10-9 MVA2015 IAPR International Conference on Machine Vision Applications, May 18-22, 2015, Tokyo, JAPAN

Document Image Dataset Indexing and Compression Using Connected Components Clustering

Houssem Chatbri* and Keisuke Kameyama†
*Department of Computer Science, Graduate School of Systems and Information Engineering
†Faculty of Engineering, Information and Systems
*†University of Tsukuba, Japan
*†{chatbri,kame}@adapt.cs.tsukuba.ac.jp

Abstract

We present a method for document image dataset indexing and compression by clustering of connected components. Our method extracts connected components from each dataset image and performs component clustering to make a hash table that is a compressed indexing of the dataset. Clustering is based on component similarity which is estimated by comparing shape features extracted from the components. Then, the hash table is saved in a text file, and the text file is further compressed using any available compression methodology. Component encoding in the hash table is storage efficient and done using components' contour points and a reduced number of interior points that are sufficient for component reconstruction. We evaluate our method's performances in indexing and compression using four document image datasets. Experimental results show that indexing significantly improves efficiency when used in document image retrieval. In addition, comparative evaluation with two compression standards, namely the ZIP and XZ formats, show competitive performances. Our compression rates are below 20% and the compression errors are very low being at the order of $10^{-6}\%$ *per image.*

1 Introduction

Document image datasets are a widespread medium of storing information. Nowadays, such datasets are becoming more and more large scale due to the availability of large storage media [8]. The research on document image analysis is very active and has led to numerous interesting applications [9].

Document indexing has been used in applications such as document retrieval for the sake of efficiency [3]. Indexing methods produce a representation of the data that is optimized for online querying. In addition, indexing methods have been used for dataset compression by exploiting data redundancy [7].

Approaches for document image compression using redundant information have been proposed. Haffner et al. presented a method for high resolution color document image compression by separating the image into text, pictures, and background [4][5]. Then, specific compression is applied to each category. For text compression, they use character pattern matching and substitution. Shiah and Yen presented a method for Chinese document image compression [11]. First, image segmentation is done using a priori knowledge about the documents. Then, Chinese characters are extracted using specific techniques of stroke merging.

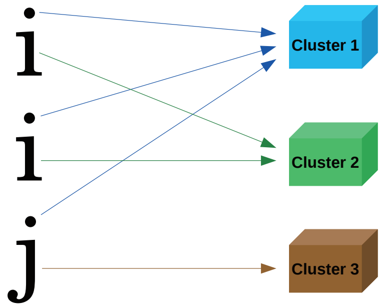


Figure 1. Similarity-based component clustering.

Compression is done using specific feature extraction and matching. Imura and Tanaka presented a similar method and evaluated it using English and Japanese documents [6]. They obtained language-dependent results. In both methods [11, 6], the compression error is not evaluated with objective metrics.

In this work, we present a method for document image dataset compression and indexing using redundant information in document images, as a part of our ongoing research on content-based document retrieval [1, 2]; We are designing a system for document retrieval that allows users to introduce handwritten queries. Then, the system retrieves the document images where the query is spotted. In order to reach online performances, dataset indexing should be implemented.

The proposed method is designed for document images where connected components high redundancy is a fair assumption. Our method exploits redundancy by performing clustering of similar connected components extracted from document images (Fig. 1). Comparing to previous techniques, our method stands out with the following aspects:

- Our algorithm is based on similarity estimation between connected components instead of character pattern images (Sec. 2.1), which makes it language-independent and more general.
- We introduce an optimized component encoding mechanism that uses some of the components' points and not all of them (Sec. 2.2).
- We save the compressed indexing as a text file that is further compressed, which enhances compression performances (Sec. 2.3).

We evaluated the proposed algorithm in indexing and compression. Experimental results demonstrate the usefulness of our algorithm as an indexing process for document retrieval (Sec. 3.2), and competitive performances comparing with two compression standards, namely the ZIP and XZ formats (Sec. 3.1).

