FACTA UNIVERSITATIS
Series: **Electronics and Energetics** Vol. 31, N^{o} 3, September 2018, pp. 461-485
https://doi.org/10.2298/FUEE1803461A

EFFICIENT IMAGE COMPRESSION AND DECOMPRESSION ALGORITHMS FOR OCR SYSTEMS

Boban Arizanović, Vladan Vučković

Faculty of Electronic Engineering, Computer Department, Niš, Serbia

Abstract. *This paper presents an efficient new image compression and decompression methods for document images, intended for usage in the pre-processing stage of an OCR system designed for needs of the "Nikola Tesla Museum" in Belgrade. Proposed image compression methods exploit the Run-Length Encoding (RLE) algorithm and an algorithm based on document character contour extraction, while an iterative scanline fill algorithm is used for image decompression. Image compression and decompression methods are compared with JBIG2 and JPEG2000 image compression standards. Segmentation accuracy results for ground-truth documents are obtained in order to evaluate the proposed methods. Results show that the proposed methods outperform JBIG2 compression regarding the time complexity, providing up to 25 times lower processing time at the expense of worse compression ratio results, as well as JPEG2000 image compression standard, providing up to 4-fold improvement in compression ratio. Finally, time complexity results show that the presented methods are sufficiently fast for a real time character segmentation system*.

Key words: *Image processing, image compression, image decompression, OCR, machine-typed documents, machine- printed documents*

1. INTRODUCTION

Character segmentation still presents a considerable challenge in image processing and other related computer science fields [1], and is a very important pre-processing stage in Optical Character Recognition (OCR) systems [2-4]. Character segmentation and character recognition [5-8] have been important subjects of research for many years [9]. Outside of the OCR systems scope, much recent work deals with extracting characters from natural and other non-document images. Taking the recent work into consideration, it is noticeable that the difficulty of character segmentation is usually underestimated compared to the process of character recognition [10,11]. Related works may be classified into those that analyze the character segmentation approach in natural images [12-14], and others that deal with character segmentation in document images. The

Received December 22, 2017; received in revised form April 30, 2018
Corresponding author: Vladan Vučković
Faculty of Electronic Engineering, Computer Department, P.O. Box 73, 18000 Niš, Serbia
(E-mail: vladanvuckovic24@gmail.com)

second group includes machine-printed documents [10,15-18], where the document structure and the shape of its elements are regular, and handwritten documents where character segmentation is challenged due to irregular document structure [11,19-29]. Old machine-typed documents are of particular significance because important historical documents are often in this form [8,30-32].

Recent research includes all levels of character segmentation. Many approaches for skew estimation, as a part of a skew correction process, are modifications of the Hough transform [33-35], with some based on correlation functions or straight line fitting [36,37]. Analyses on document image binarization parameters showed that the Otsu method and other Otsu-based methods give the best results on average [32]. A learning-based approach for finding the best binarization parameters was presented in [38]. Document image compression and decompression methods can also be exploited in the pre-processing stage of the character segmentation system, in order to efficiently store the document images. A survey of image compression algorithms used in wireless multimedia sensor networks (WMSN) was presented in [39]. Compression of large Arabic textual images based on pattern segmentation is achieved using the approach proposed in [40]. Genetic algorithm based on discrete wavelet transformation information for fractal image compression was presented in [41]. Combination of the lapped transform and Tucker decomposition, named as hyperspectral image compression, was proposed in [42]. A lossy image compression technique based on singular value decomposition (SVD) and wavelet difference reduction (WDR) was proposed in [43]. Taking the character segmentation into account, many methods have been proposed. A technique based on searching for connected regions in the spatial domain performed on a binary image was proposed in [44]. A character segmentation method based on Gaussian low-pass filter and innovational Laplace-like transform was proposed in [45]. Segmentation process adapted for real time tasks is proposed in [46] and is based on the Bayes theorem in order to exploit prior knowledge. A novel approach was proposed in [47] based on the usage of contour curvature of letters for identifying the writer of ancient inscriptions and Byzantine codices, without requiring learning algorithms or a database. Diverse methods for segmentation of handwritten documents are proposed [26]. Some techniques exploit clustering in the process of segmentation [28,48]. Gabor filter for feature extraction and Fisher classifier for feature classification were exploited in [49]. To solve the problem of touching characters in handwritten documents, self-organizing maps, SVM classifiers, and Multi-Layer Perceptron are used [21,27,50,51]. For natural images, tensor voting and the three-color bar code for segmentation have been combined [14,52].

This paper presents further improvements of the authors' character segmentation approach, which forms part of a real time OCR system for the needs of the "Nikola Tesla Museum" in Belgrade [53-56]. This paper presents pre-processing methods for document image compression and decompression, which take place after the image binarization. The proposed compression algorithms are based on the RLE data compression algorithm and document character contour extraction, while decompression algorithm exploits the scanline fill algorithm. Together with skew estimation and correction [54], and the image filtering stage, which concludes with image binarization process, this pre-processing is executed independently before the actual segmentation stage. This offers the opportunity to prepare document images for further processing later, to store document images in a compressed form, to use the improved character segmentation and recognition software independently from the pre-processing stage, and also to test independently the character segmentation and recognition stages. The results show that the proposed image compression and decompression methods perform up to 25 times faster than JBIG2 compression at the expense of much lower compression ratio, and are better than JPEG2000 image compression in its lossless mode, giving up to 4-fold improved compression ratio. Compression quality proved to be unimportant with regards to character segmentation accuracy. Additionally, the evaluated image compression and decompression methods proved to be quite efficient and suitable for use in real time system.

This paper is organized as follows: Section 2 provides a description of the related works which deal with image compression, as well as a description of the previously proposed character segmentation approach. Section 3 offers a theoretical foundation for bi-level image compression standards and JPEG/JPEG 2000 image compression standards used for comparison with the proposed algorithms, as well as a description of the RLE algorithm and a scanline fill algorithm used for the proposed methods. Section 4 provides the complete description of the proposed image compression and decompression methods. Pseudo-codes for the proposed image compression and decompression methods are given in Section 5, including the suggestion for the optimal implementation. In Section 6, a large set of experimental results for image compression methods, obtained on different PC machines is provided. Image compression and decompression results from the aspect of compression ratio and time complexity are analyzed in Section 6, including segmentation accuracy results for compressed document images. Finally, discussion of the extended real time character segmentation method, results, and future work are given in Section 7.

2. Related Works

This section gives more detailed descriptions of other image compression methods and authors' existing character segmentation approach.

A novel universal algorithm for lossless chain code compression with a new chain code binarization scheme was proposed in [57]. The compression method is based on the RLE algorithm and the modified LZ77 algorithm. Compression consists of three modes: RLE, LZ77, and COPY mode. The runs of the 0-bits are compressed using RLE, the simplified LZ77 algorithm handles the repetitions within the bit stream, and COPY mode is used if the aforementioned two methods are unsuccessful. On average, this method achieves better compression results than state-of-the-art methods.

An image compression technique for video surveillance based on dictionary learning was presented in [58]. The main concept exploits the camera's being stationary, giving image samples a high level of similarity. The algorithm transforms images over sparsely tailored, over-complete dictionaries previously learned directly from image samples, and thus the image can be approximated with fewer coefficients. Results show that this method outperforms JPEG and JPEG2000 in terms of both image compression quality and compression ratio.

An image compression technique which combines the properties of predictive coding and discrete wavelet coding was proposed in [59]. To reduce inter-pixel redundancy, the image data values are pre-processed using predictive coding. The difference between the predicted and the original values are transformed using discrete wavelet coding. A non-linear neural network predictor is used in the predictive coding system. Results show that this method performs as well as JPEG2000.

A multiplier-less efficient and low complexity 8-point approximate Discrete Cosine Transform (DCT) for image compression was proposed in [60]. An efficient Graphics Processing Unit (GPU) implementation for the presented DCT is provided. It is shown to outperform other approximate DCT transforms in JPEG-like image compression.

The image compression and decompression methods proposed in this research are intended for usage in the authors' existing character segmentation approach. Vučković and Arizanović [53] proposed an efficient character segmentation method for machine-typed documents and machine-printed documents based on the usage of projection profiles. The method consists of pre-processing and segmentation logic. The pre-processing of the character segmentation is focused on manual document skew correction [54], document image grayscale conversion (to perform the document image binarization), and noise reduction. This paper provides an extension of the pre-processing by adding the image compression/decompression to enable the efficient and independent document image storage before the segmentation stage. Segmentation logic is semi-automatic and consists of line, word, and character segmentation. All segmentation levels use the modified projection profiles technique. A new method for segmentation of words into characters based on decision-making logic is the core of the segmentation logic. This method gradually eliminates the possibility for big segmentation errors by determining the number of characters in a word using word width and the assumed average character width for a given document image. Computational efficiency is achieved using the linear image representation, with further implementation optimization using pointer arithmetic and highly-optimized low level machine code. The provided results have shown that this novel method outperforms state-of-the-art techniques in terms of both time complexity and segmentation accuracy.

