A GENERAL SEGMENTATION SCHEME FOR DJVU DOCUMENT COMPRESSION

PATRICK HAFFNER, LÉON BOTTOU and YANN LECUN
AT&T Labs-Research
200 Laurel Ave, Middletown NJ 07748
{haffner,leonb,yann}@research.att.com

and

LUC VINCENT
LizardTech, Inc
821 2nd Avenue, Seattle WA 98104
lvincent@lizardtech.com

[Proceedings of the *International Symposium on Mathematical Morphology (ISMM)*, Sydney, Australia, April 2002, *CSIRO Publications*]

Abstract.

We describe the "DjVu" (Déjà Vu) technology: an efficient document image compression methodology, a file format, and a delivery platform that together, enable instant access to high quality documents from essentially any platform, over any connection. Originally developed for scanned color documents, it was recently expanded to electronic documents, so DjVu has now truly become a universal document interchange format.

With DjVu, a color magazine page scanned at 300dpi typically occupies between 40KB and 80KB, i.e. approximately 5 to 10 times smaller than JPEG for a similar level of readability (the typical compression ratio is 500:1). Converting electronic documents to DjVu also offers substantial advantages, as described in the paper. The technology relies on a classification of each pixel as either foreground (text, drawing) or background (pictures, paper texture and color), thereby producing a segmentation into layers that are compressed separately. The novel contribution of this paper is a unified approach for segmentation of scanned or electronic documents, using a rigorous approach based on the Minimum Description Length (MDL) principle.

The foreground layer is compressed using a pattern matching technique taking advantage of the similarities between character shapes. A progressive, wavelet-based compression technique, combined with a masking algorithm, is then used to compress the background image at lower resolution, while minimizing the number of bits spent on the pixels that are otherwise covered by foreground pixels. Encoders, decoders, and real-time, memory efficient plug-ins for various web browsers are available for all the major platforms.

Key words: Arithmetic Coding, DjVu, Document Interchange, Image Compression, Minimum Description Length, Segmentation, Wavelets, 2D Hidden Markov Models.

1. Introduction

With the generalized use of the Internet and the declining cost of scanning and storage hardware, documents are increasingly archived, communicated, and manipulated in digital form rather than in paper form. The growing need for instant access to information makes the computer screen the preferred display

medium.

Compression technology for bitonal (black and white) document images has a long history (see [22] and references therein). It is the basis of a large and rapidly growing industry with widely accepted standards (Group 3, MMR/Group 4), and less popular and emerging standards (JBIG1, JBIG2).

The last few years have seen a growing demand for a technology that could handle *color* documents in an effective manner. Such applications as online digital libraries with ancient or historical documents, online catalogs for e-commerce sites, online publishing, forms processing, and scientific publication, are in need of an efficient compression technique for color documents. The availability of low-cost, high quality color scanners, the recent emergence of high-speed production color scanners, and the appearance of ultra high resolution digital cameras open the door to such applications.

Standard color image compression algorithms are inadequate for such applications because they produce excessively large files if one wants to preserve the readability of the text. Compressed with JPEG, a color image of a typical magazine page scanned at 100dpi (dots per inch) is around 100-200KB, and is barely readable. The same page at 300dpi is of acceptable quality, but occupies 300-600 KB. These sizes are impractical for online document browsing, even with broadband connections.

Preserving the readability of the text and the sharpness of line art requires high resolution and efficient coding of sharp edges (typically 300-400dpi). On the other hand, preserving the appearance of continuous-tone images and background paper textures does not require as high a resolution (typically 100dpi). An obvious way to take advantage of this is to segment these elements into separate layers. The foreground layer would contain the text and line drawings, while the background layer would contain continuous-tone pictures and background textures. This multi-layer raster representation is key element in various recent document interchange products and standards [21, 2, 12, 14].

The separation method brings another considerable advantage. Since the text layer is separated, it can be stored in a chunk at the beginning of the image file and decoded by the viewer as soon as it arrives in the client machine.

Overall, the requirements for an acceptable user experience are as follows: The text should appear on the screen after only a few seconds delay. This means that the text layer must fit within 20-40KB assuming a 56Kb/sec connection. The pictures, and backgrounds would appear next, improving the image quality as more bits arrive. The overall size of the file should be of the order of 50 to 100 KB to keep the overall transmission time and storage requirements within reasonable bounds.

Large images are also problematic during the decompression process. A magazine-size page at 300dpi is 3300 pixels high and 2500 pixels wide and occupies 25 MB of memory in uncompressed form, more memory than many devices (such as low-end PCs or hand-held devices) can comfortably handle. A practical document image viewer should therefore keep the image in a compressed form in the memory of the machine and only decompress on-demand the pixels that are being displayed on the screen.

The DjVu document image compression technique [2] described in this paper addresses all the above mentioned problems. With DjVu, pages scanned at 300dpi in full color can be compressed down to 30 to 80 KB files from 25 MB originals with excellent quality. This puts the size of high-quality scanned pages in the same order of magnitude as a typical HTML page (which are around 100KB on average). DjVu pages are displayed progressively within a web browser window through a plug-in, which allows easy panning and zooming of very large images without generating the fully decoded 25MB image. This is made possible by storing partially decoded images in a data structure that typically occupies 2MB, from which the pixels actually displayed on the screen can be decoded on the fly. The DjVu encoder was designed to be as generic as possible: the only information it requires about the document is its scanning resolution. This puts an extraordinary constraint on the segmentation algorithms used to obtain the foreground/background separation.