2 The Proposed Approach

The proposed algorithm takes as input a document image dataset, and produces a compressed file using sequential clustering of connected components and text file compression. The algorithm proceeds as follows: The document image dataset contains M images. For each image I_i, the connected components, $\{C_j\}_{j=1}^{N_i}$, are extracted, where N_i refers to the number of components in I_i. Then, a discrete function $f(C_j)$ returns the cluster index corresponding to C_j if it has been already registered in hash table $Table$, or -1 otherwise. Consequently, C_j is registered in $Cluster_k$, or a new cluster $Cluster_{k_0}$ is created for C_j. This processing populates $Table$ with clusters of connected components. Then, $Table$ is saved in a text file $TxtFile$. Finally, the output $CompressedFile$ is produced by compressing $TxtFile$ using any text compressing algorithm.

In the following, we explain the mechanism for component similarity estimation (Sec. 2.1), component encoding (Sec. 2.2), and hash table compression (Sec. 2.3).

2.1 Component similarity estimation

Similarity between components is estimated using shape features extracted from connected components as done in [1][2]: For a component C_j, a feature vector $\overrightarrow{V_j}$ is extracted by calculating the distribution of pixels in polar coordinate where the origin is the component's centroid. The similarity between two components C_a and C_b is equivalent to the Histogram Intersection between their corresponding vectors, which is calculated as follows:

$$S(C_a, C_b) = \sum_{r=0}^{R-1} \sum_{\theta=0}^{\Theta-1} min(V_{r,\theta}^a, V_{r,\theta}^b) \tag{1}$$

where R and Θ refer to the radial and angular number of sections. Two components C_a and C_b are considered similar if they satisfy $S(C_a, C_b) > \delta$, where $\delta \in [0, 1]$ is a similarity threshold.

Using this feature extraction and matching mechanism, the function $f(C_j)$ is implemented as follows:

$$f(C_j) = \begin{cases} k, & \textbf{if} \quad \exists C_k[S(C_j, C_k) > \delta] \\ -1, & \textbf{otherwise} \end{cases} \tag{2}$$

where C_k refers to the cluster center of $Cluster_k$.

2.2 Component encoding

For the sake of optimal compression, the number of points in a component is reduced before saving it in the text file $TxtFile$. The component encoding algorithm extracts the necessary points to reconstruct a component. For a component C_j, the contour points and several non-contour, or *interior points*, are sufficient to reconstruct the component by connected component analysis. Therefore, only those points are needed to be saved. Fig. 2 shows examples of original components and their corresponding reconstruction points.

Algorithm 1 shows the component encoding steps: $List_j^R$ refers to the list of points needed for component reconstruction. First, the contour CP and an *interior point* IP are extracted, and added to $List_j^R$. Then, the reconstructed component C_j^R is produced using $List_j^R$. L points $\{P_l\}_{l=1}^L$ which exists in C_j but not in C_j^R are detected. Then, one point from $\{P_l\}_{l=1}^L$, P_1, is added to $List_j^R$. The iterations of producing C_j^R are repeated until C_j^R and C_j match.

Algorithm 1 Component encoding

define $List_j^R$: List of Points in the jth component
$CP \leftarrow$ ContourPoints(C_j)
$IP \leftarrow$ InteriorPoint(C_j)
$List_j^R \leftarrow CP, IP$
while stop = false **do**
  $C_j^R \leftarrow$ ReconstructComponent($List_j^R$)
  $\{P_l\}_{l=1}^L \leftarrow$ DifferencePoints(C_j^R, C_j)
  if $\{P_l\}_{l=1}^L$ is empty **then**
    stop = true
  else
    $List_j^R \leftarrow P_1$
  end if
end for

2.3 Hash table compression

The hash table, $Table$, is saved in a text file that is used of image reconstruction. In the text file, a header contains information about the images' names and sizes, and the rest of the file contains information about clusters which are the location of the connected component (centroid and image index), *interior points* and contour points, and locations of similar connected components.

Afterwards, the text file is compressed using any available text compression mechanism to produce a compressed indexing of the document image dataset. The idea behind using a text file is to exploit the character redundancy inside a plain text which is a main feature of text compression algorithms. After compressing the text file, the result is a binary file that has a reduced size.

