Locating and Recognizing Text in WWW Images*

Daniel Lopresti
dpl@research.bell-labs.com
Bell Laboratories
Lucent Technologies, Inc.
600 Mountain Avenue
Murray Hill, NJ 07974

Jiangying Zhou
jiangying@summus.com
Summus Ltd.
Suite 2200
2000 Center Point Drive
Columbia, SC 29210

November 18, 2002

Abstract

The explosive growth of the World Wide Web has resulted in a distributed database consisting of hundreds of millions of documents. While existing search engines index a page based on the text that is readily extracted from its HTML encoding, an increasing amount of the information on the Web is embedded in images. This situation presents a new and exciting challenge for the fields of document analysis and information retrieval, as WWW image text is typically rendered in color and at very low spatial resolutions.

In this paper, we survey the results of several years of our work in the area. For the problem of locating text in Web images, we describe a procedure based on clustering in color space followed by a connected-components analysis that seems promising. For character recognition, we discuss techniques using polynomial surface fitting and "fuzzy" n-tuple classifiers. Also presented are the results of several experiments that demonstrate where our methods perform well and where more work needs to be done. We conclude with a discussion of topics for further research.

Keywords: *document analysis, information retrieval, optical character recognition, WWW image text.*

1 Introduction

Traditionally, the field of document analysis has focused on the translation of information contained in paper documents to an electronic form. The myriad of problems that arise in the process have been studied for decades. Some are widely accepted to be difficult, while others have been addressed satisfactorily, at least in certain special cases. Many of the problems still considered open have nonetheless received a good deal of attention in the literature, and are well-understood, if not yet solved.

*Appears in *Information Retrieval*, 2(2/3): 177-206, May 2000.

A recent development of note, however, is the explosive growth of the World Wide Web (WWW) and the rapid proliferation of electronic documents it has fostered. Since 1993, the number of WWW servers has been increasing at an exponential rate, currently doubling every six months; it now totals over 7,000,000 [31]. The popular Alta Vista search engine indexes 250,000,000 Web pages, and processes tens of millions of HTTP requests each day [1, 22]. While it is now generally acknowledged, even by members of the Web community, that electronic documents will never totally supplant paper ones [28], there are compelling reasons to consider whether the analysis techniques originally developed for paper documents might have applications in the online world.

It might seem as though there are few, if any, "hard" problems in the case of electronic documents. Simple procedures (translators) can be used to convert from one encoded format to another. For example, OCR'ing the body of an HTML document is entirely unnecessary, as the text it contains can be easily extracted using available parsers, as illustrated in Figure 1. All of the popular Web search engines currently use this approach; while they incorporate sophisticated indexing techniques to guarantee quick response-times, they employ nothing in the way of complex document analysis. Thus, the problem would appear to be primarily one of *conversion* and not *analysis*.

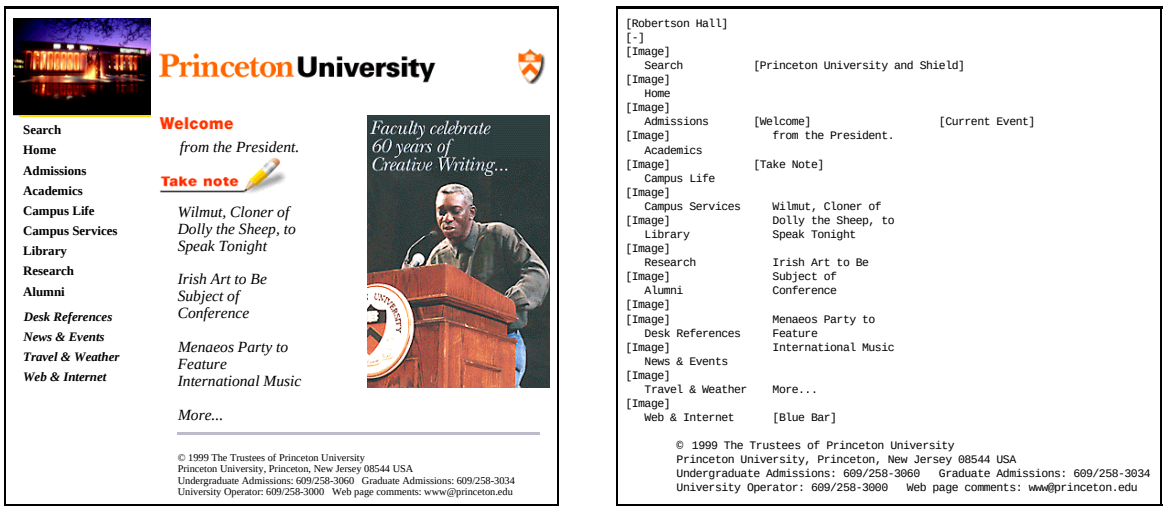


Figure 1: Example of a WWW page (left) and its textual content (right).

First impressions can be deceiving, however. To motivate our discussion, we note that the text string "Creative Writing," featured prominently in the snapshot on the left side of Figure 1, is completely absent from the extracted text on the right side of the figure. This is because it exists only as a bitmap image, and current WWW tools fail to capture such information. To the document analysis community, though, this problem is a classic one.

Raw ASCII text constitutes only one of the many data types in HTML documents. As Web page design becomes more sophisticated, more and more of the text present on the WWW is being embedded in images. The chart on the left in Figure 2 plots the percentage of text that appears in bitmap format (typically GIF) for an informal sampling of 25 WWW homepages. Note that there is a wide range, but that almost all pages present at least some

fraction of their text in image format. In some cases, the fraction is quite large (*e.g.*, 72% in the case of Page 13 in the chart). The average for these 25 pages is approximately 15%.[1]

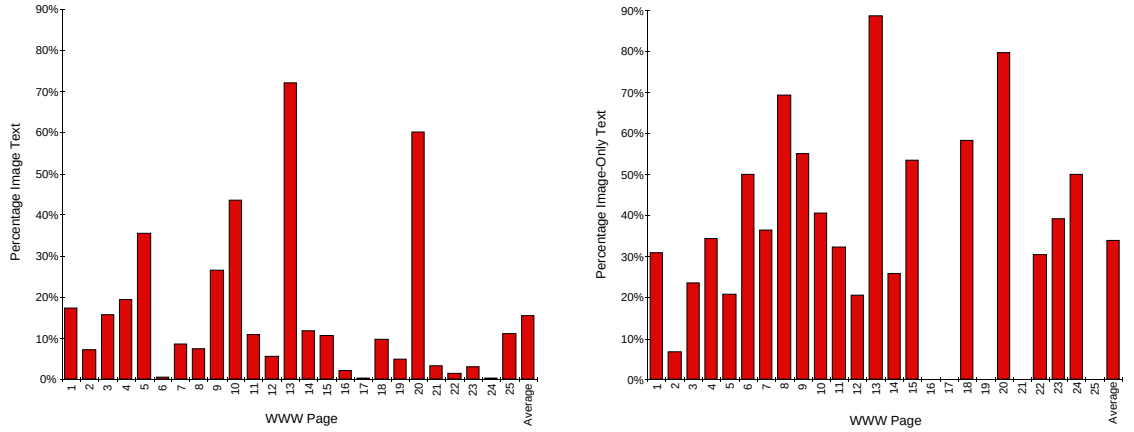


Figure 2: Percentages of total text presented in image format (left) and terms presented *only* in image format (right) for an informal sampling of 25 WWW homepages.

Perhaps more revealing is the chart shown on the right in Figure 2. Here we indicate the percentage of terms that appear in image format, but nowhere else in the HTML text. These documents are currently not being indexed on this information, and any search for the associated terms would fail. Note that for many of these pages, a nontrivial fraction of the image text does not appear elsewhere (to a maximum of 89%). The average here is 34%.

These figures suggest that a significant amount of the information in Web documents may be invisible to text-only methods. On the other hand, how information is actually encoded may not be obvious or even important from a user's perspective. The phrase "Creative Writing" is simply a string of text on a Web page, and the user may well want to search for it using a traditional keyword query. Consequently, there is a need to develop techniques for recovering the text in Web images.

In this paper, we survey the results of several years of our work in this area [15, 17, 33, 34, 35]. In particular, we focus on two key problems that distinguish the analysis of Web text from traditional black-on-white printed text: locating text strings in color images, and recognizing isolated, low-resolution, multi-colored characters. While these tasks are central to the automated processing of Web images, they are not by themselves sufficient to build a complete, end-to-end system. Hence, the results we report should be interpreted as first steps towards the ultimate goal of enabling access to all of the textual information contained

[1] Throughout this paper, we shall present statistics and experimental evaluations based on small, informally gathered datasets of WWW images containing text. Unless otherwise stated, all data was collected from real Web pages, without regard to how easy it would be to process using the algorithms under consideration. Although not necessarily identical in every case, there was some amount of overlap between the various datasets we used. Because of the enormous scope of the WWW and the small size of our test sets, the results we report cannot be considered definitive – only suggestive of an interesting and potentially important problem in document analysis and information retrieval.

in Web documents.

The remainder of this paper is organized as follows. In Section 2, we describe the two most common formats for images on the Web and the various technical issues that arise. We discuss the problem of locating text in WWW images in Section 3. In Section 4, we consider the recognition of characters that have been extracted from Web images. Section 5 reviews other work generally related to the problems in question. Finally, in Section 6 we offer our conclusions and suggest topics for further research.

