A Fast Segmentation Algorithm for Bi-Level Image Compression using JBIG2

Dave A. D. Tompkins and Faouzi Kossentini
Signal Processing and Multimedia Group
Department of Electrical and Computer Engineering
University of British Columbia
Vancouver, BC V6T 1Z4 Canada
{davet,faouzi}@ece.ubc.ca http://spmg.ece.ubc.ca

(Invited Paper)

Abstract

The emerging JBIG2 standard allows compliant encoders to achieve very high compression rates on bi-level images, especially when images are properly segmented into regions of line-art, halftones and text. We propose a fast method that is very effective at separating text from non-text regions, even when the regions are non-rectangular or have skew. Our method can also detect regions of reverse-coloured text. In most cases, our method increases the compression performance of the encoder. More importantly, our method can improve encoding speeds considerably, often by an order of magnitude.

1. Introduction

The Joint Bi-Level Image Expert Group (JBIG) has recently completed the committee draft of the JBIG2 standard [1]. JBIG2 is a significant improvement over existing bi-level and facsimile standards, and will have numerous applications beyond facsimile, including document archiving and document transfers over the Internet. For an overview of JBIG2, see [2].

JBIG2 only defines the requirements for decoding a compliant bitstream, leaving the encoder design open and flexible. Different JBIG2 encoders will have varying levels of sophistication, speed, and compression performance.

A JBIG2 bitstream can contain several different region segments that, when combined together, will compose the entire image. JBIG2 supports three basic coding methods for compressing a region segment: Generic, Halftone, and Text. Each method is optimized for a specific type of image.

Generic regions are encoded directly as a bitmap with one of two methods. The first method is known as Modified-Modified-Read (MMR), and is used in the Group 4 (T.6) facsimile standard. The second method is a variation of the template based arithmetic coding used in the JBIG1 standard. The second method (MQ) achieves higher compression and is the most effective method for regions of line-art, figures and graphs.

Halftone regions are encoded as both a grayscale image and a halftone pattern dictionary. A JBIG2 encoder essentially reverses the halftone process so that decoders can re-halftone the grayscale image. Halftone regions can achieve high levels of compression, but except in special circumstances, are lossy. Figure 1 is an example of a halftone region.

Text regions are referred to as *symbol* regions in JBIG2, as the compression works for symbols that can be from any alphabet or be non-text. The symbols themselves are stored in dictionaries and are encoded as generic regions. Symbol regions contain the information required to position a symbol from the dictionary at a specific location in the image.

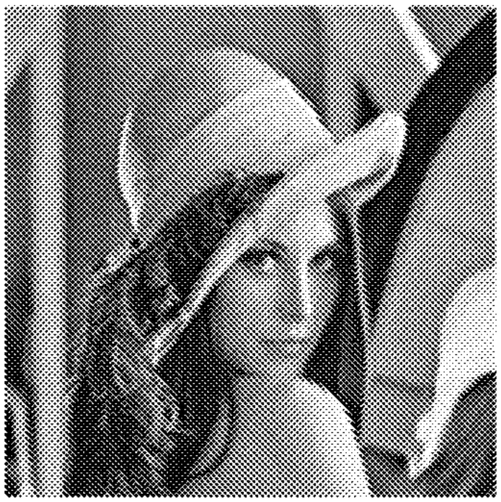


Figure 1. Lena Halftone at 200 dpi

This work was supported by both the Natural Sciences and Engineering Research Council of Canada and Image Power, Inc.

A typical image encountered by a JBIG2 encoder will be a formatted document, containing any combination of text, tables, figures, line-art and halftone regions. To take full advantage of methods available within JBIG2, a sophisticated encoder will segment the image, and then use the best coding method for each region.