3 Experimental results

We evaluate the algorithm's performances in terms of compression and indexing. Throughout the experiments, we set the component descriptor dimensions to $R = 3$ and $\Theta = 12$, and the similarity threshold to $\delta = 0.99$. In the following, we call our method C3 as abbreviation to Connected Components Clustering.

3.1 Compression performances

3.1.1 Evaluation procedure

We used three printed binary document image datasets that have been collected as follows:

- Dataset 1: 356 document images taken from the book of abstract of the 2014 World Congress on Computational Intelligence. The images are compressed in PNG-ZIP format, their size is 2479 × 3508 and their resolution is 300 dpi.
- Dataset 2: 159 document images taken from the book "Memoirs of John R. Young Utah Pioneer 1847"[1]. The images are compressed in PNG-ZIP format, their size is 2489 × 3518 and their resolution is 300 dpi.
- Dataset 3: 1320 document images taken from the book "Soothill-Hodous: A Dictionary of Chinese Buddhist Terms"[2]. Images contain English and Chinese words. The images are compressed in TIFF-Group4 format, their size is 2479 × 3508 and their resolution is 300 dpi.

The evaluation procedure consists of calculating the size of the compressed file and the error rate. The *error rate* ξ quantifies the number of pixel differences between the reconstructed image and its corresponding original image over the dataset, and it is calculated as follows:

$$\xi = \frac{1}{M}\sum_{i=1}^{M}\frac{1}{H\times W}\sum_{x=0}^{H-1}\sum_{y=0}^{W-1}|I_i^R(x,y) - I_i(x,y)| \quad (3)$$

where M is the number of images in the dataset, I_i and I_i^R refer to the original and reconstructed images, and H and W are the height and width of I_i.

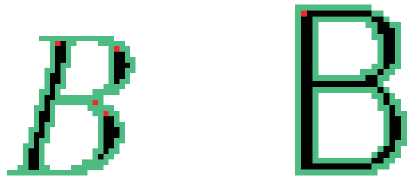
We compare our method, C3, combined with another standard compression method, namely ZIP or XZ [10], against using the standard compression method directly on the dataset.

3.1.2 Results and discussion

Compression performance

Table 1 shows the compression results: For all three datasets, C3 achieved higher compression comparing with using the ZIP or XZ compression directly. The best compression came with combining our method with XZ compression, in which case the compression rates (i.e. the size of the compressed file divided on the size of the original dataset) were respectively 6.4%, 2.2% and 16.6%. As for ZIP and XZ, their performances is explained by the fact that the images are already compressed. Therefore, no further significant compression can be achieved.

The performance of the proposed method is affected by the component redundancy in the document image dataset (Fig. 3). This can be seen particularly by the compression rates of Dataset 1 and Dataset 2, being 6.4%, 2.2% respectively. In case of these datasets, the number of redundant components is at the order of 10^3. For Dataset 3, the compression rate is 16.6%, as the number of redundant components is at the order of 10^2. The performances are also due to the optimized component encoding using a reduced number of points. Table 1 shows the *encoding ratio* which is equal to the number of encoded points divided by the initial number



(a) (b)

Figure 2. Reconstruction points in components (contour points are highlighted in green and interior points are highlighted in red): (a) In case of a nearly thin component, the number of encoded points is not significantly reduced. Here, *encoding ratio* = 73%. (b) In case of a thick component, the number of encoded points is significantly reduced. Here, *encoding ratio* = 48%.

of points. The *encoding ratio* is affected by the thickness of connected components (Fig. 2); The thicker a component is, the less number of points needed for reconstruction comparing with the initial number.

Information loss

The proposed compression method is lossy, and that is due to the tolerance of the descriptor used to estimate component similarity (Sec. 2.1). In our experiments, the error rate values were very low and we observe that it does not affect the document image readability. The component similarity threshold δ can be used as a parameter that controls the trade-off between the compression rate and the error rate.