2 WWW Image Formats and Related Issues

In principle, any format could be used to distribute images over the Web. All that is required is writing an appropriate browser "plug-in" to support it. However, two formats in particular have emerged as de facto standards: GIF and JPEG. Each of these is applicable in its own specific domain. While it is possible to adopt the high-level view that all Web images are simply 2-D arrays of pixels, there are good reasons to distinguish between the formats and their various "flavors." Such an understanding can contribute to the design of better-quality algorithms for locating and recognizing embedded text, as well as suggest new topics for inquiry.

An important consideration, independent of format, is that WWW images are designed to be viewed on computer monitors. This manifests itself in two ways: liberal use of color (up to 24 bits deep), and low spatial resolution (72 ppi). The former can actually make up for some of the limitations induced by the latter. Contrast this with the input to traditional document analysis, which typically consists of black text on a white background, printed and scanned at resolutions of 300 dpi or higher. A 10-point character nominally measures 83 pixels tall when scanned at 600 dpi. The same character may only be 10 pixels tall when displayed on-screen and yet still remain readable if the proper colors are used. Figure 3 illustrates the difference in spatial resolutions for a line of text typeset in 10-point Helvetica.[2]

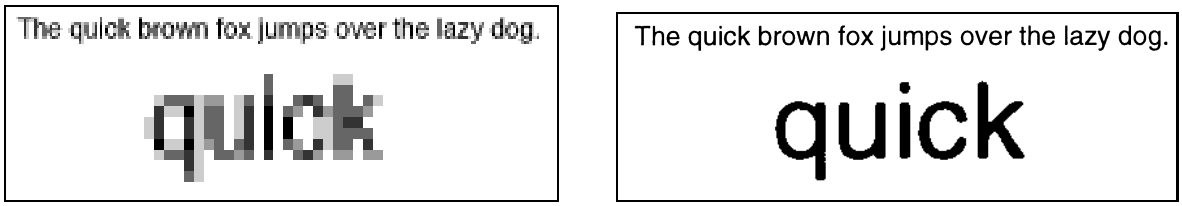


Figure 3: Text typeset in 10-point Helvetica for the Web (left) and printed/scanned (right).

Figure 4 presents some telling statistics for a small collection of characters gathered from WWW pages. Note in particular the wide range of sizes and numbers of colors. For comparison, the chart also shows the average areas for the same characters printed in Times font at 600 dpi and scanned at 300 dpi. Many of the Web samples are comparable in size to 4- or 5-point printed characters.

[2]Most of the examples displayed in this paper originated as full-color images and as such are much more legible. The need for hardcopy that can be reproduced economically means we must make do with grayscale approximations here.

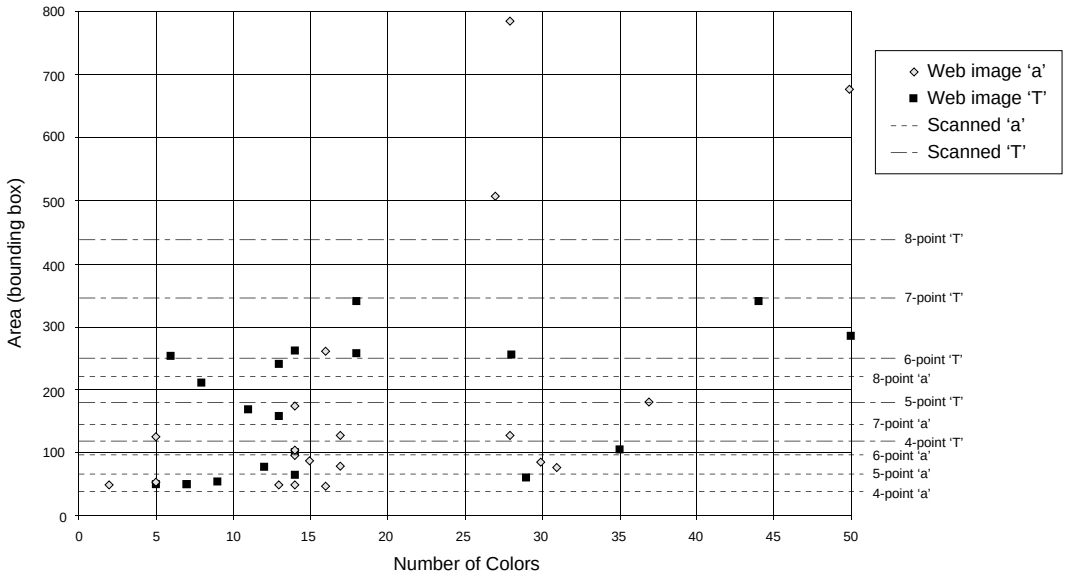


Figure 4: Areas and numbers of colors for characters collected from Web images.

2.1 Image Formats

Turning our attention to image formats, GIF (for "Graphics Interchange Format") was originally developed by CompuServe for use in exchanging images via e-mail. It is by far the most common image format encountered on the WWW today. In particular, nearly all of the small icons one finds sprinkled across Web pages are GIF's, as well as many of the larger images.

To save space and conserve bandwidth, GIF incorporates LZW compression. Since LZW is a lossless scheme, the decompressed image is the same quality as the original; there is never any degradation. Because the alphabet of possible symbols (*i.e.*, colors) is relatively small, LZW works well, especially when there are large regions of uniform color. These factors make GIF an efficient format for stylized graphics, but less appropriate for photographs.

On the other hand, JPEG, developed by the Joint Photographic Experts Group, combines DCT-based compression in the frequency domain with Huffman coding of the resulting DCT coefficients.[3] It supports 24-bit color (*i.e.*, any of over 16 million colors can appear in an image). This makes it the best choice for continuous tone images such as photographs.

Unlike GIF, JPEG is lossy. Compression is achieved by discarding the high order DCT coefficients. This saves space, but degrades the high frequency information in the image. As a result, JPEG is somewhat problematic when applied to images that contain text which is, in effect, a high frequency signal. JPEG's block-oriented nature can lead to visible distortion (artifacts) in the vicinity of characters. A textured "halo" is apparent surrounding the letters in Figure 5 (especially obvious to the immediate left of the 'Y'). For this reason, JPEG is usually reserved for images consisting solely of photographic data

[3]To be precise, JPEG is not a file format, it is a compression algorithm. The proper name for the associated file format is JFIF (JPEG File Interchange Format).

(*e.g.*, the designer of the Web page shown in Figure 1 chose to make the image on the right side a GIF, most likely because of the text it contains).

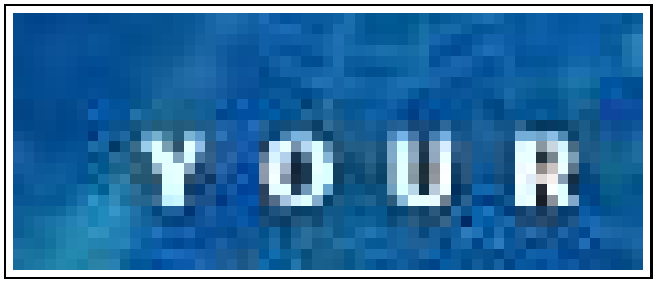


Figure 5: JPEG compression artifacts.

2.2 Anti-aliasing

Anti-aliasing is the process of reducing the jagged appearance of a sharp edge by blending the pixels at its boundary with the background. It is independent of the image file format. Figure 6 illustrates this effect for a fragment of text originally typeset in 48-point Times.

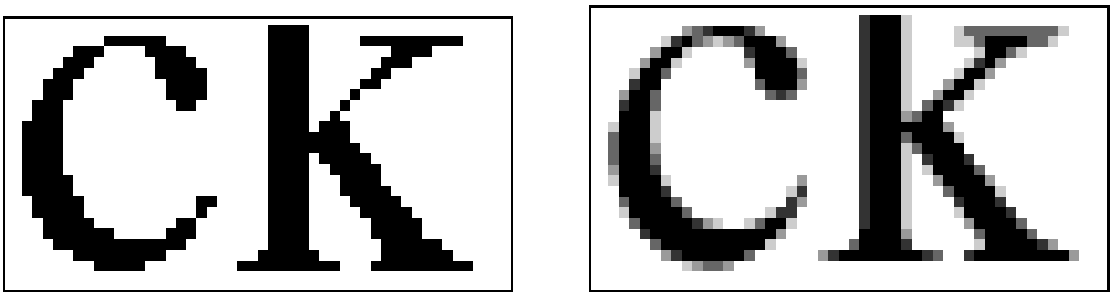


Figure 6: Aliased (left) and anti-aliased (right) text fragments.

While anti-aliased edges are smoother and more pleasing to the eye, there are in fact occasions where aliased text is preferable (see [26], page 96). This is particularly true for small font sizes – here anti-aliasing may result in excessive blurring, making the text more difficult to read (the text on the left side of Figure 3 was anti-aliased).

Since the matter is one of the designer's preference, text encountered in a Web image can be either anti-aliased or not. This issue could complicate the building of OCR classifiers for such characters, as will be discussed later.