Traditionally, document segmentation algorithms have been developed for Optical Character Recognition (OCR) applications. A historical summary of the available methods can be found in [3]. Segmentation methods can be loosely classified as bottom-up, top-down or both. Bottom-up approaches to document segmentation generally start with individual letters on a page, and then based on text-layout conventions, group letters into words, words into paragraphs, and so on. Line-art and halftones are often detected by their size, or their non-text layout. Top-down approaches take advantage of the fact that formatted documents usually have margins surrounding each region. The page can be subdivided into different regions by examining the white-space in the document. Alternatively, top-down methods will use the bit-density or texture of the document to identify and classify regions. Some modern segmentation methods still have problems with skew (text appearing on an angle), non-rectangular regions, reverse-coloured text, and foreign languages. Many methods also require large quantities of training sets to properly identify regions. Unfortunately, even OCR segmentation methods that are relatively fast can be too slow for some bi-level image compression systems.

In this paper, we present a method that quickly segments an image into text and non-text regions. It can detect regions of reverse-coloured text, and is not adversely affected by skew or irregularly shaped regions. Our algorithm decreases encoding time significantly, and almost always improves compression performance.

This paper is organized as follows. Section 2 outlines the challenges of designing a fast segmentation algorithm, and the differences between segmenting for JBIG2 and OCR applications. Section 3 describes our segmentation method, with our experimental results discussed in Section 4 and our conclusions in Section 5.

2. Segmenting for JBIG2

Although the general objectives of JBIG2 segmentation and OCR segmentation are similar, their requirements are quite different. In general, an OCR segmentation method has to be much more accurate. Misinterpreting a block of text as a graphic or not detecting a region of reverse-coloured text may be catastrophic in an OCR application, while in a JBIG2 environment similar errors will only lead to sub-optimal compression performance.

Because they are required to be more accurate, OCR algorithms are allowed to be slower. In general, symbol based bottom-up approaches are faster than top-down strategies as the number of symbols is much smaller than the number of pixels [4]. Most bottom-up strategies require that all of the symbols in the document are extracted. Although symbol extraction can be executed relatively quickly, the analysis of the symbols can be quite costly. For a proper analysis, some sorting or comparison operations must be performed, which can be of complexity $O(n \cdot log_2 n)$ or worse. Images with halftone regions can easily have over 10,000 symbols, which may make even a simple analysis too costly for a JBIG2 application. JBIG2 segmentation methods should avoid a full symbol analysis.

A JBIG2 segmentation method must also consider the consequences of misinterpreting a region type. There will be considerable loss in quality if a lossy halftone coding method is used for a text or line-art region. The generic coding method is designed for a wide variety of region types, and can be used to losslessly compress any region. The symbol coding method can be used for non-text images, and can even achieve higher compression rates than the other methods, although a comparable or slightly poorer compression performance is more common. However, the danger in misinterpreting non-text data as text is not the slightly smaller compression performance, but rather the potentially large difference in execution time.

To achieve the high levels of compression possible in symbol regions, an encoder must perform a detailed analysis on all of the symbols. This symbol analysis is even more complex than a segmentation scheme, and may have a worst-case complexity of $O(n^2)$. If a halftone region with tens of thousands of symbols is misinterpreted as text, the result can be catastrophic. As a result, cautious JBIG2 encoders should be biased towards non-text regions, which is the opposite of the guideline for OCR applications.

3. Segmentation Method

Our objective was to develop a fast method of separating text from non-text regions. We wanted to avoid performing a costly symbol analysis at the segmentation stage, and prevent our symbol region coder from performing a text analysis on a halftone region. We were primarily concerned with lossless coding at facsimile resolutions (200 dpi). We were also interested in a mechanism for detecting regions of reverse-coloured text. And finally, if possible, we wanted to avoid using a method requiring a large set of training data.

Our method requires that an analysis be performed on a reduced image. Instead of reducing the image by downsampling, a block technique is employed [5]. Each pixel in the reduced image corresponds to a *N*x*M* block in the original image. A reduced pixel is white if and only if all of the pixels in the corresponding block are white. Based on this reduction criterion, the reduced image appears dark and smudged, which is why the technique is often called *smearing.* The technique is illustrated in Figure 2.