3. Theoretical Background

This section provides a theoretical foundation for the state-of-the-art image compression standards used for comparison with the proposed algorithms, including the description of the standard algorithms used in the proposed image compression and decompression methods. The state-of-the-art theoretical background provided in this section covers the compression standards for bi-level images, which are especially suitable for document images, as well as a general JPEG and JPEG2000 image compression standards.

3.1. Compression standards for bi-level images

Bi-level images are represented using only 1 bit per each pixel. This bit denotes a black or white color and has a value 0 or 1 depending on the color. For this reason, bi-level images are also referred to as black and white images. Bi-level images usually contain a few specific types of elements such as text, halftone images, and line-art which includes graphs, equations, logos, and other similar features. First compression standards have been designed for facsimile (fax) images. Fax standards include Group 3 (G3), Group 4 (G4), and JBIG standard which is the basis of the later developed JBIG2 standard. G3 standard includes the modified Huffman (MH) coding which combines the variable length codes of Huffman coding with standard RLE coding of the repetitive sequences, and the modified relative element address designate (READ) coding, also called the modified READ (MR) coding. G4 standard uses the modified MR (MMR) coding, which similarly as G3 standard has MH as a basic coder.

The JBIG compression standard recommended by the Joint Bi-Level Image Experts Group is a lossless compression standard used for binary images such as scanned text images, computer-generated text, fax transmissions, etc. This standard can work in three separate modes of operation: progressive, progressive-compatible sequential, and single-progression sequential. Taking the coders into consideration, JBIG uses the arithmetic coding, exploiting the QM coder variant. Context-based prediction is used in the encoding process. In order to ensure a significantly higher compression ratio over the previously described compression standards, the modified lossy version of JBIG has been proposed and named JBIG2. Although the JBIG standard also supports a lossy compression, the lossy compression quality it provides is very low. On the other side, JBIG2 provides both higher compression ratio for lossless compression and lossy compression with a very high compression ratio. JBIG2 supports three basic coding modes: generic, halftone, and text coding mode. Generic coding mode uses either the MMR or MQ variant of arithmetic coding. Halftone coding is used for halftone images. Coding part is based on generic coding using a pattern, having a multi-level image as an output. The decoder obtains the halftone image using the multi-level image and previously used pattern. Finally, dictionary-based text coding is used for textual content. Each representative textual symbol is firstly encoded using the generic coding and is stored in the dictionary together with its position. Decoding is achieved in a straight-forward way, using the dictionary. Difference between the lossy and lossless text compression is in pattern matching type. Lossy compression uses a hard pattern matching and similar letters are coded with the same dictionary entry. Soft pattern matching is used for lossless compression where refinement coding is exploited in order to make a necessary difference between the already stored letter in the dictionary and the current letter. Different modes are used for different document regions. Sometimes text regions are classified as generic regions in order to obtain better results. In average JBIG2 gives 3-5 times higher compression ratio than G4 compression standard and 2-4 times higher compression ratio compared with JBIG standard.

3.2. JPEG and JPEG 2000 image compression standards

JPEG and JPEG 2000 are well-known image compression standards used for comparison with the proposed methods and evaluation of their performance [61]. JPEG, which stands for Joint Photographic Experts Group, is a lossy image compression algorithm. The goal of the JPEG compression algorithm is to eliminate a high frequency colors in the image which cannot be observed by the human eye. This way original and compressed images would be usually the same visually, but the compressed image would be smaller in size.

JPEG image compression algorithm consists of few steps:

1) Image partitioning - The whole image is divided into blocks, size of 8×8. The choice of the block size is also an important part of this step. In case of blocks of bigger size, it is possible to happen that there would exist blocks with big areas of the similar color structure, but since blocks are observed as a whole, those similarities cannot be exploited to obtain a better compression results. On the other side, in case of smaller blocks the processing would be much slower because in the next step a Discrete Cosine Transform (DCT) needs to be performed on each block. For these reasons the blocks size of 8×8 are taken as an appropriate choice.

2) DCT - In this step, DCT is performed on each block matrix. DCT is a similar transform as a Discrete Fourier Transform (DFT) since both transforms map a function to frequency domain, except the fact that DCT uses only cosine function, without dealing with imaginary parts. After derivation of the DCT term, it is noticeable that values of DCT are half of DFT values:

$$f_m = 2\sum_{j=0}^{N-1} f_j \cos\left(\frac{m\pi}{2}\left(j + \frac{1}{2}\right)\right) \quad (1)$$

where f_m is the DFT coefficient. The values of DCT can be defined as:

$$g_m = \sum_{j=0}^{N-1} f_j \cos\left(\frac{m\pi}{2}\left(j + \frac{1}{2}\right)\right) \quad (2)$$

If DCT values are taken into consideration and represented as a matrix, values with lower frequency will be grouped in the upper left side, while the higher frequency values, which are not visible for the human eye, are in other parts of the matrix. The goal of the next step is to eliminate these high frequency values.

3) Elimination of the high frequency values – For this task, it is necessary to multiply the DCT matrix by the appropriate mask matrix. In case of 8 × 8 blocks, the mask matrix that eliminates high frequency values would have the next form:

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

4) Inverse DCT - Finally, the last step in this process is to apply the inverse DCT on each obtained block and use the obtained values to form the new image of the same dimensions as the original image.

In order to describe the JPEG 2000 image compression standard and point out its advantages over the JPEG compression standard, its comparison with JPEG standard would be appropriate at this point:

1) Transformation type - While the previously described JPEG algorithm uses the 8 × 8 DCT, JPEG 2000 uses the wavelet transform with lifting implementation. Using the wavelet transform, the better energy compaction and resolution scalability is obtained.

2) Partitioning domain - While JPEG uses partitioning in the space domain by dividing the image into blocks and applying transformations on each block, JPEG 2000 performs partitioning in the wavelet domain. This way the blocking artifacts which appear during partitioning in the space domain, as a result of partial application of DCT transform, would be eliminated.
3) Entropy coding - JPEG algorithm encodes the DCT coefficients one by one, while JPEG 2000 encodes the wavelet coefficients bitplane by bitplane. In case of JPEG, the resulting bitstream cannot be truncated, while in case of JPEG 2000 truncation is allowed, which enables the bitstream scalability.
4) Rate control - Compression ratio and the amount of distortion when JPEG image compression algorithm is used can be determined by the quantization module, while JPEG 2000 uses the quantization module only for conversion of float wavelet transform coefficients to integer coefficients, and the bitstream assembly module is used to determine the compression ratio and the amount of distortion. This allows final bitstreams of the certain compression ratio to be easily converted to bitstreams of another compression ratio without repeating the entropy coding and transformation process.

3.3. Run-length encoding (RLE) algorithm

RLE is a standard widely used lossless data compression algorithm. The logic of this algorithm is to replace each repeating of some specific pattern with a symbol which describes that pattern and a value which defines the number of consecutive repeats of that pattern in the given sequence. In literature, the application of this algorithm to text compression is usually explained. The simplest example of algorithm application is in case that pattern is a single character. Suppose that the next sequence of characters is given:

$$ssaaacccasdaaeaaaaaaasdddaddddwwwa$$

In this case, the compressed text will be the following:

$$2s3a3c1a1s1d2a1e7a1s3d1a5d3w1a$$

The original text has 35 characters, while the compressed text has 30 characters. The gain in this concrete example is not significant, but in practice algorithm can deal with a large amount of binary values and can be very efficient. The larger runs of the same values exist in sequence, the higher compression ratio will be achieved. In general case, the pattern does not necessarily need to be a single character. It could be a word or even a sentence. In such cases, it is also mandatory to use a delimiter between the compressed information for each pattern run, since it is unknown how many characters each pattern has and decompression would be impossible without having this information. The pseudo-code for general RLE algorithm is shown in the following listing.

```
RUN-LENGTH-ENCODING
Input:
  sequence s.
Output:
  array Compressed.
 1:    I := 0
 2:    while I < LENGTH(s) do
```

```
CurrentPattern := s[I]
RunLength := 0
while I < LENGTH(s) AND s[I] = CurrentPattern do
RunLength := RunLength + 1
I := I + 1
end while
Compressed ← RunLength
Compressed ← CurrentPattern
end while
return Compressed
```

The sign ← is used to represent the assignment operator for array.