2.3 Spatial Sampling Effects

Lopresti, Nagy, Sarkar, and Zhou recently observed that when scanning a page, the effectively random placement of the sampling grid (*i.e.*, the CCD sensor array) relative to the page can lead to significant variation in the resulting character bitmaps [14, 21]. An analogous effect occurs when producing a GIF or JPEG image using software such as Adobe

Photoshop. Abstract characters are placed on a virtual grid with a much finer resolution than the final output. At some point, however, these must be mapped to the intended resolution; the abstract characters are sampled at the coarser grid rate, yielding variability much like the physical scanning process.

Figure 7 shows eight instances of the same 12-point Helvetica 'e' taken from an image rendered at Web resolution. The difference in the bitmaps is dramatic and due entirely to the placement of each character as the image was created.

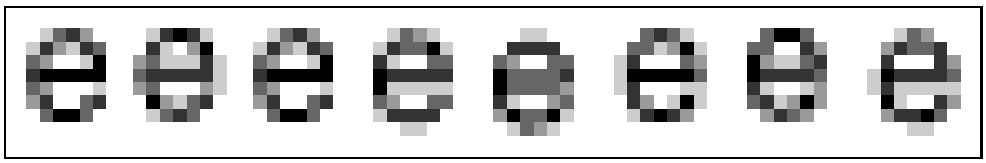


Figure 7: The spatial sampling effect for anti-aliased text (12-point Helvetica).

2.4 Other Issues

The process of designing a Web graphic is a highly creative, labor-intensive activity. Since attracting attention is often the primary goal, a large number of different techniques are employed. The powers of human perception still greatly exceed what is possible using a machine. Text can be overlayed on a complex background and made so nearly transparent that it would be impossible to locate much less recognize using existing approaches. Consider, for example, the letter 'a' in Figure 8 which blends perfectly into the background texture and yet can easily be segmented out by a human. This makes the image more interesting, but much harder to analyze for its textual content. Weinman presents a good survey of the various issues that arise when designing graphics for the Web [26].

Figure 8: An image with hard-to-segment text.

Moreover, unlike printed documents which are static, Web pages can be dynamic. This nature is manifested in several different ways. Perhaps most germane to the topic at hand are the animated GIF's now becoming popular. These are, in fact, part of the definition of the GIF89a standard. To extract text contained in such images is a challenge beyond the scope of the current paper.

It should also be noted that the problems of locating and recognizing text in WWW images would be moot if designers always elected to include the embedded text in the source document. The HTML image tag has an "alternate text" parameter ideally suited to this purpose (the HTML comments tag could also be used). This is, however, merely a convention; there is no way to enforce such a policy and many Web pages simply ignore it.

3 Locating Text in WWW Images

The general problem of locating text in color images has been addressed under different contexts in the literature. Most of these methods, however, are designed to process scanned images at a much higher resolution than what is available on the WWW. In a paper by Zhong, Karu, and Jain, the authors discuss two approaches for automatically locating text on CD covers [32]. Huang, *et al.* propose a technique based on grouping colors into clusters for foreground/background segmentation of color images [6]. Doermann, Rivlin, and Weiss present methods for identifying text (often stylized) in logos and trademarks [3]. Wu and Manmatha describe a text extraction and recognition algorithm for the "OCR-able text" in color images [29].

Some recent work with a connection to the subject focuses on locating text in video streams. In a paper by Li, Kia, and Doermann, the authors propose resolution enhancement based on Shannon interpolation to enable a better separation between text and background in video images [11]. Lienhart and Stuber describe a method for digital video that utilizes the inter-frame dependencies to enhance character segmentation [12].

Our approach for locating Web image text, to be described in the remainder of this section, is to divide the problem into three stages: color clustering, character detection, and layout analysis. Color clustering groups together multiple colors that are part of the same text and reduces the overall number of colors present in the image. Character detection identifies connected components in each color group and classifies them as "character" or "non-character." In the layout analysis stage, we check to see which of the detected characters are consistent with possible text layouts and eliminate those that do not meet the requirements.

An underlying assumption in our text detection algorithm is that each character is rendered in a *homogeneous* color. This assumption covers GIF images in which the text is composed of a single color, or a simple, uniform texture. It excludes instances where, for example, a character is rendered with its top half one color and its bottom half a different color.

3.1 Color Clustering

The goal of color clustering is to identify pixels from the same section of text and group them into a single cluster. In an earlier work, we presented a global clustering procedure that relies only on the RGB similarity of colors in the image [33]. The approach we describe here employs a hierarchical method that combines both the RGB similarity and the local spatial proximity of colors.

The RGB clustering method we use is based on the Euclidean minimum-spanning-tree (EMST) technique. The EMST-based clustering approach is a well-known method and has been researched extensively over the years. It has been shown that the method works well on a variety of distributions [4, 30].

Given N points in a M-dimensional Euclidean space, the Euclidean minimum spanning tree problem is usually formulated as a problem in graph theory: let the N points be nodes and let there be a weighted edge joining every pair of nodes such that the weight of the edge equals the distance between the two points. The Euclidean minimum-spanning-tree is a tree connecting all nodes such that the sum-total of distances (weights) is a minimum. Several robust and efficient algorithms are available to compute the EMST from a given point set [20].

It is easy to see how an EMST can be used as a basis for clustering. By definition, each edge in the EMST of a set of points S is the shortest edge connecting two partitions, P and $(S - P)$, of S. Intuitively, points in the same cluster should be connected by shorter edges than points in different clusters. Hence, by removing the longest edges from the EMST we separate the points of S into disjoint clusters.

To apply the EMST clustering algorithm, we view each color as a point in a three dimensional RGB space. For a Web image in GIF format, there are at most 256 unique colors, hence, the number of nodes in an EMST is no more than 256. The distance between two colors is computed as the Euclidean distance of their RGB elements. Once the EMST is constructed, we compute the average distance of the edges. Edges whose distances are larger than the average by a predetermined amount are then removed from the EMST. The EMST tree thus trimmed may contain several disjoint sub-trees, each of which corresponds to a color cluster.

EMST-based clustering works well when the color of each character is clearly distinguishable from the background. The method is less successful for text involving large numbers of colors. As mentioned earlier, text in Web images is often anti-aliased. This can introduce numerous border pixels with similar but not quite identical colors. Figure 9(a) is one such example. There are approximately 230 colors in the original version of this image, of which 130 are different hues of blue used for the text. This effect is more apparent in Figure 9(b), where a different random color has been assigned to each of the original colors in Figure 9(a).

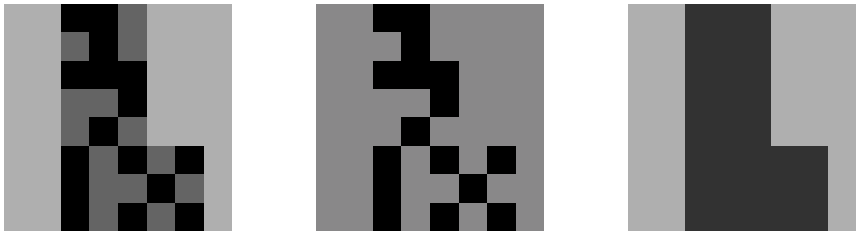
In the clustering process, this gradual change of color produces a chain-like EMST linking colors from the foreground (dark blue) to the background (white). The EMST-based clustering method may not properly break such a chain-like tree structure, resulting in either a failure to separate the background from the foreground text, or fragmentation of the characters.

When dealing with textures, the EMST-based approach may fail because it measures color similarity using only RGB components. Figure 10 illustrates the problem. The letter 'L' in the image is rendered with a "checkerboard-like" texture. The actual color pattern is replicated in Figure 11. The color values in RGB space are: $A = (0, 0, 0)$, $B = (100, 100, 100)$ and $C = (175, 175, 175)$. To a human observer, it is obvious that colors A and B together form the letter 'L' and that color C is the background. However, in terms of RGB distance, color B is closer to the background color C (RGB distance = $75\sqrt{3}$) than to color A (RGB

(a)

(b)

Figure 9: (a) Anti-aliased text in a GIF image. (b) Same image with colors reassigned.



(a) (b) (c)

Figure 10: (a) A letter 'L'. (b) Clustering based on RGB distance. (c) Clustering based on combined RGB/spatial distance.

distance = $100\sqrt{3}$). Thus, in a color clustering scheme using a pure RGB distance, colors B and C will be clustered together.

To address the issues just raised, we observe that colors that are spatially close to each other tend to belong to the same object. In order to improve the clustering performance, we incorporate a *local* clustering process prior to the application of the EMST method. This procedure takes the spatial distribution of colors into consideration when grouping similar colors.

First, we introduce a measure for the "spatial proximity" of colors: let $d_p(X)$ denote the distance between a given pixel p and the closest pixel with color X. For example, if we let p be the pixel at the fourth column and first row in Figure 11 and X be color B, then $d_p(B) = 1$. The closest pixel of color B is at the fifth column and first row, one pixel to the right of p. If we let X be color C, then $d_p(C) = 2$ since the closest pixels of color C are at the second and fifth columns of the first row. Denote by N an $m \times m$ neighborhood in

C	C	A	A	B	C	C	C
C	C	B	A	B	C	C	C
C	C	A	A	A	C	C	C
C	C	B	B	A	C	C	C
C	C	B	A	B	C	C	C
C	C	A	B	A	B	A	C
C	C	A	B	B	A	B	C
C	C	A	B	A	B	A	C

Figure 11: Symbolic color map for the letter 'L' of Figure 10(a).

the image, and by $c(p)$ the color of pixel p. Let set $\mathcal{P}_N(X)$ represent all pixels of color X in the neighborhood N:

$$\mathcal{P}_N(X) = \{p \in N \mid c(p) = X\} \tag{1}$$

Then the spatial distance from color X to color Y in the neighborhood of N is calculated as follows:

$$D^s_N(X,Y) = \frac{1}{\#\mathcal{P}_N(X)} \sum_{p \in \mathcal{P}_N(X)} d_p(Y) \tag{2}$$

where $\#\mathcal{P}_N(X)$ is the number of elements of $\mathcal{P}_N(X)$. Equation 2 is the average of the distances from every pixel of color X to the nearest pixel of color Y.

