Kolmogorov Complexity-Based Ideas for Locating Text in Web Images

Martin Schmidt, Vladik Kreinovich, and Luc Longpré
Department of Computer Science, University of Texas at El Paso, El Paso, TX 79968 USA
emails {mschmidt,vladik,longpre}@cs.utep.edu

***Abstract*—The gaining popularity of the World Wide Web increases security risks. Search tools monitor plain text in web pages, but search for text in graphical images is still difficult. For this search, we use the fact that the compressed images with text have different size than images without text.**

I. Practical Problem

The gaining popularity of the World Wide Web also means increasing security risks. As the World Wide Web has become an affordable way for different political groups to reach a broad audience it is becoming harder to monitor all these web sites for their content. While numerous web search tools can be used to automatically monitor plain text in web pages, search for text in graphical images is still a considerable challenge. This fact is used by designers of such web pages who "hide" their text by placing it inside of graphical images, avoiding detection from regular search engines. At present, the only known way to find all occurrences of suspicious words like "terror" in images is to use character recognition to find and read all the texts in all the images. Performing such a character recognition is still a very computational intensive task which takes a long time, and has to be done for every image, because the only known way to check whether the image contains text is to apply a character recognition to this image.

II. Idea

A. Main Idea: We Must Consider Simple Images

The object of hiding the text inside an image is to make it hard for search engines to detect it, but not hard for the user to read. Therefore, the image used as a background for the text should be very simple. It should not overpower the text at any point, making parts of the text hard to read, or draw attention away from the text itself. Also, in order not to be distracting from the text, the background should be reasonable homogeneous. A good example is a background created by Windows 98, where a small image, called a *tile*, which is repeated (as tiles on a wall are) to cover the entire screen.

The necessity to make the text easily seen also limits the use of colors. Colors enhance the image, but many people do not perceive the color correctly, and it is known that even for those who do, it is much easier to find a piece of the image which is of different brightness than a piece of different color [1]. Therefore, if we want a text to be easily visible, we should make it stand out in a black-and-white (grey scale) version of the image.

So, to detect images with text, we should look for simple, homogeneous black-and-white images with text superimposed on them.

For such simple images, adding a text increases their complexity. So, if an image is mostly simple, but has one area which is more complex, this is a good indication that this area may contain text.

We want to use this idea to find images which are likely to contain texts, and to locate the text in these images.

B. When Is an Image Simple? Kolmogorov Complexity and Compression Size

To use this idea, we must be able to tell when an image (or a part of the image) is simple and when it is not. When is an image simple? Intuitively, if an image that we see on the screen can be described by a simple formula (e.g., a circle, a line, etc.), or the image is a combination of several subimages describable by simple formulas, then we consider this image simple. On the other hand, if there is no simple formula which describes the image, i.e., the only way to describe this image is to list the intensities of all the pixels, then the image is clearly complex. Thus, we can define a *complexity* of an image as the smallest size of a program which generates this image. This notion was introduced in the 1960s by G. Chaitin, A. Kolmogorov, and R. Solomonoff, and it is called *Kolmogorov complexity* $K(x)$ (see, e.g., [2]). In terms of Kolmogorov complexity, we can say that an image is simple if its Kolmogorov complexity is small.

Unfortunately, it has been proven that Kolmogorov complexity is not algorithmically computable; so, we cannot simply compute the *exact* values of Kolmogorov complexity and check whether an image is simple or not. Instead, we should use *approximate* values of Kolmogorov complexity.

This notion of an approximate Kolmogorov complexity may seem very theoretical, but, as we will show, it can be naturally reformulated in image-related terms. Indeed, as we have mentioned earlier, when we say that an image is simple, we mean that instead of describing the image by its bitmap, i.e., by the intensities of all the pixels, we can describe this very image by storing a very small amount of data (e.g., the radii of the circles if we have an image consisting of circles). In other words, an image is simple if it can be *compressed* into a small size file from which this original image can be losslessly decompressed. Similarly, our description of what it means for a image to be complex means that we cannot compress its bitmap description into any file of smaller size (or, to be more precise, any such compression would be *lossy* in the sense that it will not be able to reconstruct the original image exactly). Thus, a Kolmogorov complexity of an image can be viewed as the smallest possible size of the image's compression.

To determine the exact value of Kolmogorov complexity, we must use the "best" possible compression. the fact that Kolmogorov complexity is not algorithmically computable means that it is not possible to design the "best" image compression algorithm: whichever image compression algorithm we use, there is always a possibility of improvement.

The better the compression algorithm, the more the size of the compressed file approximates Kolmogorov complexity. So, to check whether an image is simple or not, we will use the size of this image under the best known compressions.

Of course, selecting "the best" compression algorithm of the many existing ones is not a precisely formulated task. However, in our selection, we can use the fact that informally, people solve this task all the time by selecting a compression method for their images. So, a natural selection of a compression method would be to use most widely used compression technique. Since we are talking about the web, we therefore want to choose a technique which is most widely used on the web. The two most widely used compression techniques used to post images on the web are `jpg` and `gif`. The tendency right now is to use fewer `gif` files and more `jpg` ones, for three reasons:

- `gif` uses insufficiently many colors (256);
- *gif* is proprietary, while `jpg` is freely available;
- lately, special chips are being developed that hardware support `jpg` compression and decompression, thus making this format much faster in use than formats which require software to compress and decompress.