3.2 Indexing performances

We implemented the proposed algorithm as an indexing mechanism for our ongoing document retrieval project [1, 2]. Then, we conducted retrieval experiments using Zanibbi and Yu's dataset [12]. This dataset contains 200 document images taken from a conference proceedings, and 240 printed and handwritten query images of mathematical expressions.

A core part of our document retrieval algorithm is comparing the connected components of the query against the connected components of the dataset images. In case of non-indexed implementation, all components of the document images are considered. While in case of an indexed implementation, only the components forming the clusters are considered.

We report the average duration of a component comparison process using a desktop computer equipped with a 3.40 GHz CPU. In case of a non-indexed implementation, the average duration to run a comparison was equal to 3,579 ms. While in case of indexing, the average duration was equal to 705 ms. The improvement in efficiency is then equal to 507%.

4 Conclusion and future work

In this work, we present a method for document image dataset indexing and compression by clustering of connected components. Our method extracts connected components from each dataset image and performs sequential clustering to make a hash table that is a compressed indexing the dataset. Then, the hash table is saved in a text file, and the text file is further

[1] Available at http://www.gutenberg.org/ebooks/46391
[2] Available at http://dev.ddbc.edu.tw/glossaries/

Table 1. Compression results using three datasets

Dataset	Original size	Compression method	Compression size	Error rate ξ	No. of components	No. of clusters	Encoding ratio
Dataset 1	107 MB	ZIP **C3-ZIP** XZ **C3-XZ**	102.7 MB **10.5 MB** 102.5 MB **6.9 MB**	1.5×10^{-6}	2 713 162	1 031	94.2 %
Dataset 2	50.2 MB	ZIP **C3-ZIP** XZ **C3-XZ**	34.4 MB **1.7 MB** 34.2 MB **1.1 MB**	0.1×10^{-6}	414 854	239	75.7 %
Dataset 3	44 MB	ZIP **C3-ZIP** XZ **C3-XZ**	37.6 MB **13.3 MB** 26.9 MB **7.3 MB**	0.3×10^{-6}	1 835 719	10 792	52.4 %

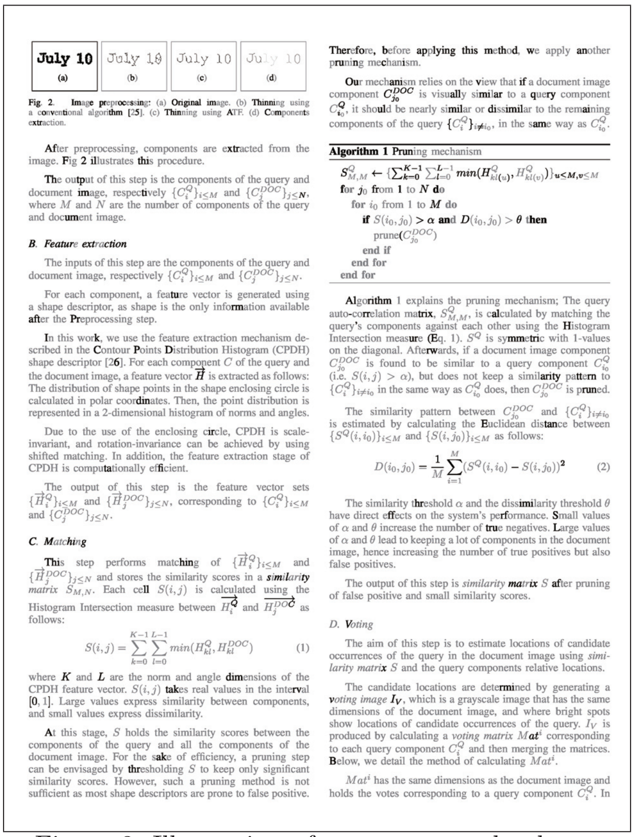

July 10 (a) | July 10 (b) | July 10 (c) | July 10 (d)

Fig. 2. Image preprocessing: (a) Original image. (b) Thinning using a conventional algorithm [25]. (c) Thinning using ATF. (d) Components extraction.