The distance defined in Equation 2 is not symmetric, *i.e.*, $D^s_N(X,Y)$ and $D^s_N(Y,X)$ are not necessarily equal. However, if pixels of two colors X and Y are from the same pattern and thus spatially interwoven, both $D^s_N(X,Y)$ and $D^s_N(Y,X)$ will be small. To make the distance symmetric, we define:

$$\overline{D}^s_N(X,Y) = \frac{D^s_N(X,Y) + D^s_N(Y,X)}{2} \tag{3}$$

Let $D^c(X,Y)$ be the RGB distance between colors X and Y. The combined RGB/spatial distance between colors X and Y in neighborhood N is defined as the following function of their RGB and spatial distances:

$$D_N(X,Y) = D^c(X,Y) \times \overline{D}^s_N(X,Y) \tag{4}$$

Continuing the example from Figure 11:

$$\begin{array}{lll} D^s_N(A,B) = 1.00 & D^s_N(B,A) = 1.00 & D^s_N(A,C) = 1.46 \\ D^s_N(C,A) = 1.77 & D^s_N(B,C) = 1.59 & D^s_N(C,B) = 2.12 \end{array}$$

$$\begin{array}{lll} \overline{D}^s_N(A,B) = 1.0 & \overline{D}^s_N(B,C) = 1.86 & \overline{D}^s_N(A,C) = 1.62 \\ D_N(A,B) = 100\sqrt{3} & D^N(B,C) = 140\sqrt{3} & D_N(A,C) = 122\sqrt{3} \end{array}$$

We observe that the new combined distance between A and B is smaller than the combined distances between either A and C or B and C. If we were to cluster the colors in Figure 11

using the distance defined in Equation 4, colors A and B would be correctly grouped together (see Figure 10(c)).

In applying the local clustering scheme to actual images, we first divide the image into non-overlapping $m \times m$ blocks and process each block independently. The parameter m is chosen such that each region is roughly bi-tonal ($m = 8$ in the current implementation). This allows us to make the assumption that there will likely be only two primary clusters for pixels in the region: one corresponding to the foreground color and the other to the background color. The clustering algorithm is a bottom-up approach that outputs one, two, or three color clusters depending on the input block. The three-cluster situation may arise in cases where the region contains a foreground object (*e.g.*, a text fragment), its shadow, and the background.

The clustering process first uses the well-known nearest-neighbor technique to group pixels in a block into three clusters. It then decides either to combine the clusters further or to stop clustering. This decision is based on comparing the remaining distances with a fixed threshold. If all the colors are extremely similar and distributed evenly throughout the block, the algorithm will output only one cluster.

3.2 Character Detection

Once the color space has been clustered, the next step is to find connected components belonging to each color cluster and identify those components that correspond to text. Here we use a classification process based on geometric features extracted from the connected components. These include size, aspect ratio, the presence of holes and branches, *etc.* [33].

To reduce the complexity of the analysis, the classification process assumes that a text component corresponds roughly to a single character. Unfortunately, the connected components extracted from color clustering sometimes contain more than one character. Multi-character components may arise when text is underlined or characters touch (*e.g.*, due to anti-aliasing), or when the clustering process fails for some other reason. Figure 12 shows two examples where the single-character-component assumption is violated.

Our remedy is to introduce a segmentation step into the classification process. Notice that this segmentation problem is different from the one in traditional document analysis: in the latter case, the text is already identified and the object of the segmentation is to isolate individual characters, whereas the goal here is to decide whether or not a connected component is in fact text. Indiscriminately breaking components into smaller pieces will result in an excessive number of false alarms since non-text objects can be made to look like text when broken up.

The classification process is divided into three phases. In the first, we select candidate text-like components based on features that are relatively invariant to touching. These include size, stroke width, and white-to-black-area ratio. In the second phase, we single out those components from the first phase that have an "elongated" shape (*i.e.*, components whose width/height ratio exceeds a given threshold) and apply the segmentation procedure.

The segmentation process first determines the dominant color for each component – this is the color used by the majority of the pixels in the component. The dominant color of a character usually appears in the interior of its structure, while other colors are present

(a)

(b)

Figure 12: Examples of characters touching in GIF images.

around its edges. We use this knowledge to aid the segmentation process.

Two criteria are used for segmentation. The first is an analysis of the color distribution along scanlines. Currently, we assume that text is roughly horizontal. The segmentation computes a histogram of foreground pixels at each column and identifies potential break-points. A column position is designated as a potential break-point if it contains only a small number of foreground pixels, of which few are the dominant color.

The second criterion is an examination of the color correlation between adjacent scan-lines. The segmentation procedure checks the consistency between pixels in adjacent columns. We expect columns that fall between characters to exhibit a low color correlation. Consider, for example, Figure 12(b); if we take any two adjacent columns within the letter 's' or the letter 'o', we see that the number of rows whose pixel-pairs are either both foreground or both background is greater than the number of rows where only one pixel is foreground. On the other hand, at the right edge of the 's', the number of rows whose adjacent pixels have the same type is much smaller than the number of rows whose adjacent pixels have different types. This is where we place break-points. To avoid over-segmentation, we further restrict that the minimum distance between break-points be half the height of the connected component.

Finally, all the components from the above two phases are subjected to an additional filtering process. This makes use of more sophisticated features such as the number of holes and the number of upward/downward ends to further weed out non-text components.

Figure 13 shows the character detection result for the GIF image from Figure 9(a). Note that almost all of the pixels corresponding to characters have been identified, along with a small number of false alarms.

Figure 13: Results of the character detection stage.

3.3 Layout Analysis

The layout analysis we use can be viewed as a form of post-processing. Character detection based solely on geometric features cannot be made 100% reliable. A wide range of non-text objects may resemble characters and thus be erroneously classified. The post-processing step attempts to eliminate these false alarms by using additional heuristics based on layout criteria typical of text.

In post-processing, we consider each color cluster independently. We first group characters to find words or text lines. Characters that are close in the horizontal direction and whose top and bottom extents are aligned (or nearly so) are said to belong to the same word. A measure of "goodness," which we call the *saliency* of the text, is then calculated for each word. The saliency of a word is computed as follows:

$$S = \frac{c}{1 + \sigma_h/\bar{h} + \sigma_b/\bar{h}} \tag{5}$$

where σ_h and σ_b are the standard deviations of the heights and the baselines of the characters in the word, $\bar{h}$ is the average height, and c is the *character factor* which is 0, 0.5, or 1 for a word containing one, two, or more than two characters, respectively.

Saliency is a simple heuristic measure that reflects the degree of height and positional alignments of the characters in a word. Clearly, the saliency measure gives higher scores for words containing larger numbers of characters and having more regular typography.

Post-processing eliminates those words whose saliency measures are lower than a given threshold. In addition, we apply a set of procedures to handle certain special cases:

Nested components Occasionally, the pixels inside the holes of characters such as letters 'o' and 'b' satisfy the criteria for being characters themselves. This results in false words detected inside of actual words. An example of this can be seen in Figure 13, where parts of the white background are classified as characters. These false alarms tend to align fairly well and can often be grouped into words. However, they usually have lower saliency measures than the words they lie within because their alignment is not quite as good. Therefore, we eliminate all words that lie within another word whose saliency score is higher.

Shadow text Text in GIF images sometimes includes a shadow effect. This may result in the detection of partially overlapping words, one corresponding to the actual word, the other to its shadow (see Figure 14). We note that a word's shadow usually contains fewer pixels than the word itself and can be eliminated on this basis.

Figure 14: An example of text with a shadow effect.

Figure 15: Final output of the algorithm for the example from Figure 9(a).

Figure 15 shows the final output of the algorithm for the input image from Figure 9(a). Observe that the false alarms seen in Figure 13 have been eliminated.

3.4 Experimental Evaluation

We tested our text extraction algorithm on 482 GIF images collected from real WWW pages. All of these images contained some amount of text. Table 1 summarizes several statistics for the test database.

Our current method for evaluating performance is to manually transcribe the text from each image before and after running our algorithm. In either case, the decision as to whether a given character is "readable" is subjective. Generally, if enough pixels are missed (or added) that a character's shape is visibly affected, the character is regarded as unreadable (see, for example, the letter 'S' in "HOTELS" in Figure 15). Although in many cases a human could still read the text in question, OCR processes tend to be much more sensitive; hence, we err on the side of being conservative. We then apply an approximate string matching procedure to count the number of correctly detected characters.

Since most classifiers are capable of filtering out small and/or oddly-shaped components as noise, it is impossible to know when a "false alarm" is likely to induce a recognition error. Hence, for the purposes of the following analysis we ignore spurious, non-character components that appear in the output.