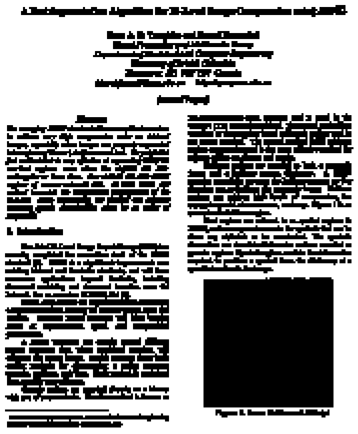


Figure 2. Smearing Reduction Technique
Title page of this paper scanned at 200 dpi, and then reduced with smearing (8x8)

Where a bottom-up approach could have been quite slow on the full image, it is now quite feasible on the reduced image. After reducing the image, we extract all of the symbols using the 8-connected technique described in [4]. We then examine each symbol to determine if it has non-text characteristics. In general, halftone regions and line-art will appear as large black symbols, while text regions will consist of several small symbols. If a symbol is determined to be non-text, then all of the pixels corresponding to the symbol in the original image are removed as a region and encoded separately.

In the reduced image, reverse-coloured text regions will have the same characteristics as halftone regions. When a non-text region is removed from the original image, we test to see if it is reverse-coloured text. To perform this test, we reverse the region and then repeat the analysis we did on the entire image. If the reduced image is still a large black blob, then it is most likely a halftone. Conversely, if there are a large number of small symbols, then it is most likely text. The process of reversing the region, reducing the image and extracting the symbols can all be performed very quickly, and is illustrated in Figure 3.

The segmentation method can be summarized as follows. The original image is reduced, and all of the symbols from the reduced image are extracted. Each symbol corresponds to a region in the original image, and is checked for non-text characteristics. If the symbol is classified as non-text, the corresponding region from the original image is removed. The removed region is reversed, reduced and the symbols are extracted. If there are a large number of symbols the region is encoded as reverse-coloured text. Otherwise, the region is encoded as non-text. After the non-text and reverse-text regions have been removed from the original image, the remaining region is encoded as text. For each of these steps, there are thresholds and parameters that will determine how sensitive the implementation is.

The first parameters to consider are the dimensions of the reduction block, *N* and *M*. For an efficient implementation, it is vital that *N* be a multiple of 8. More than likely, the image data will be stored as 8 horizontal

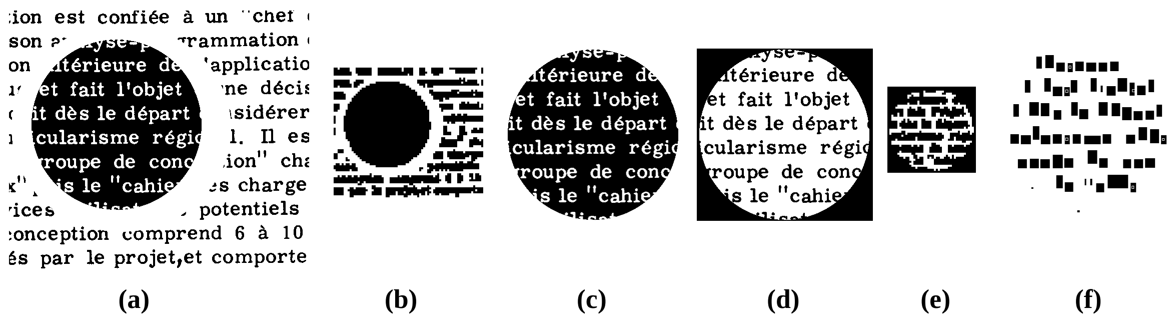


Figure 3. Detecting Reverse-Coloured Text Regions. The original image (a) is reduced (b) and then the non-textual region is extracted (c). The extracted region is then reversed (d) and reduced (e), where it is now obvious that numerous symbols exist, and so symbol coding can be performed (f).

pixels per byte, and restricting *N* to multiples of 8 will allow the reduction and region extraction operations to be performed in a byte-wise manner, significantly improving the speed of the encoder. There is no such restriction on *M*, but by keeping *M* and *N* equal, the method becomes invariant to image orientation, which may be a desirable feature. For our implementation, we were concerned with images at 200 dpi with 10 point (or higher) text and used a block size of 8x8. Figure 2 illustrates the results of the 8x8 block size on the title page of this paper. The most important feature of this reduced image is the halftone area, which appears completely black. In the text region, we can see how the horizontal space between words is maintained, but in some circumstances, adjacent rows of text become connected. This occurs when full height letters (such as h) appear below dangling letters (such as y). If we want to minimize these connections, the value *M* can be reduced, or 4-connected symbol extraction can be used instead of 8-connected. The 8x8 block size is also effective at higher resolutions, but a larger block size can be used to improve the speed even further.