3.4. Scanline fill algorithm

Scanline fill algorithm belongs to the region filling algorithms. Instead of algorithms with flooding approach which fill the contour by coloring the connected pixels of the same color at pixel level, the scanline fill algorithm is defined at geometric level and fills the contour in a horizontal or vertical direction, i.e. row by row or column by column. Illustration of the scanline fill algorithm is shown in Fig. 1.

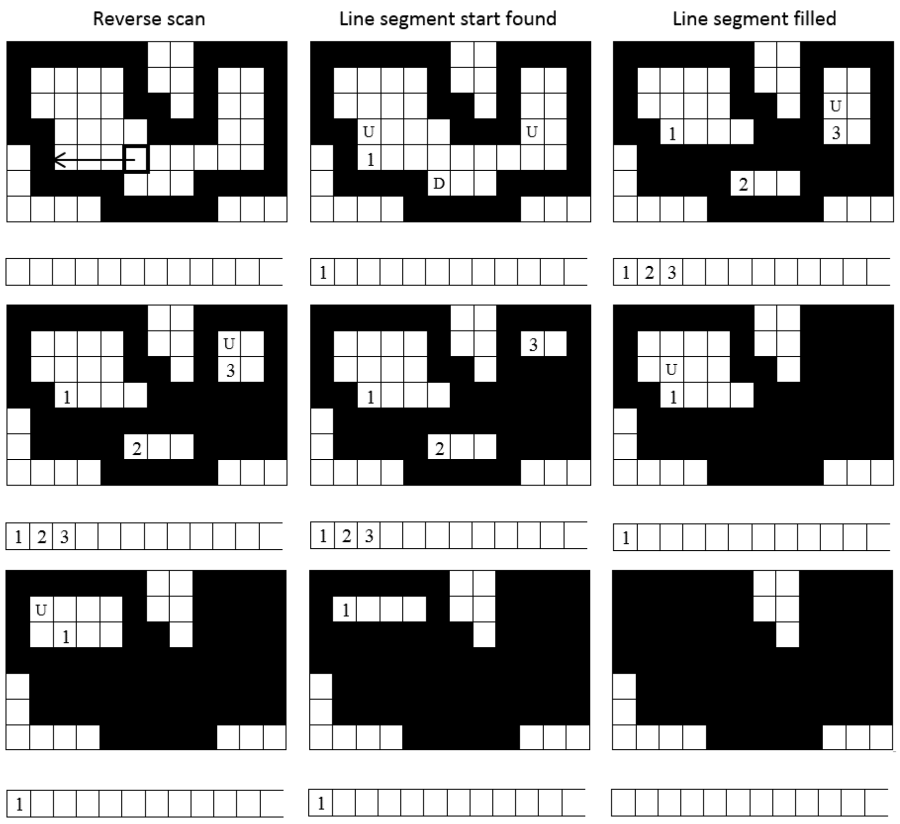

Fig. 1 Illustration of the scanline fill algorithm

As it is shown in Fig. 1, the algorithm starts from a seed pixel and searches for the last pixel on the left side which color is the same as the seed pixel color. Once that pixel is found, the pixel row is processed from left to right until the pixel which color is not the same as the seed pixel color or the end of the pixel scanline is encountered. The color of each pixel in the current pixel row is changed to the fill color if its color is the same as the seed pixel color. Also, the pixels above and below the current pixel are checked and they are pushed to stack if their color is the same as the seed pixel color. Once the pixels above or below the current pixel are pushed to stack, the next pixels above and below are not considered until the new sequence of pixels which color is the same as the seed pixel color is not encountered. The next iteration starts when a new pixel is taken from the top of the stack. In fact, instead of pushing to stack the coordinates of the all individual pixels which need to be processed, this algorithm pushes the start coordinates of the line segments. This ensures that each pixel is checked once and leads to better time complexity results compared with flood fill algorithm, which is the main reason to use this algorithm for image decompression.

4. PROPOSED ALGORITHMS

This section proposes new image compression and decompression methods used in the pre-processing stage of the character segmentation system, in order to compress and decompress document images before the segmentation stage. As it has already been mentioned, previously presented character segmentation approach is extended by adding the image compression/decompression part in the pre-processing stage. This step gives the possibility to divide the character segmentation system into two independent parts. The first part is a pre-processing part having the document image compression and decompression as a final process, and the second part is a document image segmentation. The most evident gain achieved here is the possibility to execute two independent system parts in different moments. This is important due to several reasons. This way a document image compression and document image segmentation can be done on different machines. Additionally, the previously compressed/decompressed document images can be processed using different versions of the segmentation engine. This is very important feature since it also allows efficient testing of the segmentation engine. Finally, image compression allows efficient storing of document images which can save a significant disc space.

In this section two image compression and decompression methods are proposed. The first method is completely based on RLE algorithm and can be used for both machine-typed documents and machine-printed documents. The second method uses document character contour extraction in combination with scanline fill algorithm. The second method works with machine-printed documents, but its application on machine-typed documents is limited due to irregular structure of document image characters caused by low quality of documents. These image compression and decompression methods are presented in the following subsubsections.

4.1. Image compression and decompression using RLE

The first proposed image compression and decompression methods used in the pre-processing stage of character segmentation system employ the RLE algorithm for data compression. After the image binarization, the image consists of black and white pixels, thus the RLE algorithm proved to be excellent choice. This approach is general and can be used for all types of binarized images, but RLE algorithm gives better compression results in case of document images than e.g. in case of natural images. The reason for this lies in fact that RLE algorithm searches for runs of black and white pixels in each pixel scanline. In natural images these runs are short, while in document images the background has large runs of white pixels. Illustration of the RLE algorithm including the coding format is given in Fig. 2.

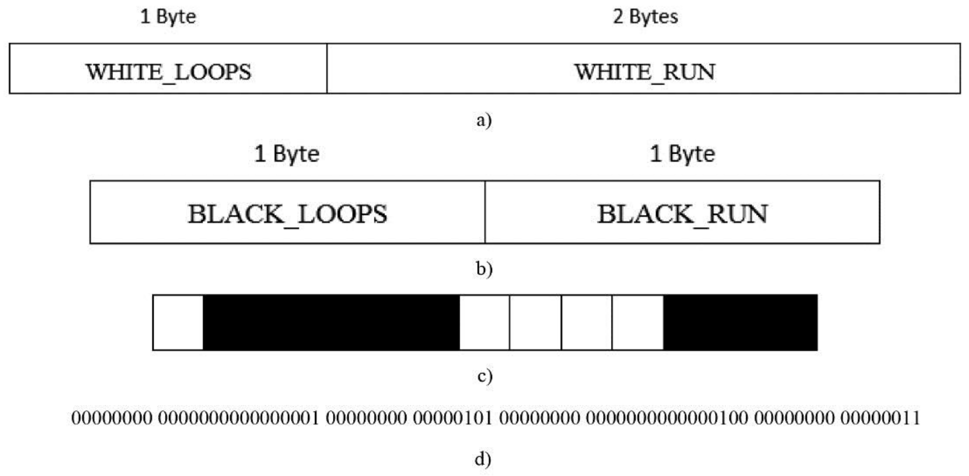

Fig. 2 Document image compression using RLE algorithm: (a) Compression format for white pixel runs, (b) Compression format for black pixel runs, (c) Example of pixel scanline, (d) Compressed pixel scanline.

Fig. 2 shows an example of image compression using RLE algorithm. RLE algorithm counts white and black pixels and stores the information about pixel runs in compressed file. Storing is achieved using 3 bytes in overall for information about the white pixels and 2 bytes for information about the black pixels. In both cases the pixels are counted until the maximal value is reached. Since 2 bytes are used for white pixels (WHITE_RUN), this value is 216 = 65536, while in case of black pixels (BLACK_RUN) this value is 28 = 256. When these values are reached, the WHITE_LOOPS or BLACK_LOOPS byte is incremented and WHITE_RUN and BLACK_RUN values are set to 0. The whole process of counting is then repeated. Since white pixels are a part of the background and are dominating in document images, it is expected that 1 byte is not enough for storing the information about the number of consecutive white pixels. On the other side, black pixel runs are not expected to be too long since they represent document characters and some spaces between characters are expected, thus only 1 byte is used for storing this information.

Document image decompression is straight forward. First byte is always multiplied by 256 or 65536 in case of black pixels or white pixels, respectively. After that, this value is incremented by value of the next byte or 2 bytes. The obtained value represents the number of consecutive pixels of the same color in the current run of pixels. This process is repeated until the end of the compressed file.