	Minimum	Maximum	Average
Number of characters	2	496	34
Number of colors	2	256	61
Width (pixels)	20	667	285
Height (pixels)	8	799	75

Table 1: Test database statistics (482 GIF images).

The results of this experiment are depicted graphically in the chart on the left in Figure 16. Here we plot the performance both for the approach mentioned earlier using only RGB clustering as well as the algorithm just described based on combining RGB clustering and spatial proximity. For the latter, the average character detection rate for this set of examples was found to be 68.3%. While this may seem rather low, it is a significant improvement over the RGB-only result (a 47% average detection rate for a similar database [33]). The combined approach not only improved the overall performance, it also reduced the percentage of "catastrophic" failures (images where the detection rate was 20% or less), as indicated in the chart. Figure 17 shows some of the sample images and their corresponding results.

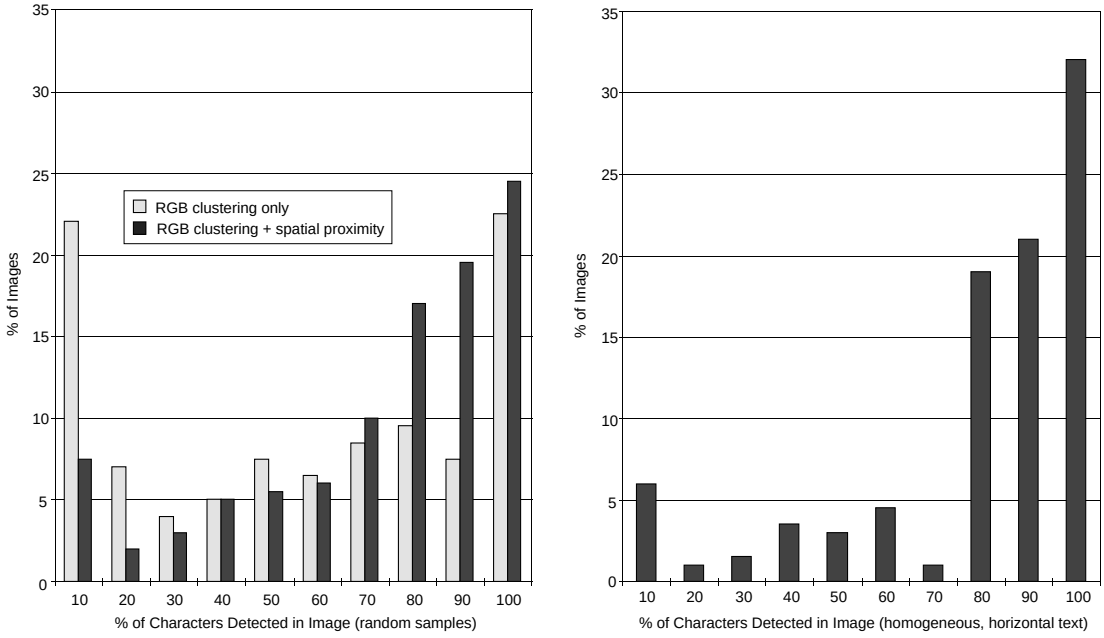


Figure 16: Web image text detection performance.

The test images used in the above experiment were selected randomly from WWW pages without regard to several of the assumptions made earlier. As a consequence, some contained non-homogeneously-colored text and/or non-horizontal layouts. To see how well our algorithm performs when these conditions hold, we manually inspected the test set and partitioned it into two groups: the first consisting of 315 images which met the assumptions, and the second the remaining 167 images which did not. The chart on the right side of Figure 16 shows the results when running our algorithm on the images from the first group. The average character detection rate in this case rose to 78.8%.

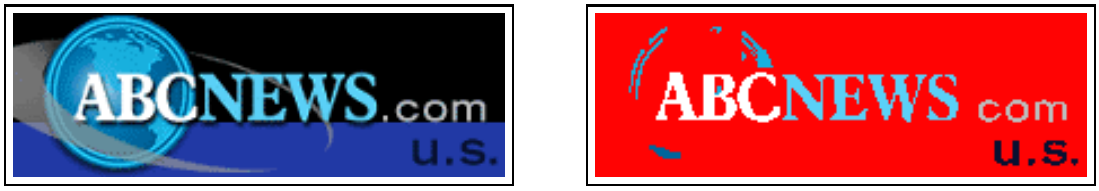
Figure 18 presents some difficult examples from the second group of images. These include non-linear text layout, embossed text, non-homogeneously-colored text, and stylized text (a company logo). We broke this set into three distinct cases:

1. Skewed or wrapped text. Only a few instances of severe skew angles or wrapped text

(a)

(b)

(c)



(d)

Figure 17: Examples of GIF images and the results of text detection.

(a) (b) (c) (d) (e)

Figure 18: Examples of images the current algorithm cannot handle well.

were found in the test set. Most images in this category contained text skewed just slightly off the horizontal.

2. Non-homogeneously-colored text. Detection rates for text in this class were low, demonstrating the importance of the homogeneity assumption.

3. Extremely small text. Some text in Web GIF images is very small (3-5 pixels high). This results in missed-detections since the algorithm uses a size threshold to filter out "noise" in the image.

Table 2 shows a performance analysis for the images in each of these classes.

4 Recognizing Text in WWW Images

The difficulties faced in attempting to recognize characters extracted from WWW images are analogous to the problems encountered in OCR'ing low resolution, poor quality text. This topic has been previously examined from several perspectives. Lee, Pavlidis, and

	Number of Images	Average Character Detection Rate
Case 1 (skewed)	18	70.5%
Case 2 (non-homogeneous)	12	36.9%
Case 3 (extremely small)	21	44.3%

Table 2: Performance on various difficult test cases.

Wasilkowski, for example, conducted a theoretical study on the trade-offs between spatial resolution and quantization resolution in signal processing [8]. Their work shows that it is possible to recognize text successfully at low resolutions if color information is preserved during the digitization. It is predicted that for 10-point text the sampling rate could be as low as 100 dpi (but not much lower).

Based on this result, Li and Pavlidis later proposed a character recognition technique which extracts features directly from a gray level input image [25]. Their method takes the digitized image as a terrain (with height denoting the "darkness" of the pixels) and analyzes the topographic structures of the terrain. The pixels are classified into different topographic feature classes such as *ridge*, *pit*, *saddle*, *ravine*, *etc.* according to the estimated first and second directional derivatives of the image intensity surface. It then extracts the basic structural information from the input image and represents the features as a topographic feature graph. The recognition process then compares the graph with predefined prototype graphs.

Li and Pavlidis's method provides a powerful framework for analyzing shapes. The surface fitting method that we describe here is, in spirit, similar to their approach. Our method assumes that the foreground color is roughly uniform for a given character in a Web image. It detects a representative foreground color for each character and computes the difference between this color and the color of each pixel. The result forms a 3-D surface similar to a terrain. We then derive a set of features from the surface for recognition.

Clearly, topographic feature analysis is computationally intensive. An alternative approach is to use techniques related to matched filters. A representative of this kind of method is orthogonal expansions [10]. In this method, a character in color space is expressed as a linear superimposition of different orthogonal primitive patterns. Note that a Web image usually contains a relatively small amount of text, hence the orthogonality assumption is usually valid since the text is most likely of a single font. In addition, noise commonly seen in scanned images, such as blurring or speckles, does not appear in Web images so a small number of primitives may be sufficient.

In our approach, a text detection algorithm such as the procedure described in the previous section is used to identify character-like connected components in each color class based on their shapes. These character-like components are assumed to be segmented and extracted from the original color input image. Typically, each represents a single character. Next, we proceed to convert the extracted character images into gray scale by comparing the difference between the representative color of the foreground and the color of each pixel. The former is assigned a gray value of 255. All other colors are assigned gray values of

$255 - d$, where d is the distance between the color and the representative color. We then apply a recognition process to each character image.

4.1 The Polynomial Surface Fitting Method

The first recognition method we consider is based on polynomial surface fitting. We represent a character shape by a polynomial surface function. Here the shape in the intensity image is treated as a 3-D surface, *i.e.*, $z = g(x, y)$ where (x, y) is the pixel location and z is the intensity value. A polynomial function of certain degree can then be used to fit to this surface.

In theory, the higher the degree of the polynomial, the better it can capture the shape of the character. However, a high degree polynomial is also more sensitive to noise. Currently, we use fourth degree polynomials for the surface representation [9, 24]:

$$z = f(x, y) = \sum_{0 \leq i,j \leq 4} a_{ij} x^i y^j \tag{6}$$

The fitting of the polynomial surfaces is done based on the least square principle. Let $Z = \{(k, l, t) \| 0 \leq k < K,\ 0 < L \leq J\}$ represent the image data, where (k, l) specifies the pixel coordinates and t is the image intensity value. The best polynomial surface representation is a vector of coefficient a_{ij} such that the mean square error:

$$\frac{1}{KL} \sum_{k,l} (\sum (a_{ij} k^i l^j - t)^2 \tag{7}$$

is minimized.