DjVu was initially designed for scanned documents. However, it soon became apparent that DjVu could also significantly improve the compression rate and speed of rendering of electronic documents such as PostScript, PDF or MSWord, as these document description languages are generally slow to render, may produce very large files and are often platform dependent. In this case, one would like to compress the document produced using computerized methods that do not rely on an expensive-to-compute, pixel based representation of the document, but rather on an existing *structured page information* [3]. A text processing software, for instance, represents a document using high level objects such as text, fonts, colors, embedded images, etc.

The technologies used in DjVu can be divided in three main groups: segmentation, compression and distribution/viewing. This paper gives an overview of the DjVu compression technologies, pieces of which is described in more detail in [2]: Section 2 explains the general principles used in DjVu compression and decompression. A recent description of distribution/viewing technologies is available in [11].

The novel contribution of this paper is an extended and unified description of the principles and algorithms that drive the DjVu segmentation [7, 3]. Section 3 shows how the Minimum Description Length (MDL) principle can be turned into a practical and universal segmentation tool. Its application to the optimization of the segmentation in electronic documents is shown in Section 4. How the same MDL principle defines foreground/background segmentation algorithms for scanned documents is illustrated in Section 5.

2. The DjVu Compression Method

The basic idea behind DjVu is to separate the text from the background and pictures and to use different techniques to compress each of those components. Traditional methods are either designed to compress natural images with few edges (JPEG), or to compress black and white document images almost entirely composed of sharp edges (Group 3, MMR/Group 4, and JBIG1). The DjVu technique combines the best of both approaches. A foreground/background

separation algorithm segments images into separately compressed layers:

- The **background layer** contains a low resolution (typically 100 dpi.) background image representing details that are best encoded using a continuous tone technique. This usually includes the document background and the pictures.
- The **foreground layer** contains a high resolution bitonal *mask image* (typically 300 dpi) that accurately defines the shape of details with sharp edges such as text and line-art. The color information is either encoded as a solid color per connected component in the mask, or as a very low resolution *foreground image* (typically 25 dpi) whose colors are applied using the mask as a stencil.

A pixel in the decoded image is constructed as follows: if the corresponding pixel in the mask image is 0, the output pixel takes the value of the corresponding pixel in the appropriately up-sampled background image. If the mask pixel is 1, the pixel color is taken from the foreground color.

The mask image is encoded with a new bi-level image compression algorithm called JB2 or *DjVuBitonal*. It is a variation on AT&T's proposal to the emerging JBIG2 standard. The basic idea of JB2 is to locate individual shapes on the page (such as characters), and use a shape clustering algorithm to find similarities between shapes [1, 8]. Shapes that are representative of each cluster (or in a cluster by themselves) are coded as individual bitmaps with a method similar to JBIG1. A given pixel is coded with arithmetic coding using previously coded and neighboring pixels as a context (or predictor). Other shapes in a cluster are coded using the cluster prototype as a context for the arithmetic coder, thereby greatly reducing the required number of bits since shapes in a same cluster have many pixels in common. In lossy mode, shapes that are sufficiently similar to the cluster prototype may be substituted by the prototype. Another chunk of bits contains a list of shape indices together with the position at which they should be painted on the page. In cases where the document has multiple pages, it is advantageous to build a library of shapes that are common to all the pages, in addition to the library of shapes specific to each page. JB2 uses a fast incremental technique to extract common shape libraries across pages.

For the background and foreground images, DjVu uses a progressive, wavelet-based compression algorithm called IW44 or *DjVuPhoto*. IW44 offers many key advantages over existing continuous-tone image compression methods.

- It uses the fast "lifting" method to perform the wavelet transform [19].
- The wavelet transform can be performed entirely without multiplication operations, relying exclusively on shifts and additions, thereby greatly reducing the computational requirements.
- The intermediate image representation in memory is designed to allow progressive refinement of the wavelet coefficients while occupying a memory footprint proportional to the number of non-zero coefficients, and not to the number of pixels.
- The data structure allows efficient on-the-fly rendering of any sub-image at any prescribed resolution, in a time proportional to the number of rendered

Document title	no compression	GIF compression	JPEG compression	**DjVu** compression
hobby p15	25MB	1562	469	**58**
medical dict.	16MB	1395	536	**110**
time zone	9MB	576	249	**36**
cookbook	12MB	1000	280	**52**
hobby p17	24MB	1595	482	**52**
U.S. Constit.	31MB	2538	604	**134**
hobby p2	24MB	1213	383	**68**
ATT Olympic	24MB	955	285	**41**

TABLE I

Compressed files sizes (in KB) for 8 documents using the following compression methods: no compression, GIF on the 150dpi image, JPEG on the 300dpi image with quality 20 and DjVu with 300dpi mask, and 100dpi backgrounds.

pixels (not image pixels). This last feature is particularly useful for efficient panning and zooming through large images.

A masking technique based on multiscale successive projections [5] is used to avoid spending bits to code areas of the background that are covered by foreground characters or drawings.

Both JB2 and IW44 rely on a fast adaptive arithmetic coder called the ZP-coder [4]. The arithmetic coder uses a *context* formed by previously transmitted neighboring pixels to predict the value of the pixel under consideration, and to code its value using a nearly optimal amount of information (within a few percent of the Shannon limit). The ZP-coder is faster than other approximate binary arithmetic coders.

According to an independent study by Inglis [9] with bitonal scanned documents, JB2 in lossless mode achieves an average compression ratio of 26.5, which can be compared to 13.5 for MMR/Group 4, and 19.4 for JBIG1. The lossy mode brings another factor of 2 to 3 over lossless, with more improvement on mostly textual images, and less on images with pictures and in low-quality images.