4.2. Image compression and decompression using document character contour extraction and scanline fill algorithm

The second proposed method for image compression and decompression employs the combination of the algorithm based on document character contour extraction and scanline fill algorithm. The compression algorithm uses the 2D processing of a document image, since character contours need to be obtained from all four sides. Document image is processed in horizontal and vertical direction and distances between the black pixels which represent starting and ending pixels of the black runs are stored in the compressed file. For this purpose, 2 bytes can be used to store the distance between two black pixels. It should be mentioned that in both compression methods the number of bytes used for storing the information about the white and black pixel runs is dependent primarily on the image dimensions. Small images are expected to have short runs, while large images are expected to have long runs of pixels of the same color. Therefore, 1 byte can be used for both white and black pixels in case of small images, while in case of large images 2 bytes are necessary. Another important factor is a structure of a document image. If textual content is dominating in a document image, background areas are not huge and even in large images 1 byte can be used for storing the information about pixel runs. On the other side, if background area is dominating, even in medium images 2 bytes would not be enough to store the information about pixel runs.

The process of image decompression is more specific. After obtaining the offsets of black pixels which represent the contours of the characters, in the first step of decompression method contours are drawn to the output image. The second step uses the iterative scanline fill algorithm. The main idea here is to scan the whole output image and fill the contours which represent the background with background color, while character contours will be filled with black color. This is achieved by repeating execution of the scanline fill algorithm. After that, a background color of a document image is replaced with white color and the original binarized document image is obtained. The illustration of image decompression is shown in Fig. 3.

It is assumed that the first pixel in a document image is a background pixel, thus the scanline fill algorithm starts execution from the first pixel. The next step is filling of the closed contours which are the document characters contours. In order to fill these contours, the algorithm searches for the next white pixel. When the next white pixel is found, the colors of the previous two or three pixels are checked. It is also assumed that contour edge is not wider than two pixels, but generally this does not need to be a case. In case that the previous two pixels have the black color and background color, or in case that the previous three pixels have black color, black color, and background color, the character contour is found and needs to be colored in black using scanline fill algorithm. This procedure is repeated until the end of a document image. The final image will contain only the background color and the black color. The final step is to change the background color back to white color and binarized document image will be obtained.

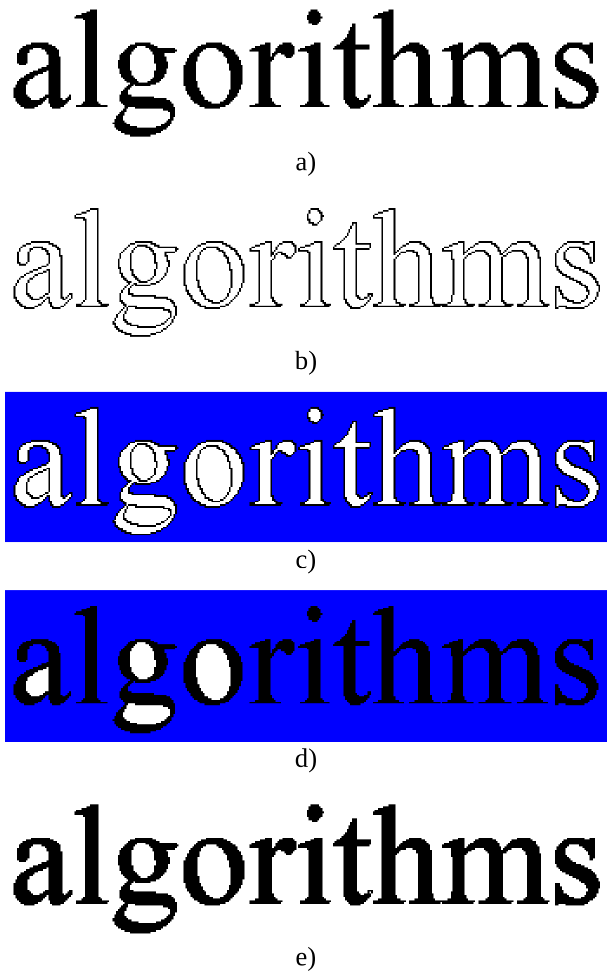

Fig. 3 Document image decompression using scanline fill algorithm: (a) Original image, (b) Resulting image after character contours extraction, (c) Resulting image after background filling, (d) Resulting image after character contours filling, (e) Final binarized image.

5. Implementation

This section provides pseudocodes for the proposed image compression and decompression methods. In order to achieve an efficient implementation, linear image representation could be used. Linear image representation is obtained by storing the image pixels linearly in a one-dimensional array. This representation is efficient since the memory organization is also linear and image pixels will be stored in successive memory locations, which will provide the fastest possible access to the image elements. The first presented method for image compression and decompression uses the RLE data compression algorithm. The pseudocode of the image compression algorithm is shown below.

```
RLE-IMAGE-COMPRESSION
Input:
  image f.
Output:
  array Compressed.
CurrentColor := WHITE
while not end of image f do
    RunsCount := CALCULATE-RUN-LENGTH(f, CurrentColor)
    Factor := RunsCount div MaxRun
    RunLength := RunsCount mod MaxRun
    Compressed ← Factor
    if CurrentColor = BLACK then
        Compressed ← RunLength
    else
        Compressed ← RunLength div 256
        Compressed ← RunLength mod 256
    endif
    REPLACE-MAX-RUN(MaxRun)
    REPLACE-CURRENT-COLOR(CurrentColor)
end while
return Compressed
```

This pseudocode represents a general application of the RLE algorithm for compression of binarized images. This algorithm provides lossless image compression and its application is not limited on document images, but as will be shown in the experimental section, this method provides better compression results when applied on document images. In order to provide the optimal compression, 2 bytes are used for storing the information about black pixel runs and 3 bytes are used for storing the information about white pixel runs. The Image decompression method is straightforward. The following listing shows the pseudocode of the decompression algorithm.

```
RLE-IMAGE-DECOMPRESSION
Input:
  image f, array Compressed.
CurrentColor := WHITE
while not end of Compressed do
    RunLength := 0
    Factor := Compressed[Current]
    if CurrentColor = BLACK then
        RunLength := Compressed[Current + 1]
    else
        RunLength := Compressed[Current + 1] * 256
        RunLength := RunLength + Compressed[Current + 2]
    endif
    FILL-RUN(f, RunLength, CurrentColor)
    REPLACE-CURRENT-COLOR(CurrentColor)
end while
```

The image decompression algorithm reads the pixel runs information from the compressed file and regenerates the original image. Since the compression is lossless, the decompressed image will be the same as the original image, but results from the experimental section will prove that image compression quality is actually unimportant for segmentation accuracy. The pseudocode from the previous listing corresponds to the pseudocode for image compression shown in the first listing, regarding the number of bytes used for representing the pixel runs.

The second proposed method for document image compression and decompression uses the character contour based document image representation and the scanline fill algorithm. Contour image representation is used for image compression, while image decompression uses the iterative scanline fill algorithm. The image compression algorithm performs the 2D image processing to obtain the distances between the black pixels which form the character contours in a document image. Depending on the document image dimensions, 1 byte or 2 bytes can be used to store the distance between two black pixels which represent edges of document character contours. The following listing shows the pseudocode for the document image compression algorithm based on character contour extraction.

```
CONTOUR-IMAGE-COMPRESSION
Input:
  image f.
Output:
  array Compressed.
for each pixel row in image f do
    CurrentColor := WHITE
    while not end of current pixel row do
        RunsCount := CALCULATE-RUN-LENGTH(f, CurrentColor)
        Compressed ← RunsCount div 256
        Compressed ← RunsCount mod 256
        REPLACE-CURRENT-COLOR(CurrentColor)
    end while
end for
for each pixel column in image f do
    CurrentColor := WHITE
    while not end of current pixel column do
        RunsCount := CALCULATE-RUN-LENGTH(f, CurrentColor)
        Compressed ← RunsCount div 256
        Compressed ← RunsCount mod 256
        REPLACE-CURRENT-COLOR(CurrentColor)
    end while
end for
return Compressed
```