So, in this paper, we use, as a measure of the image's complexity, the size of its `jpg` compression.

To confirm that adding a text to a simple image affects the size of the `jpg`, and to see how exactly it affects, we performed several numerical experiments.

III. Experiment

A. Description and Results of the Experiment

In our experiment, we tested two different simple backgrounds: one a very simple design and the second one a more complex background design (tile background).

On a 600×400 pixel background, we inserted text `Large text` formed with letters of different size into the top left quadrant (of size 300×200). We used Courier New Font of different pixel size for the letters.

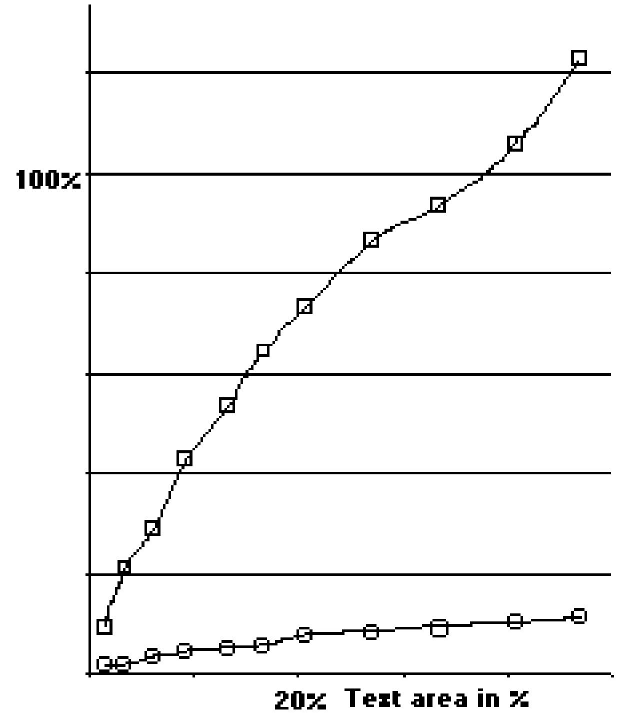
We then calculate the size of the corresponding `jpg` file, and analyzed how this size depends on the relative area of the image.

In this experiment, we used the texts of the following sizes:

height	length	area	
		absolute	relative
0	0	0	0%
10	90	900	1.50%
15	130	1,950	3.25%
20	180	3,600	6.00%
25	220	5,500	9.17%
30	265	7,950	13.25%
35	285	9,975	16.63%
40	310	12,400	20.67%
45	360	16,200	27.00%
50	400	20,000	33.33%
55	445	24,475	40.79%
60	470	28,200	47.00%

For each text size, the absolute `jpg` size of a corner with text, and the relative increases of this sizes over the size of the original `jpg` background corner (without text) are presented in the table and in the graph:

	simple image		tile image	
relative area	absolute size	relative size	absolute size	relative size
0%	5,653	0%	23,391	0%
1.50%	6,154	8.86%	23,756	1.56%
3.25%	6,832	20.86%	23,813	1.80%
6.00%	7,288	28.92%	24,100	3.03%
9.17%	8,062	42.61%	24,341	4.06%
13.25%	8,667	53.32%	24,515	4.81%
16.63%	9,313	64.74%	24,689	5.55%
20.67%	9,794	73.25%	25,124	7.41%
27.00%	10,523	86.15%	25,264	8.01%
33.33%	10,928	93.31%	25,430	8.72%
40.79%	11,623	105.61%	25,775	10.19%
47.00%	12,585	122.63%	26,005	11.18%



B. Analysis of the Experimental Results

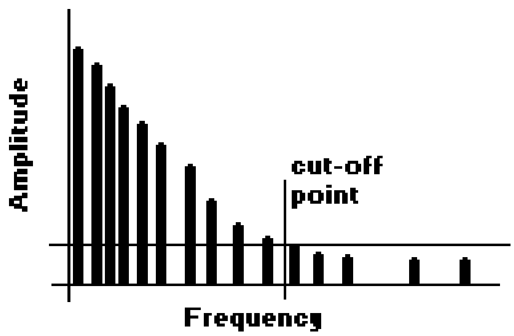
We see that adding text to images does increase the file size. In order to use this increase for text detection, we would like to know how exactly the increase in the file size depends on the text size; if we know this dependence, then, from the sizes of the `jpg` files, we will be able not only to tell that the image probably contains a text, but also estimate the size of this probable text. Knowing the approximate size would definitely help in actually detecting the text.