After preprocessing, components are extracted from the image. Fig 2 illustrates this procedure.

The output of this step is the components of the query and document image, respectively $\{C_i^Q\}_{i \le M}$ and $\{C_j^{DOC}\}_{j \le N}$, where M and N are the number of components of the query and document image.

B. Feature extraction

The inputs of this step are the components of the query and document image, respectively $\{C_i^Q\}_{i \le M}$ and $\{C_j^{DOC}\}_{j \le N}$.

For each component, a feature vector is generated using a shape descriptor, as shape is the only information available after the Preprocessing step.

In this work, we use the feature extraction mechanism described in the Contour Points Distribution Histogram (CPDH) shape descriptor [26]. For each component C of the query and the document image, a feature vector $\vec{H}$ is extracted as follows: The distribution of shape points in the shape enclosing circle is calculated in polar coordinates. Then, the point distribution is represented in a 2-dimensional histogram of norms and angles.

Due to the use of the enclosing circle, CPDH is scale-invariant, and rotation-invariance can be achieved by using shifted matching. In addition, the feature extraction stage of CPDH is computationally efficient.

The output of this step is the feature vector sets $\{\vec{H}_i^Q\}_{i \le M}$ and $\{\vec{H}_j^{DOC}\}_{j \le N}$, corresponding to $\{C_i^Q\}_{i \le M}$ and $\{C_j^{DOC}\}_{j \le N}$.

C. Matching

This step performs matching of $\{\vec{H}_i^Q\}_{i \le M}$ and $\{\vec{H}_j^{DOC}\}_{j \le N}$ and stores the similarity scores in a *similarity matrix* $S_{M,N}$. Each cell $S(i,j)$ is calculated using the Histogram Intersection measure between $\vec{H_i^Q}$ and $\vec{H_j^{DOC}}$ as follows:

$$S(i,j) = \sum_{k=0}^{K-1} \sum_{l=0}^{L-1} min(H_{kl}^Q, H_{kl}^{DOC}) \quad (1)$$

where K and L are the norm and angle dimensions of the CPDH feature vector. $S(i,j)$ takes real values in the interval $[0,1]$. Large values express similarity between components, and small values express dissimilarity.

At this stage, S holds the similarity scores between the components of the query and all the components of the document image. For the sake of efficiency, a pruning step can be envisaged by thresholding S to keep only significant similarity scores. However, such a pruning method is not sufficient as most shape descriptors are prone to false positive.

Therefore, before applying this method, we apply another pruning mechanism.

Our mechanism relies on the view that if a document image component $C_{j_0}^{DOC}$ is visually similar to a query component $C_{i_0}^Q$, it should be nearly similar or dissimilar to the remaining components of the query $\{C_i^Q\}_{i \ne i_0}$, in the same way as $C_{i_0}^Q$.

Algorithm 1 Pruning mechanism

$S_{M,M}^Q \leftarrow \{\sum_{k=0}^{K-1} \sum_{l=0}^{L-1} min(H_{kl(u)}^Q, H_{kl(v)}^Q)\}_{u \le M, v \le M}$
for j_0 from 1 to N **do**
  for i_0 from 1 to M **do**
    if $S(i_0, j_0) > \alpha$ **and** $D(i_0, j_0) > \theta$ **then**
      prune($C_{j_0}^{DOC}$)
    end if
  end for
end for

Algorithm 1 explains the pruning mechanism; The query auto-correlation matrix, $S_{M,M}^Q$, is calculated by matching the query's components against each other using the Histogram Intersection measure (Eq. 1). S^Q is symmetric with 1-values on the diagonal. Afterwards, if a document image component $C_{j_0}^{DOC}$ is found to be similar to a query component $C_{i_0}^Q$ (i.e. $S(i,j) > \alpha$), but does not keep a similarity pattern to $\{C_i^Q\}_{i \ne i_0}$ in the same way as $C_{i_0}^Q$ does, then $C_{j_0}^{DOC}$ is pruned.