The previous pseudocode describes the algorithm for image compression which uses the 2D image analysis to obtain the pixels which represent edges of the document character contour. Generally, this algorithm can be used for all binarized images, but its application on document images is more effective. The reason is that binarized document images have less details compared with say, natural images. Also, document images have large areas with background color which will ensure a good compression ratio. With regards to compression quality, this algorithm gives worse results than other methods, but it does not affect the segmentation accuracy. This fact justifies the usage of this compression algorithm in the character segmentation system. The following listing shows the pseudocode for document image decompression based on the scanline fill algorithm.

```
SCANLINE-FILL-IMAGE-DECOMPRESSION
Input:
  Image f, array Compressed.
Offset := 0
while not end of row compression bytes do
RunLength := Compressed[Current] * 256
RunLength := RunLength + Compressed[Current + 1]
Offset := Offset + RunLength
SET-PIXEL-COLOR(f, Offset, BLACK)
end while
Offset := 0
while not end of column compression bytes do
RunLength := Compressed[Current] * 256
RunLength := RunLength + Compressed[Current + 1]
Offset := Offset + RunLength
SET-PIXEL-COLOR(f, Offset, BLACK)
end while
SCANLINE-FILL(0, BLUE_COLOR, WHITE_COLOR)
for each pixel in f do
if GET-PIXEL-COLOR(f, Current) = WHITE then
if GET-PIXEL-COLOR (f, Current – 1, Current - 2) = [BLACK, BLUE] OR
GETPIXEL-COLOR(f, Current – 1, Current – 2, Current - 3) = [BLACK,
BLACK, BLUE] then
SCANLINE-FILL (Current, BLACK, WHITE)
endif
endif
SCANLINE-FILL(0, WHITE, BLUE)
```

As aforementioned, the pseudocode for image decompression using the scanline fill algorithm can be applied in the case of machine-printed documents. The efficiency of the decompression algorithm is highly influenced by the efficiency of the scanline fill algorithm implementation, since the scanline fill algorithm must be executed multiple times. In the pseudocode presented in the previous listing, two and three previous pixels of the current pixel are checked. In the general case, this conditional statement can be changed since it is possible that character contour edges are wider than 1 or 2 pixels as it is assumed here. Using the linear image representation, as suggested at the start of this section, it is possible to achieve a real time implementation of the proposed methods, which will make them suitable for use in the real time character segmentation system.

6. EXPERIMENTS

Proposed image compression and decompression methods, as a part of character segmentation system, are tested on several PC machines. Results are analyzed from different aspects in order to provide the complete insight into the extended character segmentation approach and its capabilities. Image compression and decompression methods are evaluated from the perspective of the image compression ratio and time complexity, to the perspective of the segmentation accuracy when specific compression methods are used. Evaluation of the image compression ratio is performed using the standard test set of images. Test set consists of six black and white images: Baboon, Barbara, Cameraman, Goldhill, Lena, and Peppers. Each pixel intensity value is represented using 3 bytes, one byte for red, green, and blue pixel intensity value component. In order to obtain the comparative results, JBIG2 and JPEG2000 image compression standards are used. For both image compression standards, the performances of their lossless modes are evaluated.

The most important metric for evaluating the compression methods is compression ratio. In case of images, compression ratio is a ratio between the original image file size and compressed image file size. Higher values for compression ratio mean that compression method is better regarding this metric. Comparison of the image compression ratio for standard set of images and different compression methods is given in Table 1.

Table 1 Comparison of the compression ratio results for different compression methods

Image (Image Dimensions)	Image File Size (KB)	Compression Ratio		
		JPEG2000 (Lossless)	JBIG2 (Lossless)	**RLE**
Baboon (512x512)	769	3.544	46.048	**8.010**
Barbara (512x512)	769	5.961	113.255	**9.859**
Cameraman (256x256)	193	3.642	82.833	**12.867**
Goldhill (512x512)	769	6.303	111.288	**19.718**
Lena (512x512)	769	7.539	156.939	**21.971**
Peppers (512x512)	769	8.640	201.837	**30.76**

As it is visible from Table 1, JBIG2 compression gives much better compression ratio results than the proposed algorithm. This comes from the sophisticated nature of the JBIG2 algorithm, which is specialized for black and white images. Compression ratio and time complexity are two most important measures for the quality of the compression algorithm. Although the proposed algorithms fail to surpass the JBIG2 compression ratio results, the time complexity results shown later will justify the usage of the proposed methods. Additionally, Table 1 shows that RLE based image compression method presented in this paper provides a higher compression ratio than JPEG2000 image compression standard in its lossless mode. RLE based method is not limited to document images, therefore it can be used for non-document images as it is a case with standard test images. It is clear that this method ensures the possibility to store a huge amount of compressed document images efficiently without occupying a lot of disc space. The second presented compression method is limited to specific document images and performance of this method is analyzed later in this section.

Previous results represent the general analysis of the image compression methods. Since the proposed image compression and decompression methods will be used in character segmentation system, their performances on document images should be analyzed. In order to perform this analysis, image compression methods are tested using two document images. These document images are machine-printed documents since the second method is limited on machine-printed documents which have the regular structure and character contours can be extracted correctly. Compression ratio results for these document images and different image compression methods are shown in Table 2.

Table 2 Comparison of the image compression ratio for machine-printed document images for different image compression methods

Image Dimensions	Image File Size (KB)	Compression Ratio			
		JPEG2000 (Lossless)	JBIG2 (Lossless)	**RLE**	**Contour Extraction/Scanline Fill**
719x328	692	14.417	640.741	**67.184**	**46.443**
1266x924	3429	9.741	357.933	**27.878**	**19.373**

As it is expected, the JBIG2 compression again achieves much better compression ratio results. In this case the compression ratio is much higher than in case of standard test set images because JBIG2 has a separate mode for compression of a textual content, as it is explained in Section 3. Taking the proposed methods into account, they perform very well on document images compared with JPEG2000 standard. In overall, RLE based method gives the second best results, while the contour extraction method provides worse, but still competitive results. It is important to mention that JPEG2000 image compression standard performs worse than RLE algorithm because of its nature. The color transformation stage of JPEG2000 algorithm generates two color channels which are being compressed in this stage. The channel related to black and white image features is not compressed the same way, thus the JPEG2000 compression gives worse results in case of black and white images, i.e. gives better compression ratio results in case of full-color images with many color and contrast transitions.

Contour extraction based compression in combination with scanline fill decompression is a lossy compression, since it cannot reconstruct the original image perfectly. Although it could imply that this method does not perform well enough, further analysis will deny this fact. In order to justify the usage of the second proposed image compression method, the segmentation results for document images previously compressed and decompressed using different methods are given in Table 3.

Table 3 Comparison of the segmentation accuracy results for different image compression methods used in the pre-processing stage

	Segmentation Accuracy (%)		
	JBIG2/JPEG2000 (Lossless)	**RLE**	**Contour Extraction/Scanline Fill**
Line Segmentation	81.54	**81.54**	**80.32**
Word Segmentation	78.28	**78.28**	**78.14**
Character Segmentation	87.08	**87.08**	**86.92**

These results are obtained using the chosen ground-truth machine-printed documents. As expected, the results for JBIG2, JPEG2000, and RLE based compression are identical. The most important conclusion here is that contour extraction based compression in combination with scanline fill decompression gives slightly worse results than previous compression methods. The reason for this lies in sensitivity of the evaluation metrics and also in the specificity of the character segmentation technique. In general, this technique is not sensitive on small changes in document image structure and therefore the segmentation accuracy results are similar to those obtained using the lossless compression methods. Fig. 4 shows the comparison of the original binarized image and image obtained after compression and decompression using the second proposed method.

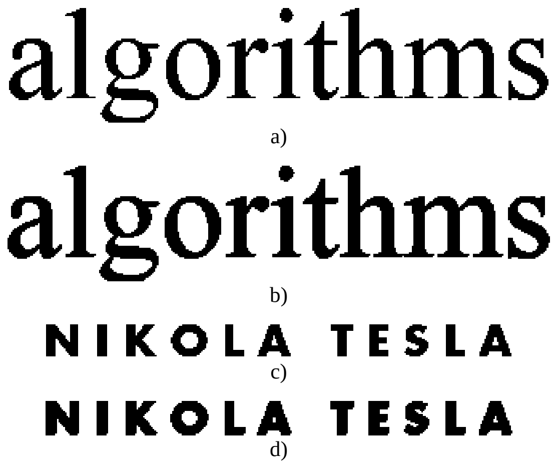

Fig. 4 Comparison of the original images and images obtained after compression/decompression using the second method: (a) First original image, (b) First image after compression/decompression, (c) Second original image, (d) Second image after compression/decompression

Visual results from Fig. 4 clearly show that compression quality is sometimes irrelevant. Although the second method is the worst among all analyzed methods regarding the compression quality and provides a lossy compression, differences between original and final images are negligible in case of character segmentation. In order to clearly demonstrate this conclusion, Fig. 5 shows the same images from Fig. 4 after being processed using the character segmentation algorithm.