From the polynomial representation we then extract a set of features for recognition. Notice that the monomial basis functions $x^i y^j$ described in the above equation are not orthogonal to each other, thus the coefficients of lower-order monomials will change when the degree of the polynomial surface fit increases. Hence, the coefficient vector for monomial basis functions is not a good feature vector. Instead, we use orthogonal Legendre polynomial basis functions to represent the polynomial surface:

$$P(n, x) = \frac{1}{n! 2^n} \frac{d^n (x^2 - 1)^n}{dx^n} \tag{8}$$

The surface will have the following representation: $\sum_{0 \leq i,j \leq 4} b_{ij} P(i, x) P(j, y)$. Here b_{ij} is the coefficient vector for Legendre polynomial basis functions. These coefficients are used as a feature vector for the character shape. A linear classifier is used to compare each feature vector with a reference vector for each character class and then assign the character to the class with the closest distance.

The polynomial surface representation only captures the global shape of a character. Characters whose shapes are similar to each other (*e.g.*, 'c' and 'e') may not be distinguished reliably using this method since variations in the fonts and color intensities may introduce significant "cross-over" of the class distributions. One possible solution to this problem is to group character classes of similar shape into "meta-classes" (*e.g.*, 'c' and 'e' would form

a meta-class). An input pattern is first classified into one of the meta-classes. A second stage classification is then used to distinguish among characters in the same meta-class by examining finer features from the image.

For example, in the second recognition stage, we might use the topographic features described by Li and Pavlidis to extract the skeleton from the image. Figure 19 shows the ridge points in images of letters 'a' and 'e' detected by the method. As can be seen, the method recovers the skeletons remarkably well despite severe distortion in the character shapes.

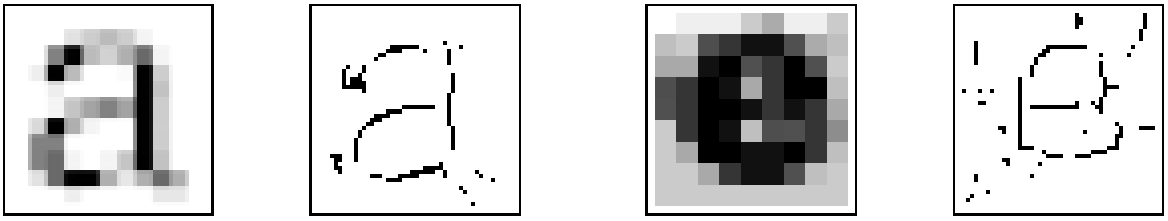


Figure 19: Ridge points detected in GIF images of 'a' and 'e'.

4.2 The N-tuple Method

The polynomial surface representation method just described requires a significant amount of time to compute. Furthermore, it does not work well when the resolution in terms of color intensities is coarse to begin with (*e.g.*, only a few shades of color are used to represent a character). An alternative we have explored is to use an approach adapted from n-tuple classifiers.

N-tuple classification is a relatively old idea, first proposed in 1959 [2]. It has received only scant attention since the early days of OCR research. For recognizing text in Web images, however, it appears quite promising, as we shall show.

An n-tuple is simply a set of locations in an image with specified colors. For a binary image, an n-tuple represents the presence or absence of a specific configuration of black and white pixels in a given pattern. For recognition, an n-tuple is superimposed onto an input image and the colors at the given locations are compared. An n-tuple is said to "fit" the image if the colors at all its locations match with the image color at the corresponding locations. Usually, several n-tuples are used as templates and the tuples can be shifted over the image in order to find the best fits.

Most n-tuple methods reported in the literature are defined over a binary input space. An obvious way of applying such n-tuple methods to Web text OCR is to threshold the color image into a binary bitmap and treat the problem as standard n-tuple classification. The drawback of doing so is that the information contained in the color bits will be lost in the thresholding process. The approach we developed, on the other hand, takes into consideration the "fuzziness" of the differentiation between foreground objects (*i.e.*, text characters) and the background in the input image. Instead of an exact fit, our method measures the degree a tuple matches the input image pattern and finds the class the tuple matches with the maximum likelihood.

To do so, we first assign to each pixel in the input image a value between 0 and 1 which represents the certainty the pixel belongs to the foreground. Let $P[i,j]$ be the input image and c_f be the representative color of the foreground determined by the text extraction algorithm of the previous section. We define the normalized color difference between c_f and the pixel color at pixel (i,j) as a measure of confidence the pixel belongs to the foreground:

$$I[i,j] = 1 - \frac{d(P[i,j], c_f)}{\max_{i,j} d(P[i,j], c_f)} \tag{9}$$

where $d()$ is the distance between c_f and pixel color $P[i,j]$. From the definition, we have $0 \leq I[i,j] \leq 1$. $I[i,j] = 1$ indicates a definite foreground pixel, and $I[i,j] = 0$ represents a definite background pixel.

To apply the n-tuple method, let $\{t_i = (p_{i1}, p_{i2}, ...p_{in})\}, i = 1, 2, ..., m$ be a set of n-tuples defined over the input image space of dimension $W \times H$. Each p_{ij} specifies a location in the image space. Given an image pattern, we first assess the probability that the pixel in the image at location p_{ij} as specified by the i^{th} n-tuple is foreground and compute a joint probability distribution for all location combinations.

Formally, let $b = b_1 b_2 ... b_n$, where $b_i \in \{0, 1\}$. We say pixel location p_{ij} of tuple i is designated to be foreground if $b_j = 1$, and background if $b_j = 0$. When every b_j in b has been assigned a particular designation, we say that b represents a specific foreground location configuration. Let $P^c(i : b_1 b_2 ... b_n)$ denotes the probability that the foreground color appears in location configuration $b_1 b_2 ... b_n$ in the image pattern c. For example, for a 7-tuple, $P^c(i : 0110000)$ represents the probability of the foreground color appears at locations p_{i2} and p_{i3} simultaneously, but nowhere else. For an n-tuple, there are total of 2^n different possible configurations. Let B be the set of all possible configurations of $b_1 b_2 ... b_n$. For a given image pattern, $P^c(i : b_1 b_2 ... b_n)$ is computed as:

$$P^c(i : b_1 b_2 ... b_n) = \prod_{b_j=1} I[p_{ij}] \prod_{b_j=0} (1 - I[p_{ij}]) \tag{10}$$

One issue not yet addressed is how to select a set of appropriate n-tuples for the classifier. For many non-trivial recognition problems, it is usually computationally intractable to find the optimal choice of n-tuples. In our current implementation, the locations p_{ij} for each tuple are chosen randomly from the input image space.

During the training process, the value $P^c[i : b_1 b_2 ... b_n]$ is evaluated for every pattern in a character class C and a summation is calculated:

$$P(i : b_1 b_2 ... b_n) = \sum_{c \in C} P^c(i : b_1 b_2 ... b_n) \tag{11}$$

We then estimate the conditional probability of the foreground appearing in configuration $b_1 b_2 ... b_n$ given class C and tuple i as:

$$P(i : b_1 b_2 ... b_n | C) = \frac{P(i : b_1 b_2 ... b_n)}{\sum_{b_1 b_2 ... b_n \in B} P(i : b_1 b_2 ... b_n)} \tag{12}$$

In the subsequent classification process, we compute the probability of an input pattern x given n-tuple i of class C using the following equation:

$$P(x|i:C) = \sum_{b_1 b_2 ... b_n \in B} P^x(i:b_1 b_2 ... b_n) P(i:b_1 b_2 ... b_n|C) \tag{13}$$

and the probability of x given all the n-tuples of class C is given as:

$$P(x|C) = \prod_{i}^{m} P(x|i:C) \tag{14}$$

A Bayesian maximum likelihood classifier is then used to determine the class of the input pattern. We assume here that the *a priori* probabilities are the same for all character classes and assign the pattern to the class for which $P(x|C)$ is a maximum.

4.3 Experimental Results

To test the effectiveness of these two methods, we collected 50 images from the Web and extracted from them 215 lowercase characters in 8 classes. Figure 20 shows some samples of these characters, while Table 3 presents statistics for the test data.

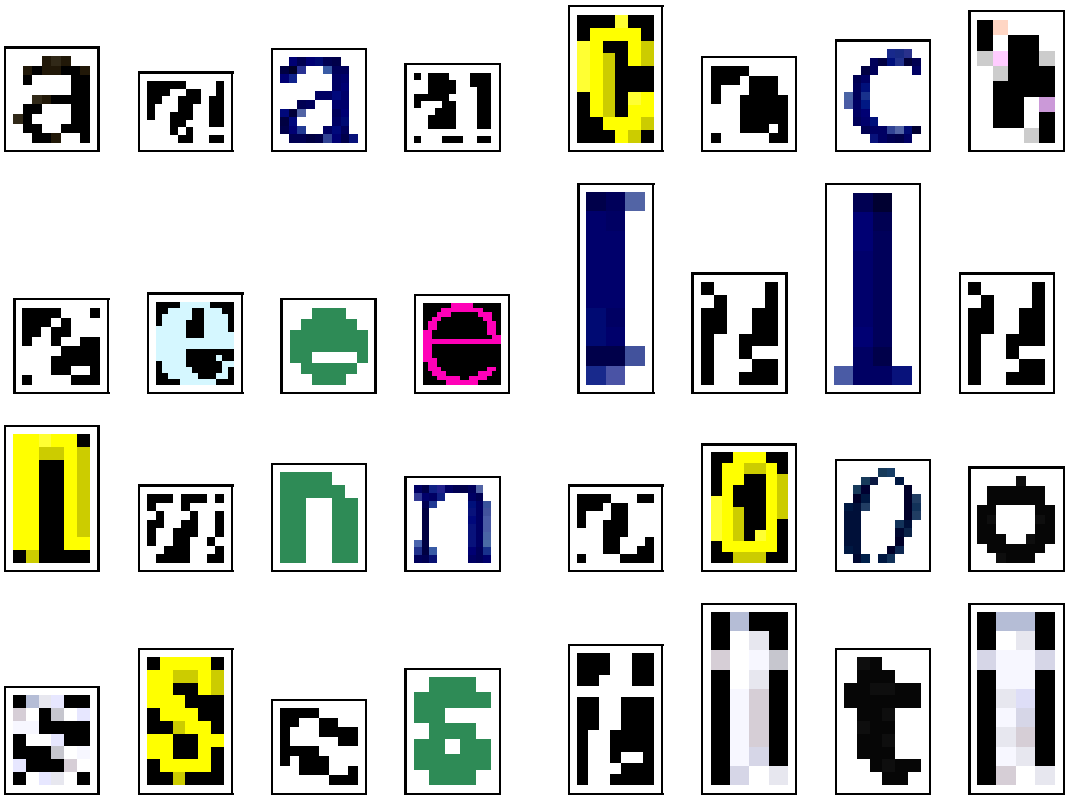


Figure 20: Sample characters from the test set.