Our experimental data shows that in both cases (and in several other cases not presented in this paper) the dependence of the file increase I on the relative text area A can be describe by a piece-wise linear formula:

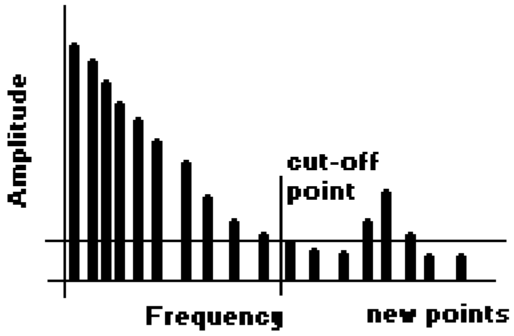
- for $A \le 20\%$, we have $I = k \cdot A$ for some coefficient k depending on the complexity of the background, and
- for $A > 20\%$, we have $I = \frac{2}{3} k \cdot A$.

To explain this dependence, we must recall how `jpg` compression works. We will give a very simplified description of this compression (see, e.g., [1] for a more detailed and more accurate description). However, even this simplified description enables us to qualitatively explain the observed dependence.

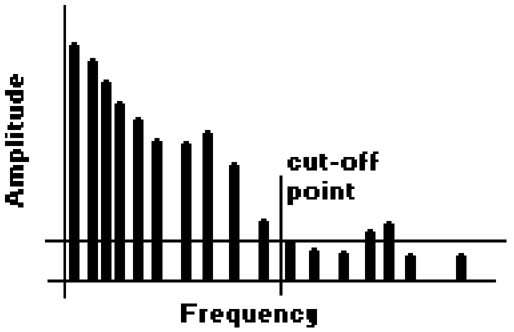
The `jpg` compression is based on the application of Discrete Cosine Transform (DCT) to different image components. Usually, for large spatial frequencies, the amplitude of DCT decreases with frequency; so, for large spatial frequencies, the amplitudes are close to 0. In `jpg` compression, the values of the amplitude which are below a certain pre-defined level, are cut off. This cut off is the main reason why the size of the `jpg` compression is smaller than the size of the original bitmap image.



A piece of text which has a small size corresponds to large values of spatial frequencies. So, when we add a small size text to the image, we increase the amplitude corresponding to these high frequencies. Thus, the coefficients which were originally cut off because of their small size become larger than the pre-defined level, and thus, not cut off any more. Thus, the size of the `jpg` file increases.



On the other hand, larger size details correspond to smaller spatial frequencies. Therefore, when we add a large size text to the image, we mainly increase the amplitudes corresponding to the spatial frequencies which were not originally cut off. Thus, adding this text to the image does not lead to such a drastic increase in the size of the resulting `jpg` file.



IV. How We Can Use These Experimental Results: Binary Search Idea

Due to these experimental result, if we have a simple image in which different quadrants have different `jpg` sizes, this means that probably, the quadrant of a larger size contains text.

This conclusion enables us not only to check that there is a text somewhere, but also to *locate* this text within a specific quadrant. We can get an even better localization of the text if we further subdivide this quadrant into four parts and by comparing the `jpg` sizes of different parts, find the part most probable to contain the text.

The resulting "binary search" algorithm does not work if the text is equally spread between several quadrants (e.g., if it is in the center of the upper half of the screen). To detect such texts, instead of dividing the screen into four non-intersecting quadrants, we can use intersecting "quadrants" each of which starts at the corresponding corner of the screen and has a size $\frac{2}{3} \times \frac{2}{3}$ of the original screen (for details, see [3]).

V. Future Research

In our experiments, we only tested how the size of the compressed image depends on the area containing the text. In reality, the compressed file size depends not only on the area, but also on the size of the letters forming the image, probably on the font used in this image, etc.

For example, for the same text size, is we use smaller letters, this means that we are changing amplitudes corresponding to higher spatial frequencies and therefore, the increase in the `jpg` size would be larger. Similarly, if we use a font with lots of small details, these details correspond to higher spatial frequencies and therefore, we expect that this font will lead to a larger increase in `jpg` size than a more plain font like the Courier font that we used.

In the future, we are planning to investigate how these factors affect the `jpg` size and thus, to get a better understanding of what type of a text we should expect in a simple image in which several parts have drastically different `jpg` sizes.

Acknowledgment

This work was supported in part by NASA under cooperative agreement NCC5-209, by NSF grants No. DUE-9750858 and CDA-9522207, by the United Space Alliance, grant No. NAS 9-20000 (PWO C0C67713A6), by the Future Aerospace Science and Technology Program (FAST) Center for Structural Integrity of Aerospace Systems, effort sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant number F49620-95-1-0518, and by the National Security Agency under Grants No. MDA904-98-1-0564 and MDA904-98-1-0564.

We would like to thank the members of the ELMOS research group, at the Computer Science Department at the University of Texas at El Paso for their helpful collaboration in this research.

References

[1] B. Furht, S. Smoliar, and H. Zhang, *Video and image processing in multimedia systems,* Norwell, Mass.: Kluwer, 1995.
[2] M. Li and P. Vitányi, *An introduction to Kolmogorov complexity and its applications,* New York: Springer Verlag, 1993.
[3] M. Schmidt, V. Kreinovich, and L. Longpré, "Kolmogorov complexity-based ideas for locating text in web images", *Complexity Conference Abstracts 1999*, Abstract No. 98–14, p. 16, April 1999.
[4] S. E. Umbaugh, *Computer vision and image processing,* Upper Saddle River, NJ: Prentice Hall, 1998.