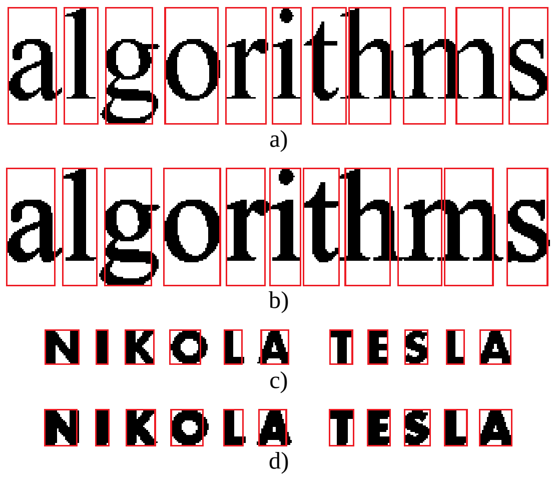

Fig. 5 Comparison of the original and compressed images after being processed using the character segmentation algorithm: (a) First original processed image, (b) First image processed after compression/decompression, (c) Second processed original image, (d) Second image processed after compression/decompression

Finally, a very important aspect of the image compression methods is time complexity. The image compression methods are intended for the usage in real-time character segmentation system and their processing time should be appropriate for that. In order to provide the reliable results, proposed image compression and decompression methods are tested on several PC machines. Comparison of the processing time for JBIG2 compression and compression/decompression performed using the proposed methods is given in Table 4.

Table 4 Comparison of the processing time for JBIG2 compression and RLE based compression and decompression method (AMD Athlon™ X4 840 Quad Core Processor 3.1 GHz)

Image dimensions (pixels)	White Pixels/ Black Pixels (%)	Processing Time (ms)					
		JBIG2 Compression		RLE Compression		RLE Decompression	
719x328	93.09:6.91	9.36832	10.12405	**0.26639**	**0.30149**	**0.23231**	**0.27278**
1266x924	83.51:16.49	44.64481	49.82263	**2.04907**	**2.59600**	**1.94450**	**3.05122**
2632x3575	98.06:1.94	353.74121	382.56647	**10.61898**	**15.07239**	**13.35769**	**21.82402**
2640x3612	98.69:1.31	372.48256	405.33698	**10.79404**	**16.99354**	**13.31302**	**21.95595**

Time complexity comparison justifies the usage of the proposed algorithms in the authors' character segmentation system. Numerical results in Table 4 are obtained after 10000 executions of the analyzed algorithms implementations. Table 4 shows the best (left) and average (right) processing time. RLE based compression algorithm provides up to 25 times faster compression than the JBIG2 compression standard. The reason for this lies in simplicity of the proposed compression method, as well as in the complexity of the JBIG2 compression. This advantage of the proposed compression method comes to the fore with a large number of documents that need to be processed in the "Nikola Tesla Museum" in Belgrade. In order to provide a reliable time complexity results, processing time are given for a set of different PC machines. These results are shown in Tables 5-14.

Table 5 Processing time for RLE based compression and decompression method (AMD Athlon™ X4 840 Quad Core Processor 3.1 GHz)

Image dimensions (pixels)	White Pixels/ Black Pixels (%)	Processing Time (ms)			
		RLE Compression		RLE Decompression	
719x328	93.09:6.91	0.26639	0.30149	0.23231	0.27278
1266x924	83.51:16.49	2.04907	2.59600	1.94450	3.05122
2632x3575	98.06:1.94	10.61898	15.07239	13.35769	21.82402
2640x3612	98.69:1.31	10.79404	16.99354	13.31302	21.95595

Table 6 Processing time for RLE based compression and decompression method (AMD Athlon™ 64 X2 Dual Core Processor 5200+ 2.6 GHz)

Image dimensions (pixels)	White Pixels/ Black Pixels (%)	Processing Time (ms)			
		RLE Compression		RLE Decompression	
719x328	93.09:6.91	0.59617	0.64098	0.45145	0.48502
1266x924	83.51:16.49	4.54918	4.98643	3.32389	3.93771
2632x3575	98.06:1.94	18.97615	22.70397	23.82398	27.92950
2640x3612	98.69:1.31	18.41798	35.85595	24.20699	42.99031

Table 7 Processing time for RLE based compression and decompression method (Intel® Core™ i3-4150 CPU @ 3.50GHz)

Image dimensions (pixels)	White Pixels/ Black Pixels (%)	Processing Time (ms)			
		RLE Compression		RLE Decompression	
719x328	93.09:6.91	0.22140	0.23022	0.14838	0.15094
1266x924	83.51:16.49	1.34219	1.39174	0.86508	0.90333
2632x3575	98.06:1.94	7.57190	7.81398	5.38810	5.69849
2640x3612	98.69:1.31	7.36194	7.66240	5.28898	5.67878

Table 8 Processing time for RLE based compression and decompression method (Intel® Core™ i5-750 CPU @ 2.67GHz)

Image dimensions (pixels)	White Pixels/ Black Pixels (%)	Processing Time (ms)			
		RLE Compression		RLE Decompression	
719x328	93.09:6.91	0.44596	0.45164	0.53324	0.53898
1266x924	83.51:16.49	2.92499	2.99345	2.82125	2.86606
2632x3575	98.06:1.94	15.47632	16.07258	21.01470	22.01499
2640x3612	98.69:1.31	15.18539	15.83180	21.34199	21.78855

Table 9 Processing time for RLE based compression and decompression method (Intel® Core™ i7-920 CPU @ 2.67GHz)

Image dimensions (pixels)	White Pixels/ Black Pixels (%)	Processing Time (ms)			
		RLE Compression		RLE Decompression	
719x328	93.09:6.91	0.42999	0.44189	0.31279	0.31529
1266x924	83.51:16.49	2.74480	2.84551	1.94092	1.95931
2632x3575	98.06:1.94	15.46358	15.60705	11.13561	11.33740
2640x3612	98.69:1.31	15.24224	15.38186	10.98920	11.21025

Table 10 Processing time for RLE based compression and decompression method (Intel® Core™ i7-4700 MQ Processor 2.4 GHz)

Image dimensions (pixels)	White Pixels/ Black Pixels (%)	Processing Time (ms)			
		RLE Compression		RLE Decompression	
719x328	93.09:6.91	0.32331	0.34429	0.21639	0.22497
1266x924	83.51:16.49	1.93728	2.05009	1.24961	1.31144
2632x3575	98.06:1.94	11.24646	11.64243	7.86414	8.18985
2640x3612	98.69:1.31	10.96036	11.31501	7.70805	8.34529

Table 11 Processing time for RLE based compression and decompression method (Intel® Core™2 Quad Q9550 CPU @ 2.83GHz)

Image dimensions (pixels)	White Pixels/ Black Pixels (%)	Processing Time (ms)			
		RLE Compression		RLE Decompression	
719x328	93.09:6.91	0.41861	0.42261	0.29742	0.29985
1266x924	83.51:16.49	2.61796	2.66822	1.86799	1.96971
2632x3575	98.06:1.94	14.93945	15.69766	13.77433	17.14845
2640x3612	98.69:1.31	14.78641	15.59181	13.69504	17.03617

Table 12 Processing time for RLE based compression and decompression method (Intel® Core™ i5-3470 CPU @ 3.20GHz)

Image dimensions (pixels)	White Pixels/ Black Pixels (%)	Processing Time (ms)			
		RLE Compression		RLE Decompression	
719x328	93.09:6.91	0.24246	0.25897	0.17094	0.18088
1266x924	83.51:16.49	1.42943	1.54486	1.00416	1.06919
2632x3575	98.06:1.94	9.77028	10.26213	6.09392	6.58797
2640x3612	98.69:1.31	9.61185	10.10193	5.96242	6.46359

Table 13 Processing time for RLE based compression and decompression method (Intel® Celeron® E3400 CPU @ 2.60GHz)

Image dimensions (pixels)	White Pixels/ Black Pixels (%)	Processing Time (ms)			
		RLE Compression		RLE Decompression	
719x328	93.09:6.91	0.45608	0.58744	0.34018	0.63256
1266x924	83.51:16.49	3.00126	3.56548	2.50430	3.85496
2632x3575	98.06:1.94	16.44169	20.02098	20.42599	27.38618
2640x3612	98.69:1.31	16.07075	19.90696	20.32023	27.60395

Table 14 Processing time for contour extraction based compression method and scanline fill decompression method (document image size of 719x328 pixels)