Character Class	Number of Samples	Average Width	Average Height	Average Colors
a	32	14	14	6
c	26	8	10	5
e	37	10	12	5
i	17	5	11	3
n	27	11	13	7
o	29	9	10	4
s	26	8	10	4
t	21	7	14	3

Table 3: Test set statistics.

Character Class	Recognition Accuracy
a	51.6%
c	88.5%
e	57.6%
i	100.0%
n	84.0%
o	44.8%
s	69.2%
t	85.7%
Average	69.7%

Table 4: Recognition accuracies for the surface fitting classifier.

For the surface fitting classifier, fourth degree explicit polynomial surfaces were computed to represent each character. In the test, we used half of the characters as training samples and stored the polynomial surface parameters as the feature vectors. We then ran the classifier on the whole data set. The average distance from the character image to be recognized to the stored feature vectors of a whole class was used as the distance measure. The class that a character belongs to was chosen as the class with the smallest character-to-class distance. We used the normalized Euclidean distance between the polynomial coefficient vectors instead of algebraic invariants to measure the distance between two characters. This is because there was not much rotation in the character data set, and the images could be scaled to compensate for differences in character size. Table 4 shows the recognition accuracies for the eight classes as well the overall performance.

From the table, we can see that the recognition results were rather poor for character classes 'a', 'e', 'o', and 's'. The reason for this is that the 'c', 'e', and 'o' triplet as well as the 'a' and 's' pair have similar shapes and it is hard to distinguish them using fourth degree polynomial surfaces since the small structural differences between these classes are usually lost in the representation. Another reason for the poor performance here is that most of the character images in our data set are coarse in the color space (as can be seen from Table 3,

Character Class	Recognition Accuracy
a	90.6%
c	92.3%
e	73.0%
i	94.1%
n	100.0%
o	96.6%
s	88.5%
t	85.7%
Average	89.3%

Table 5: Recognition accuracies for the n-tuple classifier.

the average number of unique colors in each image is quite low). A continuous surface representation is not suitable for representing these images because of the high frequency intensity changes.

For the n-tuple classifier, we scaled the sample images to fixed-size 8×8 bitmaps. Next we set the parameters n and m (the tuple dimension and the size of the tuple set) and generated the n-tuples. Again, we took half the samples from the sample set for training the classifier. We then ran the classifier over the entire data set. Table 5 shows the recognition rates for $n = 10$ and $m = 10$.

We experimented with a few combinations of n and m in the test. Figure 21 shows the correlation between the recognition rate and the tuple dimension for a fixed $m = 10$. Table 6 shows the correlation between the recognition accuracy and the tuple set size at $n = 10$.

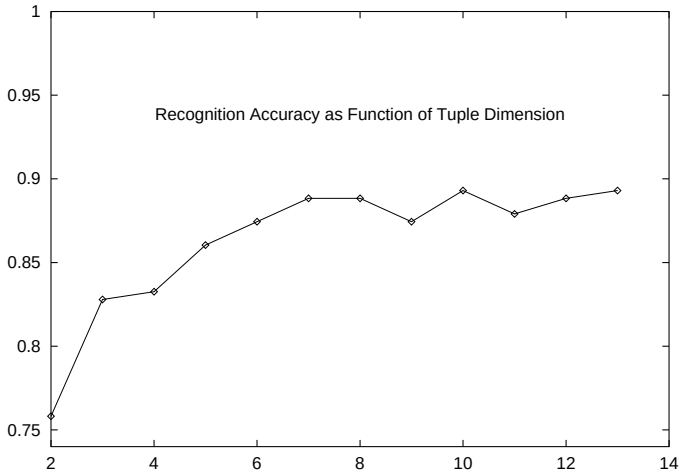


Figure 21: Correlation between recognition accuracy and tuple dimension ($m = 10$).

Number of n-tuples	Recognition Accuracy
1	76.3%
5	87.9%
10	89.3%
15	89.8%
20	90.2%

Table 6: Correlation between recognition accuracy and size of tuple set ($n = 10$).

The best recognition performance for the n-tuple classifier on this set of sample data is 90.2% at $n = 10$ and $m = 20$. Generally speaking, a large n tends to decrease the false alarm rate while increasing the mis-classification rate. Similarly, a large m also tends to increase the discrimination power of the classifier, although it also increases the computation needed. However, too many "not-so-good" tuples may also reduce the performance. How to chose tuples is currently an unresolved issue.

5 Other Related Work

There are many Web search engines now available. A number of these have become immensely popular. They typically operate using indices built by automated WWW scanners (called "robots"). As we observed earlier, this work has mainly concentrated on the query processing and text retrieval aspects of the problem, ignoring the issues of complex document analysis. Here documents are equated with the raw ASCII text that is readily extracted by parsing the HTML.

Even once it becomes possible to build complete working systems for extracting text from Web images, it is clear that recognition errors will be unavoidable. Indeed, this same problem arises in the indexing other multimedia datatypes as well (*e.g.*, speech, handwriting). Query mechanisms are needed that are robust in the presence of such "noise." One such approach, based on combining techniques from approximate string matching and fuzzy logic, is described in papers by Lopresti and Zhou [13, 16].

A large amount of basic IR research related to the World Wide Web is also taking place at universities and elsewhere. The Harvest system developed as an ARPA-funded project of the Internet Research Task Force, for example, is an integrated set of tools to gather, organize, and search information across the Internet [5]. Efficiency (*i.e.*, caching, replication, and scalability) is a primary concern. Harvest is provided with a number of "Summarizers" that can extract text from a variety of formats (*e.g.*, compressed tar archives, PostScript files), but none are capable of sophisticated document analysis. The Harvest architecture is general, however, and could be extended to incorporate such tools if they became available.

Work has also been done on developing methods to aid users in searching, filtering, and retrieving Web images. For example, Paek describes a system that combines visual features, text-related features, and the context of images within a document in an attempt to understand the image content of Web documents [19]. The text features in his system

are extracted from the URL and ALT tags on the Web page though, not from the images themselves.

Lastly, we note that researchers have begun to examine the problems associated with automatically processing paper documents for placement on the Web. Taghva, Condit, and Borsack describe the Autotag system, which attempts to capture a document's logical structure and incorporate it in the electronic version through the use of SGML tags [23]. Nagy, Seth, and Viswanathan propose a similar approach and present a thoughtful study of the various issues associated with automated document conversion [18].

6 Conclusions

In this paper, we have explored the problems of locating and recognizing text in WWW images. The rapid growth of the World Wide Web and its natural evolution towards more sophisticated uses of multimedia is creating significant opportunities as well as new challenges for document analysis as it relates to information retrieval. Although it may seem somewhat anachronistic at first, the need for OCR still remains.

After a brief survey of Web image formats, we described a method for locating and extracting text from color GIF images. Our technique employs a color clustering algorithm that attempts to group colors from the same object together based on both RGB and spatial proximities. Geometric as well as text layout features are then used to distinguish text-like from non-text-like components. Experimental results show that the method is promising, although there is also much room for improvement.

For example, better measures of "goodness" need to be developed. The current algorithm uses a measure based solely on character alignment (the variation in the heights of components). Since text detection is intended to be followed by recognition, a criterion that reflects the quality of the character images would be useful. To recover broken or missing characters, an iterative process could be introduced into the algorithm. After completing the final stage of text detection, a refined clustering procedure might be applied to areas where text has been located to ensure the integrity of the characters.

We then proceeded to describe two methods for recognizing text from Web images. The first uses a polynomial surface fitting technique. The second is based on a "fuzzy" adaptation of the classical n-tuple classifier. Both methods exploit the information contained in color bits to compensate for low spatial sampling resolution (typically 72 ppi).

Preliminary experimental results show that the surface fitting method performs rather poorly. We found that Web images often use only a few shades of color within a given character. A continuous surface representation does not appear to be suitable for these images because of the high frequency intensity changes.