The similarity pattern between $C_{j_0}^{DOC}$ and $\{C_i^Q\}_{i \ne i_0}$ is estimated by calculating the Euclidean distance between $\{S^Q(i,i_0)\}_{i \le M}$ and $\{S(i,j_0)\}_{i \le M}$ as follows:

$$D(i_0, j_0) = \frac{1}{M} \sum_{i=1}^{M} (S^Q(i,i_0) - S(i,j_0))^2 \quad (2)$$

The similarity threshold α and the dissimilarity threshold θ have direct effects on the system's performance. Small values of α and θ increase the number of true negatives. Large values of α and θ lead to keeping a lot of components in the document image, hence increasing the number of true positives but also false positives.

The output of this step is *similarity matrix* S after pruning of false positive and small similarity scores.

D. Voting

The aim of this step is to estimate locations of candidate occurrences of the query in the document image using *similarity matrix* S and the query components relative locations.

The candidate locations are determined by generating a *voting image* I_V, which is a grayscale image that has the same dimensions of the document image, and where bright spots show locations of candidate occurrences of the query. I_V is produced by calculating a *voting matrix* Mat^i corresponding to each query component C_i^Q and then merging the matrices. Below, we detail the method of calculating Mat^i.

Mat^i has the same dimensions as the document image and holds the votes corresponding to a query component C_i^Q. In

Figure 3. Illustration of component redundancy in a document image taken from [1] The clusters components are highlighted in black, and the redundant component are gray.

compressed using any available compression methodology. Component encoding in the text file is done using a reduced number of points which are sufficient for component reconstruction.

Experimental results show that our algorithm improves efficiency when used for indexing in a content-based document retrieval application, and that the compression performances are competitive. Compression produces very low compression errors that do not compromise the document readability.

We identify several directions to extend and improve the proposed method: In the present paper, centers of clusters are connected components that are extracted using pixel connectivity analysis, and centers similarity is estimated using shape features. In other applications, centers of clusters and centers similarity can be defined according to the image classes (e.g. texture patterns in case of texture images, strokes in case of handwritten signature images, etc.). When image variations such as rotation and scale change are anticipated, the centers descriptor can be tuned or a robust descriptor can be used. Moreover, the centers similarity threshold can be made loose to account for component variations caused by noise.

References

[1] H. Chatbri et al. An application-independent and segmentation-free approach for spotting queries in document images. In *IEEE ICPR*, 2014.

[2] H. Chatbri et al. A modular approach for query spotting in document images and its optimization using genetic algorithms. In *IEEE CEC*, 2014.

[3] D. Doermann. The indexing and retrieval of document images: A survey. *Computer Vision and Image Understanding*, 70(3):287–298, 1998.

[4] P. Haffner et al. High quality document image compression with DjVu. *Journal of Electronic Imaging*, 7(3):410–425, 1998.

[5] P. Haffner et al. DjVu: Analyzing and compressing scanned documents for internet distribution. In *IEEE ICDAR*, pages 625–628, 1999.

[6] H. Imura and Y. Tanaka. Compression and string matching method for printed document images. In *IEEE ICDAR*, pages 291–295, 2009.

[7] D. Karpman et al. Lidar depth image compression using clustering, re-indexing, and JPEG2000. In *SPIE Defense, Security, and Sensing*, pages 80370G–80370G. International Society for Optics and Photonics, 2011.

[8] S. Marinai et al. Digital libraries and document image retrieval techniques: A survey. In *Learning Structure and Schemas from Documents*, pages 181–204. Springer, 2011.

[9] G. Nagy. Twenty years of document image analysis in PAMI. *IEEE PAMI*, 22(1):38–62, 2000.

[10] D. Salomon. *Data compression: the complete reference*. Springer, 2004.

[11] C.-Y. Shiah and Y.-S. Yen. Compression of chinese document images by complex shape matching. *The Computer Journal*, 56(11):1292–1304, 2013.

[12] R. Zanibbi and L. Yu. Math spotting: Retrieving math in technical documents using handwritten query images. In *IEEE ICDAR*, 2011.