PC Machine Specification	Processing Time (ms)			
	Contour Compression		Scanline Fill Decompression	
AMD Athlon™ X4 840 Quad Core Processor 3.1 GHz	0.26672	0.29824	1.59406	2.06575
AMD Athlon™ 64 X2 Dual Core Processor 5200+ 2.6 GHz	1.10936	1.24688	2.94814	6.71124
Intel® Core™ i3-4150 CPU @ 3.50GHz	0.50409	0.52009	1.16008	1.18575
Intel® Core™ i5-750 CPU @ 2.67GHz	1.05348	1.06509	1.94235	1.95962
Intel® Core™ i7-920 CPU @ 2.67GHz	1.13896	1.22269	1.93016	1.97598
Intel® Core™ i7-4700 MQ Processor 2.4 GHz	0.74069	0.75961	1.69607	1.72861
Intel® Core™2 Quad Q9550 CPU @ 2.83GHz	0.89366	0.90396	1.85968	1.87623
Intel® Core™ i5-3470 CPU @ 3.20GHz	0.53816	0.57638	1.19594	1.27253
Intel® Celeron® E3400 CPU @ 2.60GHz	0.98342	1.82706	2.36290	3.41929

Time complexity results shown in Tables 5-14 are also obtained after 10000 executions of analyzed algorithms implementations. In the previous tables, the average and the best processing time are given. Provided results prove the superiority of the linear image representation used in the character segmentation system. All processing time are below 50ms which is a quite good result for real-time usage. The first method also achieves very good results in case of document images with bigger number of black pixels which is a characteristic of documents with predominant textual content. The second method processing time is primarily affected by the number of closed contour which represent character borders and need to be filled using scanline fill algorithm. Each time the closed contour needs to be filled, scanline fill algorithm for region filling needs to be executed. Since the document image used for obtaining the results in Table 14 has huge areas of the same color and small number of characters, the second method proved to be very efficient.

7. Conclusions

This paper presents an image compression/decompression stage of the authors' existing character segmentation approach. In Section 2 the description of the other image compression methods and the authors' character segmentation approach is given. Section 3 provides a theoretical background for bi-level image compression standards and a theoretical comparison of the JPEG and JPEG 2000 image compression standards used for evaluation of the proposed methods, as well as a description of the RLE data compression algorithm and scanline fill algorithm exploited for the proposed image compression and decompression algorithms. In Section 4 the image compression and decompression methods are presented. The presented image compression and decompression methods are adapted to document images and use the RLE data compression algorithm and document character contour extraction for image compression, and the scanline fill algorithm for document image decompression. The decoupling of the image compression/decompression stage allows preparation document images for further processing later, to keep the document images in compressed form and save storage memory, to use the improved character segmentation and recognition software independently from the pre-processing stage, and also to test independently the character segmentation and recognition stages. In Section 5, pseudocodes and suggestion for optimal implementation of the proposed image compression and decompression algorithms are given. In Section 6 a large set of experimental results is provided for image compression methods. The proposed image compression algorithms perform up to 25 times faster than the JBIG2 image compression and 4 times better than JPEG2000 image compression standard in its lossless mode regarding the image compression ratio, whereas the proposed algorithms give much worse compression ratio results compared with JBIG2 compression standard. The results proved that image compression quality does not affect the segmentation accuracy. Also, the proposed image compression and decompression methods proved to be suitable for a real time character segmentation system. The official evaluation of the complete system performance will be performed at the "Nikola Tesla Museum", since the approach is designed for the needs of the museum, namely for conversion of the original Nikola Tesla documents to electronic form. Future work will be focused on algorithms improvement, particularly of the segmentation logic, further approach optimization and automation, and its integration into the complete OCR system.

Acknowledgments: *This paper is supported by the Ministry of Education, Science and Technological Development of the Republic of Serbia (Project III44006-10), Mathematical Institute of Serbian Academy of Science and Arts (SANU), Museum of Nikola Tesla (providing original typewritten documents of Nikola Tesla), and Pattern Recognition & Image Analysis Research Lab (PRimA) (providing ground-truth historical machine-printed documents).*

REFERENCES

[1] A. Andreopoulos and J. K. Tsotsos, "50 Years of object recognition: Directions forward", *Computer Vision and Image Understanding*, vol. 117, no. 8, pp. 827-891, 2013.

[2] N. Bourbakis, N. Pereira and S. Mertoguno, "Hardware design of a letter-driven OCR and document processing system", *Journal of Network and Computer Applications*, vol. 19, no. 3, pp. 275-294, 1996.

[3] S. Khoubyari and J. J. Hull, "Font and Function Word Identification in Document Recognition", *Computer Vision and Image Understanding*, vol. 63, no. 1, pp. 66-74, 1996.

[4] J. Mao and K. M. Mohiuddin, "Improving OCR performance using character degradation models and boosting algorithm", *Pattern Recognition Letters*, vol. 18, no. 11-13, pp. 1415-1419, 1997.

[5] A. Namane, A. Guessoum, E. H. Soubari and P. Meyrueis, "CSM neural network for degraded printed character optical recognition", *Journal of Visual Communication and Image Representation*, vol. 25, no. 5, pp. 1171-1186, 2014.

[6] J. I. Olszewska, "Active contour based optical character recognition for automated scene understanding", *Neurocomputing*, vol. 161, no. 5, pp. 65-71, 2015.

[7] M. I. Razzak, F. Anwar, S. A. Husain, A. Belaid and M. Sher, "HMM and fuzzy logic: A hybrid approach for online Urdu script-based languages' character recognition", *Knowledge-Based Systems*, vol. 23, no. 8, pp. 914-923, 2010.

[8] G. Vamvakas, B. Gatos, N. Stamatopoulos and S. Perantonis, "A Complete Optical Character Recognition Methodology for Historical Documents", IAPR International Workshop on Document Analysis Systems, vol. 1, 2008, pp. 525-532.

[9] H. Fujisawa, "Forty years of research in character and document recognition-and industrial perspective", *Pattern Recognition*, vol. 41, no. 8, pp. 2435-2446, 2008.

[10] Y. Lu, "Machine Printed Character Segmentation - An Overview", *Pattern Recognition*, vol. 28, no. 1, pp. 67-80, 1995.

[11] Y. Lu and M. Shridhar, "Character segmentation in handwritten words - An overview", *Pattern Recognition*, vol. 29, no. 1, pp. 77-96, 1996.

[12] Á. González and L. M. Bergasa, "A text reading algorithm for natural images", Image and Vision Computing, vol. 31, no. 3, pp. 255-274, 2013.

[13] D. Karatzas and A. Antonacopoulos, "Colour text segmentation in web images based on human perception", *Image and Vision Computing*, vol. 25, no. 5, pp. 564-577, 2007.

[14] J. Lim, J. Park and G. G. Medioni, "Text segmentation in color images using tensor voting", *Image and Vision Computing*, vol. 25, no. 5, pp. 671-685, 2007.

[15] J. Min-Chul, S. Yong-Chul and S. N. Srihari, "Machine Printed Character Segmentation Method Using Side Profiles", In Proceedings of the IEEE SMC '99 Conference on Systems, Man and Cybernetics, 1999.

[16] N. Nikolaou, M. Makridis, B. Gatos, N. Stamatopoulos and N. Papamarkos, "Segmentation of historical machine-printed documents using Adaptive Run Length Smoothing and skeleton segmentation paths", *Image and Vision Computing*, vol. 28, no. 4, pp. 590-604, 2010.

[17] H. C. Park, S. Y. Ok, Y. J. Yu and H. G. Cho, "A word extraction algorithm for machine-printed documents using a 3D neighborhood graph model", *International Journal on Document Analysis and Recognition*, vol. 4, no. 2, pp. 115-130, 2001.

[18] L. Zheng, A. H. Hassin and X. Tang, "A new algorithm for machine printed Arabic character segmentation", *Pattern Recognition Letters*, vol. 25, no. 15, pp. 1723-1729, 2004.

[19] A. Choudhary, R. Rishi and S. Ahlawat, "A New Character Segmentation Approach for Off-Line Cursive Handwritten Words", *Procedia Computer Science, First International Conference on Information Technology and Quantitative Management*, vol. 17, pp. 88-95, 2013.

[20] K. Fukushima and T. Imagawa, "Recognition and segmentation of connected characters with selective attention", *Neural Networks*, vol. 6, no. 1, pp. 33-41, 1993.

[21] E. B. Lacerda and C. A. B. Mello, "Segmentation of connected handwritten digits using Self-Organizing Maps", *Expert Systems with Applications*, vol. 40, no. 15, pp. 5867-5877, 2013.

[22] H. Lee and B. Verma, "Binary segmentation algorithm for English cursive handwriting recognition", *Pattern Recognition*, vol. 45, no. 4, pp. 1306-1317, 2012.

[23] J. Oh, S. Joon, S. Sangkuk, L. Ji-Won, K. Nojun and K. Em, "Online recognition of handwritten music symbols", *International Journal on Document Analysis and Recognition (IJDAR)*, pp. 1-11, 2017.