The performance of IW44 is typical of the latest wavelet-based compression methods which form the basis of JPEG2000 [17, 20], but it has been heavily optimized to minimize the decoding time and memory footprint rather than maximize the compression ratio. The size of an IW44 file is generally 30% to 50% less than a JPEG with the same signal to noise ratio, but the biggest advantage over JPEG is the progressive refinement. Comparisons between IW44 and JPEG are available at `www.djvuzone.org/djvu/photos/jpgvsdjvu01` and in [11].

For *scanned* color documents, the full multilayer DjVu method is known as *DjVuDocument*. It can reach compression ratios of 500:1 to 1000:1. As shown in Table I, typical, letter-size color documents at 300dpi (catalog or magazine page) compressed with DjVu occupy 30KB to 80KB. Occasionally, larger documents, or document with lots of highly detailed pictures or handwriting may occupy 80 to 140KB. This is 5 to 10 times better than JPEG for a similar

Document	Type	Pages	PS/PDF	**PS2DjVu**
mask.ps.gz	LaTeX	10	400K	**78K (23s)**
paper2web.pdf	Book	327	4230K	**3424K (1235s)**
sgi.pdf	Flyer	4	484K	**106K (27)**
stanford.pdf	Map	1	412K	**170K (30s)**

TABLE II
File sizes and compression times for four different documents. Results for the proposed segmenter at 300 dpi are given in the last column. These results can be compared with the initial `ps.gz` or `pdf` file sizes.

level of legibility of the text. DjVu is particularly good at reproducing ancient documents with textured paper. Results and examples are available from the DjVu digital library at `www.djvuzone.org/djvu`. More examples are available from many commercial and non-commercial users of DjVu on the Internet.

For *electronic* color documents, the full DjVu method with optimization of the existing segmentation is known as *DjVuDigital*. Current implementations include a Ghostscript driver [6], which enables the creation of DjVu from PostScript and PDF, and a Windows Virtual Printer Driver, which lets Windows users create DjVu documents from virtually any application [13]. Table II summarizes the results obtained on four very different documents at 300 dpi. Though important variations can be observed depending on the type of electronic document converted to DjVu (PDF, Microsoft Word, PostScript, etc.), file sizes generally compare favorably. As a rule of thumb, one can expect a file size reduction of a factor 2 to 50, remarkable considering that the originals were electronic. The resulting DjVu files are a completely portable version of their original electronic counterparts: they do not depend on any system-specific objects such as fonts; like all DjVu files, they are very efficient to transmit, display, store, and they can be viewed on any platform. This makes DjVu an excellent document interchange format. Moreover, from a user experience viewpoint, DjVu enables a much faster zooming and panning.

3. MDL-based Segmentation

Segmentation techniques for documents have traditionally been based on the concept of adaptive thresholding. From the early stage, DjVu segmentation was based in the idea of of color clustering [2], which comes for computer vision. As the DjVu approach was generalized to handle any type of scanned document, the need for a "principled" approach became clear. It had to be as universal as possible and avoid heuristics that would have to be tuned on hundreds of different kinds of documents.

The Minimum Description Length (MDL) principle [16] drives the segmentation in DjVu. Each decision is made to minimize the overall coding cost. This coding cost is the sum of the number of bits necessary to encode the image (*the encoding bit cost*) and the number of bits necessary to encode the discrepancy

between the encoded image and the original image (*the discrepancy bit cost*). In the most general case, segmenting an image consists of assigning foreground or background labels to each pixel. If N is the number of pixels, segmentation is a choice among 2^N possible decisions. Comparing the true bit costs for each decision would be very expensive as it would require coding both the foreground and the background layers and measuring the resulting file size and quality *for each possible decision.*

A feasible segmentation relies on the following factorization strategy:

Component extraction Identify *connected components*, or groups of pixels, that all belong either to the background or the foreground.

Filtering Decide, for each component, whether it is preferable to code it as foreground or as background. Rather than performing a full image compression, we should rely on bit cost estimates derived from simple measurements on the component and its vicinity.

In the filtering step, two competing strategies are associated with different *encoding schemes.* The preferred encoding scheme is the one that yields the lowest overall coding cost. Like most MDL approaches used for segmentation [10], the motivation is to obtain a system with very few parameters to hand-tune. However, the MDL principle is used here to make only *one* decision per component, thus avoiding the time consuming minimization of a complex objective function.

To code the component as part of the "smooth" background only requires a background color encoding scheme with cost [1] $C^{\text{col}}_{\text{bg}}$. To code the component as a piece of foreground that sticks out of the background requires a foreground color encoding scheme ($C^{\text{col}}_{\text{fg}}$), a mask encoding scheme ($C^{\text{bin}}_{\text{mask}}$) and a background color encoding scheme ($C^{\text{col}}_{\text{bg|fg}}$, this scheme does not spend bits on the already encoded foreground).

The component is classified as foreground if

$$C^{\text{col}}_{\text{fg}} + C^{\text{col}}_{\text{bg|fg}} + C^{\text{bin}}_{\text{mask}} - C^{\text{col}}_{\text{bg}} \leq \theta \tag{1}$$

and as background otherwise.

- The mask encoding bit cost $C^{\text{bin}}_{\text{mask}}$ is roughly proportional to the perimeter of the component. This mask encoding scheme is also assumed to be lossless (in DjVu, loss in the bitonal coding is hardly noticeable): there is no mask discrepancy bit cost. Adding a position cost computed in the same fashion as in the JB2 algorithm can further refine this cost.
- As the foreground color is assumed to be uniform, the encoding part of $C^{\text{col}}_{\text{fg}}$, which is only the value of this color, can be assumed to be negligible, while the discrepancy part of $C^{\text{col}}_{\text{fg}}$ accounts for the difference between the foreground pixels and their average.