To determine if a symbol in the reduced image corresponds to a non-text region, we consider the weight and the size of the symbol. The weight of a symbol is the number of black pixels it contains. The symbol is classified as non-text if its' weight is above a certain percentage of the weight of the entire reduced image. Additionally, if the area of the symbol's bounding box is above a certain percentage of the entire area, then it is very likely the corresponding region is line-art or a figure. The threshold percentages are somewhat arbitrary, and will determine the sensitivity of the encoder. High percentages may allow small halftone regions to go undetected, while low percentages may misinterpret dense text regions as non-text. The thresholds will also be a reflection of how aggressively the encoder will try to compress the image. We found that the thresholds of 15% worked well for our test images.

To determine if a region contains reverse-coloured text, we examine the number of symbols in the reversed, reduced image. If a sufficient number of symbols are detected, then it will be worthwhile to encode the region as text. The threshold for the number of symbols must be large enough to avoid misinterpreting very dark halftones as text, and small enough to ensure reverse-text regions are not ignored. If we exclude large symbols (using the same criteria we used previously) from the reverse-text region, we can reduce the risk of interpreting a halftone as text, and our threshold can be much smaller. For example, all of the text on the title page of this paper reduces to 156 symbols. In our implementation, we chose a value of 30 symbols, which worked well for most of our test images, but did make some mistakes. It should be noted that the example in Figure 3 uses a small threshold for illustrative purposes.

4. Experimental Results

For our experiments, we used the standard ITU images as our base set. To test the flexibility of our method, we constructed numerous compound documents by combining halftone, line-art, text and reverse-text elements from the base set. We also tested our algorithm on complicated scanned documents. All speeds have been given relative to the speed required to encode the entire image with MMR generic mode, and include any encoder overhead. All of the results presented here are for lossless compression.

Overall, our method is fast and effective. Images containing traditional white space borders are flawlessly segmented. The title page of this paper represents a simple segmentation problem. Table 1 shows the results obtained from compressing the page scanned at a resolution of 200 dpi. From the reduced image of Figure 2, we can see how the halftone image was easily separated. By examining the clustered dots of Figure 1 you can see how the halftone contains twice as many symbols as the text region. Removing the halftone increases the compression performance by 14% and the speed by a factor of 1.75.

Table 1. Encoding Performance for the Title Page of this Paper (200 dpi)

Method	Number of Symbols	Comp. Ratio	Relative Time
Generic - MMR	-	5.5 : 1	1.0
Generic - MQ	-	12.3 : 1	1.3
Symbol Region	7808	13.2 : 1	2.1
Segmented	2686	15.0 : 1	1.2

Although our method can segment the title page of this paper flawlessly, more complicated images can cause some minor problems. Figure 4 illustrates the results of applying our method to the image known as CCITT2 or F17_400. In (c) we can see two errors. Even though the top region contains two different sub-regions, a line-art and a halftone region, it is removed and encoded as one region. In addition, some lines of text are included in the region. Both of these errors occur because there was not enough white-space between the regions. In (d) we can see how part of the line art has remained with the text region. These errors result in a small reduction in the compression performance. Even though the image is complicated, the segmentation works well, and demonstrates how our method handles irregular shapes and text regions with ease.

The encoding results for the image CCITT2 are shown in Table 2. Although there is a small increase in compression, there is a dramatic improvement in speed. By reducing the number of symbols in the text analysis to

4,933 from 42,423, we achieve an increase in encoding speed of almost 20 times.