[24] T. Plötz and G. A. Fink, "Markov models for offline handwriting recognition: a survey", *International Journal on Document Analysis and Recognition (IJDAR)*, vol. 12, pp. 269-298, 2009.

[25] A. Rehman and T. Saba, "Performance analysis of character segmentation approach for cursive script recognition on benchmark database", *Digital Signal Processing*, vol. 21, no. 3, pp. 486-490, 2011.

[26] N. Stamatopoulos, B. Gatos, G. Louloudis, U. Pal and A. Alaei, "ICDAR 2013 Handwriting Segmentation Contest", In Proceedings of the 12th International Conference on Document Analysis and Recognition (ICDAR), 2013.

[27] O. Surinta, M. F. Karaaba, L. R. B. Schomaker and M. A. Wiering, "Recognition of handwritten characters using local gradient feature descriptors", *Engineering Applications of Artificial Intelligence*, vol. 45, pp. 405-414, 2015.

[28] J. Tan, J. Lai, C. Wang, W. Wang and X. Zuo, "A new handwritten character segmentation method based on nonlinear clustering", *Neurocomputing*, vol. 89, pp. 213-219, 2012.

[29] M. Younes and Y. Abdellah, "Segmentation of Arabic Handwritten Text to Lines", In Proceedings of the Procedia Computer Science, International Conference on Advanced Wireless Information and Communication Technologies (AWICT 2015), vol. 73, pp. 115-121, 2015.

[30] A. Antonacopoulos and D. Karatzas, "Semantics-based content extraction in typewritten historical documents", In Proceedings of the 8th International Conference on Document Analysis and Recognition (ICDAR '05), 2005, pp. 48-53.

[31] I. Bar-Yosef, A. Mokeichev, K. Kedem, I. Dinstein and U. Ehrlich, "Adaptive shape prior for recognition and variational segmentation of degraded historical characters", *Pattern Recognition*, vol. 42, no. 12, pp. 3348-3354, 2009.

[32] M. R. Gupta, N. P. Jacobson and E. K. Garcia, "OCR binarization and image pre-processing for searching historical documents", *Pattern Recognition*, vol. 40, no. 2, pp. 389-397, 2007.

[33] L. A. F. Fernandes and M. M. Oliveira, "Real-time line detection through an improved Hough transform voting scheme", *Pattern Recognition*, vol. 41, no. 1, pp. 299-314, 2008.

[34] V. Shapiro, "Accuracy of the straight line Hough Transform: The non-voting approach", *Computer Vision and Image Understanding*, vol. 103, no. 1, pp. 1-21, 2006.

[35] C. Singh, N. Bhatia and A. Kaur, "Hough transform based fast skew detection and accurate skew correction methods", *Pattern Recognition*, vol. 41, no. 12, pp. 3528-3546, 2008.

[36] G. Bessho, K. Ejiri and J. F. Cullen, "Fast and accurate skew detection algorithm for a text document or a document with straight lines", In Proc. of the SPIE, vol. 2181, pp. 133-140, 1994.

[37] Y. Cao, S. Wang and H. Li, "Skew detection and correction in document images based on straight-line fitting", *Pattern Recognition Letters*, vol. 24, no. 12, pp. 1871-1879, 2003.

[38] A. Fernández-Caballero, M. T. López and J. C. Castillo, "Display text segmentation after learning best-fitted OCR binarization parameters", *Expert Systems with Applications*, vol. 39, no. 4, pp. 4032-4043, 2012.

[39] H. Z. Eldin, M. A. Elhosseini and H. A. Ali, "Image compression algorithms in wireless multimedia sensor networks: A survey", *Ain Shams Engineering Journal*, vol. 6, no. 2, pp. 481-490, 2015.

[40] J. Mtimet and H. Amiri, "Arabic Textual Image Compression Approach", Procedia Computer Science, vol. 35, pp. 118-126, 2014.

[41] M. Wu, "Genetic algorithm based on discrete wavelet transformation for fractal image compression", Journal of Visual Communication and Image Representation, vol. 25, no. 8, pp. 1835-1841, 2014.

[42] L. Wang, J. Bai, J. Wu and G. Jeon, "Hyperspectral image compression based on lapped transform and Tucker decomposition", *Signal Processing: Image Communication*, vol. 36, pp. 63-69, 2015.

[43] A. M. Rufai, G. Anbarjafari and H. Demirel, "Lossy image compression using singular value decomposition and wavelet difference reduction", *Digital Signal Processing*, vol. 24, pp. 117-123, 2014.

[44] Z. Zheng, J. Zhao, H. Guo, L. Yang, X. Yu and W. Fang, "Character Segmentation System Based on C# Design and Implementation", *Procedia Engineering, International Workshop on Information and Electronics Engineering*, vol. 29, pp. 4073-4078, 2012.

[45] A. Sedighi and M. Vafadust, "A new and robust method for character segmentation and recognition in license plate images", *Expert Systems with Applications*, vol. 38, no. 11, pp. 13497-13504, 2011.

[46] M. Grafmüller and J. Beyerer, "Performance improvement of character recognition in industrial applications using prior knowledge for more reliable segmentation", *Expert Systems with Applications*, vol. 40, no. 17, pp. 6955-6963, 2013.

[47] C. Papaodysseus, P. Rousopoulos, F. Giannopoulos, S. Zannos, D. Arabadjis, M. Panagopoulos, E. Kalfa, C. Blackwell and S. Tracy, "Identifying the writer of ancient inscriptions and Byzantine codices. A novel approach", *Computer Vision and Image Understanding*, vol. 121, pp. 57-73, 2014.

[48] N. B. Venkateswarlu and R. D. Boyle, "New segmentation techniques for document image analysis", *Image and Vision Computing*, vol. 13, no. 7, pp. 573-583, 1995.

[49] J. Li, M. Li, J. Pan, S. Chu and J. F. Roddick, "Gabor-based kernel self-optimization Fisher discriminant for optical character segmentation from text-image-mixed document", *Optik - International Journal for Light and Electron Optics*, vol. 126, no. 21, pp. 3119-3124, 2015.

[50] J. H. Bae, K. C. Jung, J. W. Kim and H. J. Kim, "Segmentation of touching characters using an MLP", *Pattern Recognition Letters*, vol. 19, no. 8, pp. 701-709, 1998.

[51] P. P. Roy, U. Pal, J. Lladós and M. Delalandre, "Multi-oriented touching text character segmentation in graphical documents using dynamic programming", *Pattern Recognition*, vol. 45, no. 5, pp. 1972-1983, 2012.

[52] O. Starostenko, C. Cruz-Perez, F. Uceda-Ponga and V. Alarcon-Aquino, "Breaking text-based CAPTCHAs with variable word and character orientation", *Pattern Recognition*, vol. 48, no. 4, pp. 1101-1112, 2015.

[53] V. Vučković and B. Arizanović, "Efficient character segmentation approach for machine-typed documents", *Expert Systems with Applications*, vol. 80, pp. 210-231, 2017.

[54] V. Vučković and B. Arizanović, "Automatic document skew pre-processor for character segmentation algorithm", *Facta Universitatis: Electronics and Energetics*, vol. 30, no. 4, pp. 611-625, 2017.

[55] V. Vučković, B. Arizanović and S. Le Blond, "Ultra-fast basic geometrical transformations on linear image data structure", *Expert Systems with Applications*, vol. 91, pp. 322-346, 2018.

[56] V. Vučković, B. Arizanović and S. Le Blond, "Generalized N-way iterative scanline fill algorithm for real-time applications", *Journal of Real-Time Image Processing*, vol. 13, no. 4, pp. 1-19, 2018.

[57] B. Žalik, D. Mongus and N. Lukač, "A universal chain code compression method", *Journal of Visual Communication and Image Representation*, vol. 29, pp. 8-15, 2015.

[58] J. Zhu, Z. Wang, R. Zhong and S. Qu, "Dictionary based surveillance image compression", *Journal of Visual Communication and Image Representation*, vol. 31, pp. 225-230, 2015.

[59] A. J. Hussain, D. Al-Jumeily, N. Radi and P. Lisboa, "Hybrid Neural Network Predictive Wavelet Image Compression System", *Neurocomputing*, vol. 151, no. 3, pp. 975-984, 2015.

[60] R. T. Haweel, W. S. El-Kilani and H. H. Ramadan, "Fast approximate DCT with GPU implementation for image compression", *Journal of Visual Communication and Image Representation*, vol. 40 (Part A), pp. 357-365, 2016.

[61] J. Li, "Image Compression - the Mathematics of JPEG 2000", Microsoft Research, Communication Collaboration and Signal Processing.