[1] A cost with a *col* superscript corresponds to a scheme for encoding pixel colors, while a cost with a *bin* superscript corresponds to a scheme for encoding pixel binary mask values.

– For the background, to avoid a computation over the whole image, we must compute the cost difference $C_{\text{bg}}^{\text{col}} - C_{\text{bg|fg}}^{\text{col}}$. The background encoding scheme is optimized for continuous tone images and typically requires more bits to encode sharp transitions such as the component edges. For $C_{\text{bg|fg}}^{\text{col}}$, the encoding cost for some of these sharp edges has already been paid for in the mask cost: because the IW44 wavelet encoder is able to reduce the bit rate allocated to masked parts of the background [5], there is no need to waste bits for encoding edges that arise from occlusions by the foreground and are already defined accurately by the boundary of the foreground mask. Therefore, the background cost difference lies mostly on area occupied by the candidate component and its edges. Section 4 will show a simple approximation of this cost difference based on measures on the perimeter while Section 5 will propose a more complex technique.

The next two sections examine the application of this MDL principle to optimize the compression for both electronic and scanned documents.

4. Electronic Document Segmentation

An electronic document is represented using high level objects such as text, fonts, colors, embedded images, etc. These objects also correspond to a list of drawing operations. In this section, we see how our MDL approach allows to further process this list into a foreground/background segmentation that optimizes segmentation: each component of the list is affected to either the foreground or the background layer according to a MDL principle based on perimeter ratios. This MDL-principled approach only requires one threshold and considerably reduces the need for tuning. This algorithm has been published in more detail in [3].

4.1. Extraction of overlapping components

Structured page information for electronic documents come in a large variety of file formats such as the MSWord `doc` files, PDF files, or PostScript files. Printing such files converts the structured information into a list of drawing operations such as "fill a rectangle", "draw a line", "draw an image" or "draw a piece of text". This can be interpreted as a list of predefined foreground components with a drawing order as they may overlap. However, *overlapping* components may cause problems for layered document raster formats. Even when, as this is done with the most recent versions of DjVu, the foreground can be represented as a list of components, overlapping components may not be desirable. The quality of the user experience with DjVu depends critically on the speed of browsing, zooming and panning through a document. Our experience in implementing a browser plug-in showed that non-overlapping foreground components enabled faster subsampling and rendering algorithms.

To remove overlapping, several *naive* strategies are possible, e.g. to place all the text into the foreground and all the rest into the background. But the optimal approach would be, from this list of foreground components, to select only those (or part of those) which result in the best compression, and this is

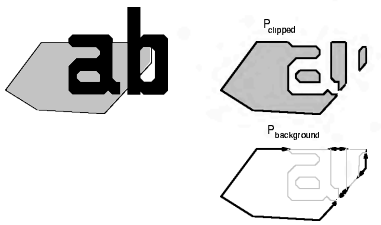


Fig. 1. A polygonal component is partially occluded by two letters. The perimeter of its visible part is named P_{clipped}. The length of the perimeter segments that do not result from the occlusion is named $P_{\text{background}}$.

where our MDL driven approach comes into play, in a rather simple way.

4.2. Filtering means comparing component perimeters

Our goal is a segmentation algorithm that decides whether each monochromatic component belongs to the foreground or the background, based on the MDL principle. The gray shape that appears in Figure 1 under the letters may be classified as foreground or background. The competing encoding schemes are as follows.

Foreground We need to add the possibility of occlusion to the general algorithm presented in the Section 3. The foreground mask bit cost $C^{\text{bin}}_{\text{mask}}$ is proportional to the perimeter P_{clipped} of the visible part of the component (i.e. after removing the component shape pixels that are occluded by other page components). The foreground color bit cost, which would correspond to any color variation, can be ignored in electronic documents.

Background We can reasonably assume that, in our electronic document, all significant color changes happen at component boundaries. As a consequence, both the background encoding and discrepancy bit costs are roughly proportional to the length $P_{\text{background}}$ of the perimeter segments that do not result from occlusions by foreground components. Furthermore, the proportionality coefficient depends on the color differences along the object boundary.

The proposed classification algorithm proceeds in a greedy way. First we prepare two empty bitmaps ($\mathcal{F}$ and $\mathcal{B}$) representing the pixels currently classified as foreground and background. Then we perform the following operations on every monochromatic component starting from the topmost component and proceeding towards the bottommost component.

i) Determine the part of the component shape that is occluded by background components drawn above the current component (by computing the intersection of the component shape and the current background $\mathcal{B}$). Remove these occluded pixels from the component shape.

ii) Determine the part of the component shape that is occluded by foreground components drawn above the current component (by computing the inter-

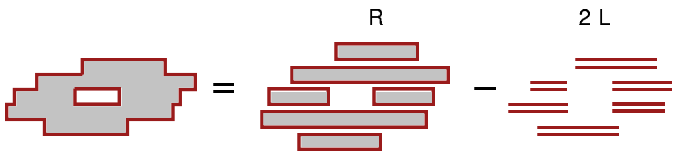


Fig. 2. The perimeter of a run-length encoded bitmap is easily computed by adding the perimeters of each horizontal run and subtracting twice the length of the contact segments between runs located on adjacent rows.

section of the component shape and the current foreground $\mathcal{F}$). Remove these occluded pixels from the component shape.

iii) The component shape now contains only the visible pixels of the component. Compute its perimeter P_{clipped}. Compute the length $P_{\text{background}}$ of the perimeter segments that do not result from occlusions by foreground components. Estimate the color difference δ along the perimeter segments that do not result from occlusions by foreground components.

iv) Enforcing Eq.(1) with $\theta = 0$ gives the final test. Let T be the ratio of the foreground and background proportionality coefficients (between the bit costs and the perimeters). If the ratio $\delta P_{\text{background}}/P_{\text{clipped}}$ is smaller than T, the component is deemed a background component and the clipped component shape is added to bitmap $\mathcal{B}$. Otherwise the component is deemed a foreground component and the clipped component shape is added to bitmap $\mathcal{F}$.