On the other hand, preliminary results show that the fuzzy n-tuple method works quite well. The major difficulty in using this approach is the selection of desirable n-tuples. It has been shown that in general, generating distinguishing tuples is a computationally intractable problem. Despite this, practical search strategies have been developed for finding "good" tuples. For example, in a paper by Jung, Krishnamoorthy, Nagy, and Shapira, the authors propose two algorithms for generating, from a small training set, a collection of n-tuples

which are "shift" invariant [7]. Such n-tuples fit each positive pattern in at least p different shift positions and fail to fit each negative pattern by at least $n - q$ pixels in each shift position. "Shift invariance" is a desirable feature since this makes the tuples independent of sample variations.

As we mentioned earlier, text in Web images usually undergoes relatively few types of distortion. This makes the n-tuple method particularly applicable to the Web OCR problem. A deeper understanding of the nature of these distortions may lead to the development of a practical n-tuple generator for Web image OCR. It is also evident that gathering training and testing data for building these specialized classifiers requires much effort. This explains the small number of classes we were limited to in the experiments of Subsection 4.3. Just as standard databases are now becoming popular in traditional document analysis, work on Web image analysis would be greatly facilitated by amortizing the costs of assembling test sets across the entire research community.

Better methods are also needed for performance evaluation. Ideally, OCR would be run on the results of text detection and the performance of the entire end-to-end system measured. However, this is itself a challenge that requires solving several more hard problems beyond what we have described in the current paper (*e.g.*, noise filtering, character segmentation).

We would be remiss not to mention that an important factor in the proliferation of image text on the Web has been the lack of control designers have over typography in standard HTML. Recent extensions to HTML, however, promise to offer much more flexibility in this regard (*e.g.*, [27]). A number of companies have banded together to develop standards for directly supporting anti-aliased fonts, for example. Hence, one might argue that a key reason for wanting to embed text in images will soon diminish, and therefore there is no need to worry about addressing the problem. To believe this reasoning, however, is to believe that the Web will become less multimedia-oriented in the future. This does not seem likely; if anything, use of the Web to deliver image data will continue to grow.

Finally, we note that locating and recognizing text in images is only one class of document analysis problems that have important implications for information retrieval on the Web. The range of document and graphics structures present online mirrors what researchers have studied on paper for decades. Table 7 list some such examples. The degree to which existing techniques can be adapted or completely new methods are required is an open question.

Topic in Document Analysis	Example WWW Application
Chemical graphics	http://www.chemie.fu-berlin.de/chemistry/bio/amino-acids.html
Engineering drawings	http://www.research.digital.com/SRC/juno-2/flange.html
Equations	http://aero.stanford.edu/OnLineAero/Acoustics.html
Maps	http://www.gpsy.com/maps
Musical scores	http://www-personal.umich.edu/~mhopkins/etude1.html
Org charts	http://www.acm.org/sigchi/listserv/chiemail.htm
Photographs	http://www.phillynews.com

Table 7: Possible WWW applications for document analysis.

7 Acknowledgements

The research described in this paper was performed while the authors were at the Matsushita Information Technology Laboratory in Princeton, NJ. We would like to thank Zhibin Lei and Tolga Tasdizen for their contributions to this work. We would also like to thank the editors of this special issue for their support and guidance, and the anonymous reviewers for their helpful suggestions to improve the quality of the presentation. The trademarks mentioned and/or depicted in this paper are the properties of their respective owners.

References

[1] Alta Vista. http://www.altavista.com.

[2] W. Bledsoe and I. Browning. Pattern recognition and reading by machine. In *Proceedings of the Eastern Joint Computer Conference*, number 16, pages 225–233, December 1959.

[3] D. Doermann, E. Rivlin, and I. Weiss. Logo recognition. Technical Report CAR-TR-688, Document Processing Group, Center for Automation Research, University of Maryland, College Park, MD 20742-3275, October 1993.

[4] R. Graham and P. Hell. On the history of the minimum spanning tree problem. *Annals of the History of Computing*, 7, 1985.

[5] Harvest Web Indexing. http://www.tardis.ed.ac.uk/harvest.

[6] Q. Huang, B. Dom, D. Steele, J. Ashley, and W. Niblack. Foreground/background segmentation of color images by integration of multiple cues. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 246–249, 1995.

[7] D. Jung, M. Krishnamoorthy, G. Nagy, and A. Shapira. N-tuple features for OCR revisited. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(7):734–743, 1996.

[8] D. Lee, T. Pavlidis, and G. W. Wasilkowski. A note on the trade-off between samping and quantization in signal processing. *Journal of Complexity*, 3:359–371, 1987.

[9] Z. Lei, D. Keren, and D. Cooper. Computationally fast Bayesian recognition of complex objects based on mutual algebraic invariants. In *Proceedings of the International Conference on Image Processing*, pages 635–638, Washington, D.C., October 1995.

[10] C. M. Leung. A practical basis set for Chinese character recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, San Francisco, CA, June 1985.

[11] H. Li, O. Kia, and D. Doermann. Text enhancement in digital video. In *Document Recognition and Retrieval VI (IS&T/SPIE Electronic Imaging'99)*, volume 3651, pages 2–9, San Jose, CA, January 1999.

[12] R. Lienhart. Automatic text recognition for video indexing. In *Proceedings of ACM Multimedia'96*, pages 21–30, Bosten, MA, November 1996.

[13] D. Lopresti. Robust retrieval of noisy text. In *Proceedings of the Third Forum on Research and Advances in Digital Libraries*, pages 76–85, Washington, DC, May 1996.

[14] D. Lopresti, G. Nagy, P. Sarkar, and J. Zhou. Spatial sampling effects in optical character recognition. In *Proceedings of the Third International Conference on Document Analysis and Recognition*, pages 309–314, Montréal, Canada, August 1995.

[15] D. Lopresti and J. Zhou. Document analysis and the World Wide Web. In *Proceedings of the IAPR Workshop on Document Analysis Systems*, pages 651–669, Malvern, PA, October 1996.

[16] D. Lopresti and J. Zhou. Retrieval strategies for noisy text. In *Proceedings of the Fifth Annual Symposium on Document Analysis and Information Retrieval*, pages 255–269, Las Vegas, NV, April 1996.

[17] D. Lopresti and J. Zhou. Locating and recognizing text in WWW images. In *Proceedings of the Symposium on Document Image Understanding Technology*, pages 193–201, Annapolis, MD, April-May 1997.

[18] G. Nagy, S. Seth, and M. Viswanathan. DIA, OCR, and the WWW. In H. Bunke and P. Wang, editors, *Handbook of Character Recognition and Document Image Analysis*, pages 729–754. World Scientific, Singapore, 1997.

[19] S. Paek. Detecting image purpose in World-Wide-Web documents. In *Document Recognition V (IS&T/SPIE Electronic Imaging'98)*, volume 3305, pages 151–158, San Jose, CA, January 1998.

[20] F. P. Preparata and M. I. Shamos. *Computational Geometry – An Introduction*, chapter 5. Springer-Verlag, 1985.

[21] P. Sarkar, G. Nagy, J. Zhou, and D. Lopresti. Spatial sampling of printed patterns. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(3):344–351, March 1998.

[22] Search Engine Watch. http://searchenginewatch.com.

[23] K. Taghva, A. Condit, and J. Borsack. Autotag: A tool for creating structured document collections from printed materials. Technical Report 94-11, UNLV Information Science Research Institute, Las Vegas, NV, December 1994.

[24] G. Taubin. Estimation of planar curves, surfaces and nonplanar space curves defined by implicit equations, with applications to edge and range image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1115–1138, November 1991.

[25] L. Wang and T. Pavlidis. Detection of curved and straight segments from gray scale topography. In *Proceedings of the SPIE Symposium on Character Recognition Technologies*, pages 10–20, San Jose, CA, 1993.

[26] L. Weinman. *Designing Web Graphics.* New Riders Publishing, Indianapolis, IN, 1996.

[27] WebFont Wizard. http://webreview.com/wr/pub/1999/02/19/feature/index.html.

[28] World Wide Web Consortium. *Workshop on High Quality Printing from the Web*, Cambridge, MA, April 1996.
http://www.w3.org/pub/WWW/Printing/Workshop_960425.html.

[29] V. Wu, R. Manmatha, and E. Riseman. Finding text in images. In *Proceedings of Second ACM International Conference on Digital Libraries*, pages 23–26, Philadelphia, PA, July 1997.

[30] C. T. Zahn. Graph theoretical methods for detecting and describing gestalt clusters. *IEEE Transactions on Computers*, 20(1), 1971.

[31] R. H. Zakon. Hobbes' Internet timeline v4.2.
http://www.isoc.org/zakon/Internet/History/HIT.html.

[32] Y. Zhong, K. Karu, and A. Jain. Locating text in complext color images. *Pattern Recognition*, 28(10):1523–1535, 1995.

[33] J. Zhou and D. Lopresti. Extracting text from WWW images. In *Proceedings of the Fourth International Conference on Document Analysis and Recognition*, pages 248–252, Ulm, Germany, August 1997.

[34] J. Zhou, D. Lopresti, and Z. Lei. OCR for World Wide Web images. In *Document Recognition IV (IS&T/SPIE Electronic Imaging'97)*, volume 3027, pages 58–66, San Jose, CA, February 1997.

[35] J. Zhou, D. Lopresti, and T. Tasdizen. Finding text in color images. In *Document Recognition V (IS&T/SPIE Electronic Imaging'98)*, volume 3305, pages 130–140, San Jose, CA, January 1998.