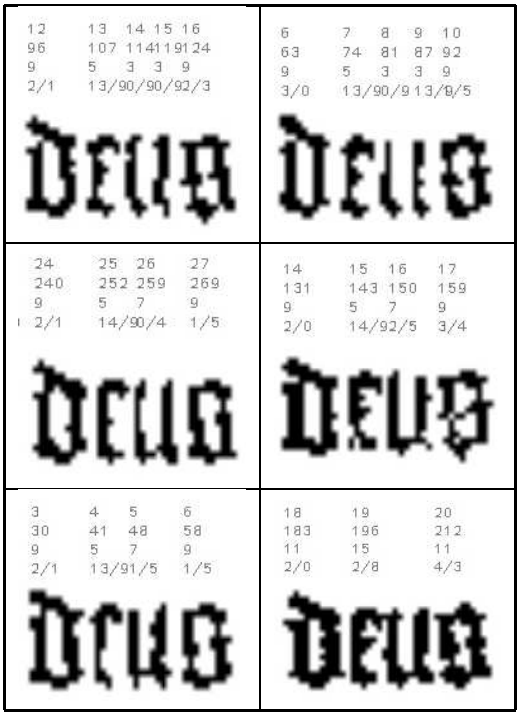


Figure 7: Query word for 'Deus' (top left) and five positive answers. Each character is annotated on top with its position, size, and cluster neurons.

6. CONCLUSIONS

In this paper we proposed a text retrieval method designed to deal with early printed books. This class of documents is represented by the Latin Gutenberg Bible that we used as a testbed since it is difficult to segment both at character and at word level. The proposed method is designed to retrieve words without performing word segmentation and without assuming a perfect character segmentation. The approach is based on a SOM-based clustering of indexed characters and on a modified Dynamic Time Warping algorithm that takes into account both the clusters similarity and the estimated widths of alternative sub-words. We are currently performing more extensive experiments in order to assess with more confidence the promising results described in the paper.

7. REFERENCES

[1] A. Belaïd, I. Turcan, J.-M. Pierrel, Y. Belaïd, Y. Rangoni, and H. Hadjamar. Automatic indexing and reformulation of ancient dictionaries. In *International Workshop on Document Image Analysis for Libraries*, pages 342–354. IEEE Computer Society,

2004.

[2] H. Cao, A. Bhardwaj, and V. Govindaraju. A probabilistic method for keyword retrieval in handwritten document images. *Pattern Recognition*, February 2009.

[3] F. Coulmans. *The Blackwell encyclopedia of writing systems*. Blackwell Publishing, 1999.

[4] M. Delalandre, J.-M. Ogier, and J. Lladós. A fast CBIR system of old ornamental letter. In *Int'l Workshop on Graphics Recognition*, pages 135–144, 2007.

[5] Y. Fataicha, M. Cheriet, J. Y. Nie, and C. Y. Suen. Retrieving poorly degraded ocr documents. *International Journal on Document Analysis and Recognition*, 8(1), 2006.

[6] M. R. Gupta, N. P. Jacobson, and E. K. Garcia. Ocr binarization and image pre-processing for searching historical documents. *Pattern Recognition*, 40(2):389–397, 2007.

[7] A. K. Jain and A. M. Namboodiri. Indexing and retrieval of on-line handwritten documents. In *Int'l Conference on Document Analysis and Recognition*, pages 655–, 2003.

[8] A. Karray, J.-M. Ogier, S. Kanoun, and M. A. Alimi. An ancient graphic documents indexing method based on spatial similarity. In *Int'l Workshop on Graphics Recognition*, pages 126–134, 2007.

[9] T. Konidaris, B. Gatos, K. Ntzios, I. Pratikakis, S. Theodoridis, and S. J. Perantonis. Keyword-guided word spotting in historical printed documents using synthetic data and user feedback. *International Journal on Document Analysis and Recognition*, 9(2-4):167–177, 2007.

[10] D. P. Lopresti. String techniques for detecting duplicates in document databases. *International Journal on Document Analysis and Recognition*, 2(4):186–199, 2000.

[11] D. P. Lopresti. Optical character recognition errors and their effects on natural language processing. In *Workshop on analytics for noisy unstructured text data*, pages 9–16, 2008.

[12] L. M. Lorigo and V. Govindaraju. Transcript mapping for handwritten Arabic documents. In *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, volume 6500, Jan. 2007.

[13] S. Lu, L. Li, and C. L. Tan. Document image retrieval through word shape coding. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(11):1913–1918, Nov. 2008.

[14] S. Marinai, E. Marino, and G. Soda. Font adaptive word indexing of modern printed documents. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(8):1187–1199, 2006.

[15] M. Meshesha and C. V. Jawahar. Matching word images for content-based retrieval from printed document images. *International Journal on Document Analysis and Recognition*, 11(1):29–38, 2008.

[16] T. M. Rath and R. Manmatha. Word spotting for historical documents. *International Journal on Document Analysis and Recognition*, 9(2-4):139–152, 2007.

[17] E. Smigiel, A. Belaïd, and H. Hamza. Self-organizing maps and ancient documents. In *Document Analysis Systems*, pages 125–134, 2004.

[18] T. Takamiya. How to make good use of digital contents: The Gutenberg bible and the HUMI project. In *Kyoto International Conference on Digital Libraries*, pages 110–112, 2000.

[19] C. L. Tan, W. Huang, Z. Yu, and Y. Xu. Imaged document text retrieval without OCR. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(6):838–844, June 2002.