4.3. Implementation Issues

The proposed algorithm makes a large number of boolean operations between bitmaps (i.e. $\mathcal{B}$, $\mathcal{F}$ and the component shapes). Our implementation represents these bitmaps using run-length encoding and performs boolean operations in time proportional to the number of runs on the relevant scan lines.

The proposed algorithm also requires the quantities P_{clipped} and $P_{\text{background}}$. These quantities can be computed as a side effect of processing the component occlusions in steps (i) and (ii).

- Quantity P_{clipped} is simply the perimeter of the clipped component shape computed at step (ii) of the algorithm. The perimeter of a run-length encoded bitmap is also computed in linear time by making a single pass on the bitmap runs and simultaneously computing the sum R of the run perimeters and the sum L of the lengths of the contact segments between runs located on adjacent scan lines. As shown in Figure 2, the bitmap perimeter P is equal to $R - 2L$.
- Quantity $P_{\text{background}}$ is easily computed using the relationship illustrated in Figure 3. It is sufficient to compute the perimeter $P_{\text{unclipped}}$ of the component shape after step (i) and the perimeter $P_{\text{occlusion}}$ of the bitmap representing the component shape pixels occluded by foreground objects. This bitmap is computed during step (ii) of the algorithm. The desired quantity $P_{\text{background}} = \frac{P_{\text{unclipped}} + P_{\text{clipped}} - P_{\text{occlusion}}}{2}$

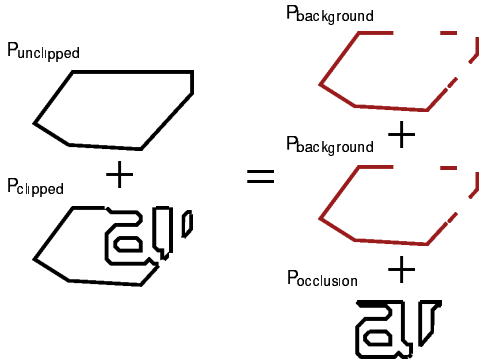


Fig. 3. The sum of the clipped and unclipped perimeter is equal to twice the background contour plus the perimeter of the occluded part of the unclipped object. This equality provides a convenient way to compute the length of the background contour.

We implemented the proposed algorithm both as a Ghostscript [6] driver and a Windows virtual printer driver [13]. Empirical results show that this algorithm provides an extremely robust and accurate segmentation. It does not rely on the textual or non-textual nature of each image component. It is able to handle documents with complex layouts such as geographical maps. Using this segmenter with the DjVu system yields very high quality images whose size is smaller than the size of the initial compressed PostScript or PDF document.

5. Scanned Document Segmentation

This section shows how both the component extraction and the filtering steps in the scanned document segmentation are driven by the MDL principle.

5.1. Component Extraction

Unlike electronic documents, candidate components are not available in scanned documents. To obtain these components, many thresholding or clustering algorithms are available. However, most of them have been developed with one type of document in mind.

To get the most general possible algorithm, we would like to follow our MDL guidelines. However, it is unrealistic to represent a pixel-level foreground/background classification as a discrete decision process. If we hypothesize that the pixels are generated by a parametric model, the MDL principle states that we should look for the model which minimizes the sum of the costs for encoding the model parameters and the discrepancy between the true image and the model-generated image.

As the simplest parametric model for the foreground/background separation, we chose a two-dimensional[2] Hidden Markov Model (HMM)[15] with two

[2] Equivalent to a causal Markov Random field

states (foreground and background) and single Gaussian distributions. In order to minimize the discrepancy bit cost, parameters are optimized for Maximum Likelihood Estimation (MLE). The encoding bit cost is a Bayesian prior distribution over the parameters. MLE is only applied over local regions of the image(typically rectangular), which are only a few hundred pixels. We found these regions large enough for a robust ML estimate of the Gaussian means, but too small to estimate the transition probabilities, which must be entirely based on some prior knowledge. As a consequence, the "consolidated" transition probability [3] between the foreground state and the background state is regarded as a prior parameter, the choice of which will be discussed in Section 5.5. Gaussian covariances are also reduced to a single prior parameter, which is the ratio between the foreground and the background covariances.

This initial foreground separation, dubbed the *ForeBack* algorithm, is designed to prefer *over-segmentation*, so that no characters are dropped. As a consequence, it may erroneously put highly contrasted pieces of photographs in the foreground. After foreground pixels have been separated, they are grouped into connected components.

5.2. Filtering

For each foreground component found by the ForeBack algorithm, the problem is to decide whether it is preferable to *actually* code it as foreground or as background.

As in Section 4, we assume the mask encoding bit cost to be roughly proportional to the perimeter of the component, but there is no possibility of overlapping. Biasing the encoding scheme to favor horizontal and vertical boundaries improves the performance.

As mentioned in Section 3, the foreground encoding scheme assumes the color of a component to be uniform. The error with respect to the uniform color can be coded with Gaussian or Laplacian distributions; the number of discrepancy bits would be proportional to the L2 (Euclidean) or L1 (Manhattan) distances; the mean value for the foreground color that minimizes this discrepancy cost would be the average or the median respectively.