Table 2. Encoding Performance for the Image CCITT2 (F17_400)

Method	Number of Symbols	Comp. Ratio	Relative Time
Generic - MMR	-	1.6 : 1	1.0
Generic - MQ	-	9.9 : 1	2.3
Symbol Region	42,423	9.0 : 1	41.0
Segmented	4,933	10.4 : 1	2.2

In almost all of the experiments we performed, we obtained a compression gain (0-20%) over a generic or a text-only encoder, and even in situations with a small (<4%) compression loss, there was a large increase in speed (>200%). We also found that completely reverse-coloured text documents were detected automatically, generating large improvements in compression performance.

5. Conclusions

We have proposed a fast algorithm for segmenting an image into text and non-text regions. Our algorithm is specifically designed for a JBIG2 encoder, and takes advantage of the segmenting structure supported in JBIG2. Our algorithm does not have problems with irregular shapes or skew, and has the additional feature of detecting reverse-coloured text regions. Our algorithm is significantly faster than a standard text-based encoder, and generally achieves higher compression rates than straightforward generic and text encoders.

6. References

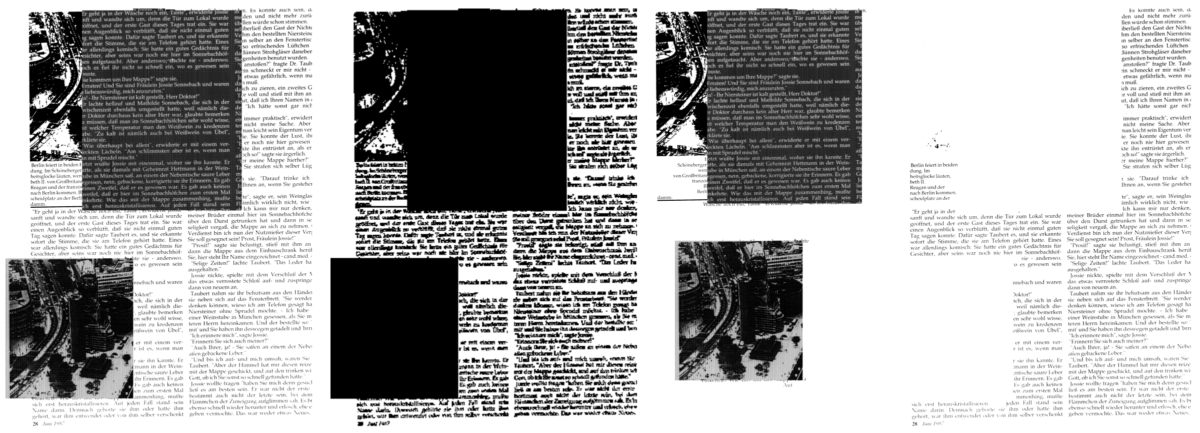
[1] JBIG Committee, *ISO/IEC JTC1/SC29/WG1 (ITU-T SG8) WD14492*. November 1998.

[2] P. Howard, F. Kossentini, B. Martins, S. Forchhammer, W. Rucklidge, F. Ono, "The Emerging JBIG2 Standard", *IEEE Transactions on Circuit and Systems for Video Technology,* Vol. 8, No. 5, September 1998.

[3] A. Jain, B. Yu, "Document Representation and Its Application to Page Decomposition", *IEEE Transactions on Pattern Analysis and Machine Intelligence,* Vol. 20, No. 3, March 1998.

[4] I. H. Witten, A. Moat, and T. C. Bell, *Managing Gigabytes: Compressing and Indexing Documents and Images,* New York: Van Nostrand Reinhold, 1994.

[5] T. Saitoh, T. Pavlidis, "Page Segmentation without Rectangle Assumption", *Proceedings of the 11th International Conference on Pattern Recognition,* Saint Malo, France. 1991.



(a) **(b)** **(c)** **(d)**

Figure 4. Fast Segmentation Algorithm. The Original Image (a) contains 42,423 symbols. The image is reduced (b) and now contains only 717 symbols. Two of the symbols are detected as a non-text and their corresponding regions are removed (c). The remaining image (d) can now be easily compressed as a text region, with only 4,933 symbols. This process increases compression (25%) and significantly reduces execution time (20 times) over a text-only approach.