The background encoding scheme assumes that the color of a pixel is the average of the colors of the closest background pixels that can be found up and to the left. There is no encoding bit cost associated with this very simple Delta model. As with the foreground, the discrepancy bit cost can be measured using either the L1 or the L2 distances between the predicted and the actual pixel colors. It is critical to use the same distance and the same proportionality coefficients for both the foreground and the background discrepancy costs.

Note that the background encoding scheme allows a slow drift in the color, whereas the foreground encoding scheme assumes the color to be constant in a connected component. This difference is critical to break the symmetry between the foreground and the background.

Section 3 shows that only a difference in background costs is needed, which

[3] As we use 2x2 pixel cliques, there are in fact 16 transition patterns, whose probabilities are expressed as a function of one common transition probability.

can be restricted to the region below and immediately surrounding the component. The ForeBack extraction algorithm makes overlapping impossible, and touching another component extremely unlikely. However, simply measuring the perimeter would not give here any accurate estimate of the cost. There can be significant color variations inside the component, and in the area immediately surrounding it: we must measure the differences in the background discrepancy costs for the color of each pixel in these areas.

5.3. Implementation Issues

The efficient implementation of simple principles can be extremely complex, and is beyond the scope of this paper, especially when the constraint is to segment 8 million pixels in a time that is of the order of one second. We will just mention two implementation issues that call for fast morphological operations [18].

The fact that the candidate component boundaries are likely to be blurred significantly complicates the filtering algorithm. On a typical 300dpi scanned document, because of a combination of losses caused by printing, decay of the ink and scanning, there is a 1 to 4 pixel border between the foreground and the background in which the luminance is blurred and the color may be unreliable. If we want to encode the image exactly, not only do we need to encode the position of the border, but also these 50 lumen variations in the foreground and the background discrepancy costs! The total coding cost may be larger than if we just encoded, with wavelets, the whole image as background. It is therefore necessary for the encoding process to ignore these boundaries.

- For the foreground color, we compute the average color over an *eroded* version of the component shape.
- The background color is encoded using the masked wavelet algorithm, with a mask which is a *dilated* version of the component shape.

Depending on the document, the width of the border may vary and it is necessary to search for the border width that will yield the lowest bit cost for the foreground/background representation. In the current DjVu segmenter, this resulted in the development of fast morphological operations (dilation and erosion) on run-encoded shapes.

Another complication arises when the foreground component combines two valid foreground components of different color that happen to be connected. As the overall color is not uniform, this component will be rejected in the background, unless we are able to break it. A typical example would be, on a map, a red road that intersects a blue river. The problem is to examine a large number of breaking hypothesis in a component that can comprises hundreds of thousands of pixels: to do that in real time would require even faster morphological operations.

5.4. Subjective Evaluation of the Foreground/Background Separation

Before tuning our model, we must establish the following "proof of concept": at the same compression rate, a document using a foreground/background seg-

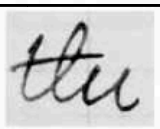
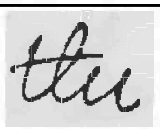
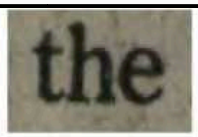
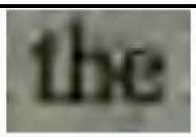
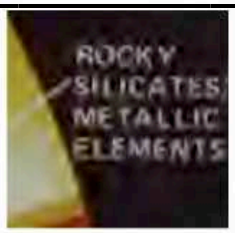
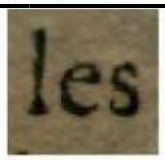
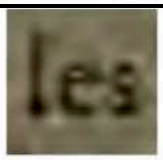
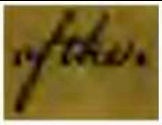
Image Description	Raw image detail	IW44 only size=DjVu	DjVu
Magazine Add % image= 56	20640K	61K 338:1	52K 396:1
Brattain Notebook % image= 22	 9534K	20K 476:1	 19K 501:1
Newspaper Article % image= 50	 12990K	 42K 309:1	40K 324:1
Cross-Section of Jupiter % image= 73	24405K	 52K 469:1	47K 519:1
XVIIIth Century book % image= 45	 12128K	 39K 310:1	37K 327:1
US First Amend-ment % image= 30	31059K	 78K 398:1	73K 425:1

Fig. 4. *Influence of the segmentation on the compressed image quality.* `Raw` *applies no compression.* `IW44` *applies wavelet compression to a unique 300dpi layer, with the quality parameter chosen to obtain the same size as multi=layer DjVu. The "% image" value corresponds to the percentage of bits required to code for the background. Each column shows the same selected detail of the image. To make selection as objective as possible, when possible, the first occurrence of the word "the" was chosen. The two numbers under each image are the file size in kilobytes and the compression ratio (with respect to the raw file size).*

mentation has a better subjective quality than a document compressed using a "smooth" encoding scheme (wavelet or JPEG) only.

Traditionally, as an "objective" measure of the quality of the compressed image, researchers use the so called Signal to Noise Ratio (SNR). In the case of foreground/background encoded documents, we ignore the borders. Over a width of several pixels we allow the encoded pixel value to diverge, sometimes a lot, from the original pixel value. While we will see that the human eye often finds DjVu foreground/background transitions "better" than the original, this discrepancy causes a very large SNR. As a consequence, an evaluation of the foreground/background separation algorithm is, by essence, subjective.

The proof of concept is obtained by examining images compressed with or without segmentation. We have selected six images representing typical color documents. These images have been scanned at 300dpi and 24 bits/pixel from a variety of sources. Figure 4 compares DjVu (with foreground/background separation) with the IW44 encoder alone (column "IW44"). It clearly demonstrates the benefits of such a separation on the textual components of the image and proves that to represent the textual components of a color document as a bi-level image is critical for DjVu performance. It also shows that to replace blurred edges with sharp edges improves readability and does not compromise the document integrity. The experimental digital library which is available at `www.djvuzone.org` shows the performance of the algorithm on a large variety of document types (it is important to stress that no hand-correction was applied to any of these images).

5.5. Tuning the Segmenter

The previous section showed that our foreground and background encoding schemes, based on what kind of loss is acceptable for the human eye, work reasonably well on specific documents. Our encoding schemes are associated with data-generating models, whose parameters are for the most part estimated on the document, apart from a few *meta*-parameters that represent our prior knowledge. To establish fully the efficiency of our MDL approach, we must verify that the number of meta-parameters to tune is minimal and that these meta-parameters are reasonably document-type independent. Most of the meta-parameters determine the prior distribution of the MLE parameters in the HMM that drives the ForeBack separation algorithm.

- **Prior for the Gaussian means** These means are obtained by MLE on local image regions, with a Bayesian prior distribution determined by the surrounding local regions. The only meta-parameter that can be chosen is the weight that is applied to this prior.
- **ForeBack transition probability** We saw that the transition probability between the foreground and the background states could not be robustly estimated on the data, so that we have to set it as a prior. As a consequence, the only meta-parameter we need is this transition probability itself. It is critical: the proportion of foreground decreases monotonously as this transition probability decreases. A low probability removes very small marks, smooth out edges and remove halftoning. When it reaches

zero, no foreground is left.

- **ForeBack threshold** This parameter corresponds to the ratio between the foreground and the background standard deviations used for the Gaussian of our Hidden Markov Model. Its normal setting breaks the symmetry between foreground and background by giving the background a larger variance.
- **ForeBack MLE block size** This is the size of the blocks (typically 16x16) on which separate foreground and background MLEs are performed. Small blocks will allow for fast changes in the foreground color; large blocks will make the estimate more robust.
- **Filter threshold** This threshold, which corresponds to θ in Eq.(1), acts as a bias in the comparison between the foreground and the background encoding schemes used in the filtering step.
- **Perimeter coefficient** In the filtering step, this is the multiplicative coefficient applied to perimeters to turn them into bit costs.

The user has access to these parameters in the DjVu encoding software[4]. However, it is necessary to provide some reasonable default options that will work for most scanned documents.

The semi-automatic tuning procedure we used is illustrated in the following example, where a reasonable value is obtained for the Foreground/Background transition probability. Figure 5 shows the impact of this transition probability on three very different types of documents. The area plots show how the DjVu file size, which is the sum of the number of bytes used to encode the mask, background and foreground layers, evolves as a function of this probability. When it is high, the document is over-segmented and bits are wasted in encoding a mask that includes images elements, texture and noise. As it converges to zero, the text gets encoded as background. Note that, to avoid interferences, the filtering stage was not applied. In each area plot, the vertical line corresponds to the value of the transition probability for which a human observed the best document quality. It has been observed that, for most documents, this point corresponds to the best compression rate. This property greatly facilitates the tuning of the segmenter.

6. Conclusion

DjVu is unique in a number of ways: its state-of-the-art segmentation, compression, and software design combine to create a platform that is so intuitive and powerful that most users never come close to realizing how many technological advances are behind DjVu. With the recent addition of electronic document

[4] The options corresponding to the parameters are available in the API or the command line of the DjVu encoder (available at `www.lizardtech.com`). The correspondence table is

Prior for the Gaussian means	`inhibit-foreback-level`
ForeBack transition probability	`pix-filter-level`
ForeBack threshold	`threshold-level`
ForeBack MLE block size	`block-size`
Filter threshold	`shape-filter-level`

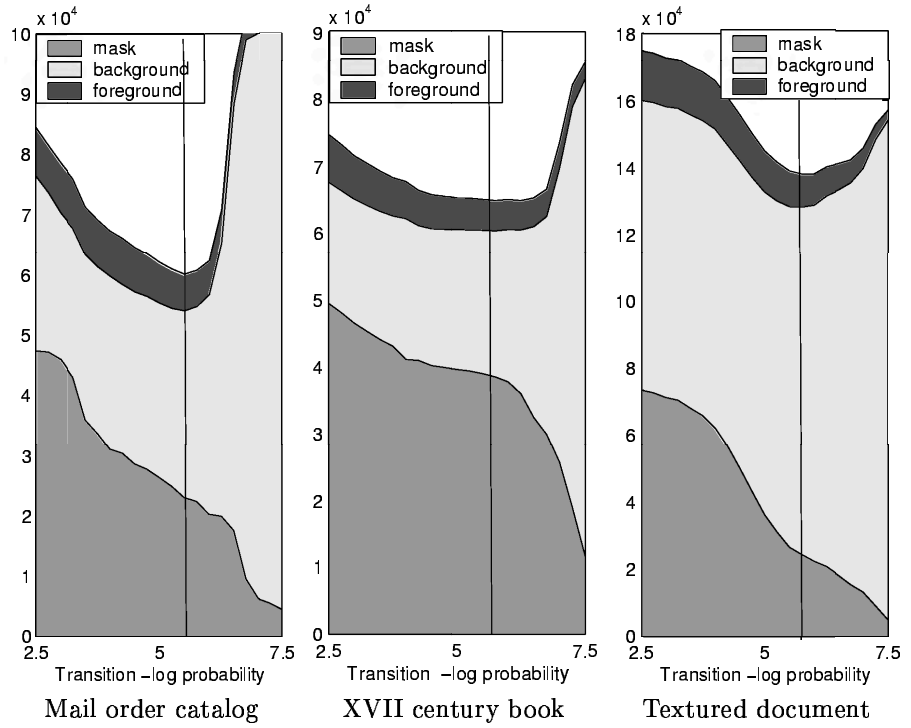


Fig. 5. *DjVu files sizes as a function of the negative log probability of the foreground/background transition, reported for three documents. The first two are browsable on the Internet at* www.djvuzone.org/djvu/cat/sharper *and* www.djvuzone.org/djvu/antics/pharm.

conversion to its portfolio, DjVu becomes a serious, powerful document interchange format, able to address the needs of a wide range of users.

A full range of DjVu authoring products, from non-commercial software packages to full-feature, advanced packages designed for enterprise users, is available from LizardTech, `www.lizardtech.com`. The technology is also partly available as open source at `djvu.sourceforge.net`, where new contributions are welcome. In addition, a growing number of third-party tools, from imaging packages such as `IrfanView` to searching and indexing engines such as `jss`, are supporting DjVu. The web site `www.djvuzone.org` is dedicated to the DjVu community , with news, documentations and many links to Digital libraries which use DjVu.

DjVu unique performance comes from that fact that it handles documents as multi-layer pixel images, resulting in a highly optimized representation. It offers many opportunities to apply existing and new image processing technologies to the field of document interchange, which is poised for growing explosively over the next decade.

Acknowledgements

The authors wish to thank Bill Riemers, Artem Mikheev, Joe Orost and Andrei Erofeev for their work on developing the DjVu system. They acknowledge Paul Howard, Pascal Vincent, and Yoshua Bengio key contributions to the DjVu technology. They also express their gratitude to Larry Rabiner for his continued support and to Jeffery Triggs for helping to the popularization of DjVu.

References

1. R. N. Ascher and G. Nagy. Means for achieving a high degree of compaction on scan-digitized printed text. *IEEE Trans. Comput.*, C-23:1174–1179, November 1974.
2. L. Bottou, P. Haffner, P. G. Howard, P. Simard, Y. Bengio, and Y. LeCun. High quality document image compression with DjVu. *Journal of Electronic Imaging*, 7(3):410–428, 1998.
3. L. Bottou, P. Haffner, and Y. LeCun. Conversion of digital documents to multilayer raster formats. In *Proceedings of the International Conference on Document Analysis and Recognition*, Sept. 2001.
4. L. Bottou, P. G. Howard, and Y. Bengio. The Z-coder adaptive binary coder. In *Proceedings of IEEE Data Compression Conference*, pages 13–22, Snowbird, UT, 1998.
5. L. Bottou and S. Pigeon. Lossy compression of partially masked still images. In *Proceedings of IEEE Data Compression Conference*, Snowbird, UT, March-April 1998.
6. GhostScript. available at http://www.ghostscript.com.
7. P. Haffner, Y. LeCun, L. Bottou, P. Howard, P. Vincent, and B. Riemers. Color documents on the web with DjVu. In *Proc. of ICIP-99*, 1999.
8. P. G. Howard. Text image compression using soft pattern matching. *Computer Journal*, 40(2/3):146–156, 1997.
9. S. Inglis. *Lossless Document Image Compression.* PhD thesis, University of Waikato, March 1999.
10. W. N. J. Sheinvald, B. Dom and D. Steele. Unsupervised image segmentation using the minimum description length principle. In *Proceedings of IAPR 92*, 1992.
11. Y. LeCun, L. Bottou, A. Erofeev, P. Haffner, and W. Riemers. DjVu document browsing with on-demand loading and rendering of image components. In *Internet Imaging*, Jan. 2001.
12. P. Macleod, X. Zhu, and L. Vincent. Method and apparatus for compressing color or gray scale documents. United States Patent 5,778,092, US Patent Office, 1998.
13. A. Mikheev, L. Vincent, and L. Bottou. Electronic conversion of documents to DjVu using a Windows virtual printer driver. In *Submitted to DAS'02*, Jan. 2002.
14. MRC. Mixed rater content (MRC) mode. ITU Recommendation T.44, 1997.
15. L. R. Rabiner and B. H. Juang. An introduction to hidden Markov models. *IEEE Trans. on Acoustics, Speech, and Signal Processing*, 3 (1), January 1986.
16. J. Rissanen. Stochastic complexity and modeling. *Annals of Statistics*, 14:1080–1100, 1986.
17. A. Said and W. A. Pearlman. A new, fast, and efficient image codec based on set partitioning in hierarchical trees. *IEEE Transactions on Circuits and Systems for Video Technology*, 6(3):243–250, June 1996.
18. J. Serra. *Image Analysis and Mathematical Morphology.* Academic Press, London, 1982.
19. W. Sweldens. The lifting scheme: A custom-design construction of biorthogonal wavelets. *Journal of Applied Computing and Harmonic Analysis*, 3:186–200, 1996.
20. D. Taubman. High performance scalable image compression with ebcot. *IEEE Transactions on Image Processing*, 9(7):1158–1170, July 2000.
21. R. Thibadeau and E. Benoit. Antique books. *D-Lib Magazine*, 1997. http://www.dlib.org/dlib/september97/thibadeau/09thibadeau.html.
22. I. H. Witten, A. Moffat, and T. C. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images.* Van Nostrand Reinhold, New York, 1994.