Title

Feature Monitored Shape Unifying for Lossy SPM-JBIG2

Journal

Signal Processing and its Applications, Sixth International, Symposium on. 2001, 2

Authors

Ye, Y

Cosman, P

Publication Date

2001

Peer reviewed

International Symposium on Signal Processing and its Applications (ISSPA), Kuala Lumpur, Malaysia, 13 – 16 August, 2001. Organized by the Dept. of Microelectronics and Computer Engineering, UTM, Malaysia and Signal Processing Research Centre, QUT, Australia

FEATURE-MONITORED SHAPE UNIFYING FOR LOSSY SPM-JBIG2

Yan Ye and Pamela Cosman

University of California at San Diego
Electrical and Computer Engineering Department
9500 Gilman Drive, La Jolla, CA 92093-0407
E-mail: {yye, pcosman}@ucsd.edu

ABSTRACT

Shape unifying is a very efficient preprocessing technique used in lossy SPM-JBIG2 systems. It permits isolated errors between the current bitmap and its reference to improve refinement coding efficiency. Compared to lossless coding, it can improve compression by about 32% while causing very little visual information loss. When bigger error clusters are permitted in shape unifying, further compression gain can be achieved but at the price of more noticeable visual information loss and even character substitution errors. In this paper we propose a feature monitored shape unifying procedure that can significantly lower the risk of substitution errors when permitting bigger errors. Experiments show that, compared to the unmonitored shape unifying, the feature monitored version can suppress more than 2/3 of all substitution errors while achieving additional compression improvements of 30-40%.

1. INTRODUCTION

The JBIG2 standard [1, 2] is the new international standard for *lossless* and *lossy* coding of bi-level images. It is meant for both text and halftone data. We only consider text images. In JBIG2, coding of text is based on either of two modes: *pattern matching and substitution* (PM&S) [3] or *soft pattern matching* (SPM) [4, 5]. We concentrate on SPM.

A typical page of text contains many repeated characters. We call the bitmap of a text character instance a "symbol." To code all the symbols in the image, we first select a group of representatives and put them into the dictionary. In lossless SPM, a symbol is coded by giving its position on the page, the index of the best matching symbol in the dictionary, and finally a lossless coding of the current symbol's actual bitmap based on that of its best match. This lossless coding, called refinement coding, is done by context-based arithmetic coding using a context drawn from both the best match bitmap, and the already coded part of the actual current bitmap. In our work we use the Hamming distance based matching criterion.

It is a difficult task to evaluate image quality for lossy compression of binary text images. Differences between the bitmaps of the original and compressed symbol can be categorized as isolated differences, clustered differences, or substitution errors. Isolated differences are tiny (1x1, 1x2, 2x1) groups of pixels which are usually not noticeable visually. Clustered differences are larger groups of pixels. Substitution errors occur when one text character (say, a letter "c") in the original image gets reconstructed as a different one (say, a letter "o") in the compressed image. In this paper, we are concerned with compression algorithms that increase the compression ratio while limiting the number of substitution errors, and also allowing strict control on the size of visible clustered errors.

In lossy coding, SPM preprocesses the original image to introduce information loss. We introduce a feature monitored shape unifying procedure that can effectively suppress the occurrences of character substitutions in lossy SPM. Although the visual appearance of decoded images degrades when bigger error clusters are permitted, character substitutions are kept low by ensuring certain features do not change with shape unifying.

This paper is arranged as follows. In Section 2, we first briefly review the three preprocessing techniques used in lossy SPM. Then we explain the idea of using geometrical features of binary patterns to monitor shape unifying so as to lower the risk of substitution errors. In Section 3, we present our experimental results. We conclude in Section 4.

2. FEATURE MONITORED SHAPE UNIFYING

2.1. Preprocessing techniques in lossy SPM

To achieve lossy compression with SPM, [4] proposed several preprocessing techniques to introduce information loss in a restricted manner. These techniques are *speck elimination*, *edge smoothing* and *shape unifying*. Speck elimina-

This research was supported by NSF grant MIP-9624729 (CAREER), and by the Center for Wireless Communications at UCSD.

0-7803-6703-0/01/$10.00©2001 IEEE

tion wipes out very tiny symbols or flying specks (symbols no bigger than 2×2). Edge smoothing fixes jagged edges by flipping protruding single black pixels or indented single white pixels along text edges. Shape unifying (see Figure 1) tries to make the current symbol bitmap as similar as possible to its reference bitmap, without introducing "visible" changes. This is achieved by flipping pixels in the current bitmap if they are isolated areas of difference with the reference bitmap. We use the term "isolated" to mean a 1×1, 1×2, or 2×1 block of pixels. The modified bitmap is then losslessly coded with refinement coding.

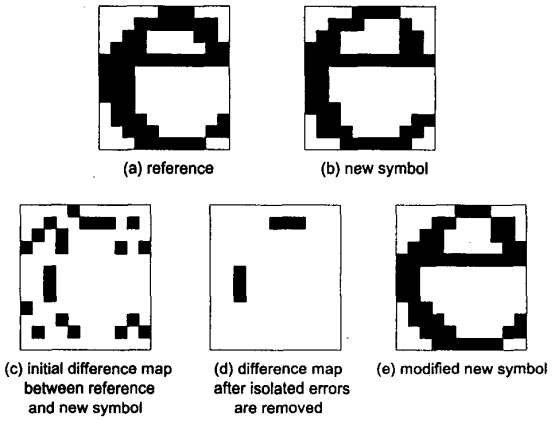


Figure 1: An example of shape unifying.

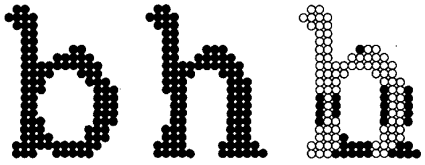
Among the three techniques, shape unifying gives the biggest compression improvement. Our experiments show that, compared to strictly lossless coding, using speck elimination and edge smoothing together can improve compression by around 8%, while adding shape unifying provides approximately 40% improvement. Shape unifying is so efficient because in refinement coding, the arithmetic coder predicts the current bitmap using information from its reference. Differences between these two bitmaps reduce the prediction accuracy and consequently refinement coding efficiency. Shape unifying reduces the number of different pixels between the two bitmaps. After shape unifying, the modified symbol (Fig. 1 (e)) is more similar to its reference (Fig. 1 (a)). Furthermore, due to the "isolated" restriction, the decoded image will contain only isolated errors compared to the original image. Therefore, substitution errors are very unlikely to occur except for very tiny symbols. The loss of visual information is almost imperceptible.

Speck elimination and edge smoothing are truly preprocessing techniques. Shape unifying is not strictly a preprocessing procedure since it is not done until after symbols find their references in the dictionary, which, using the more efficient dictionary design techniques we have previously proposed [5, 6], will not happen until the entire dictionary itself is decided. Once the entire symbol set is altered with these three approaches, the page is compressed losslessly; no further loss will be introduced.

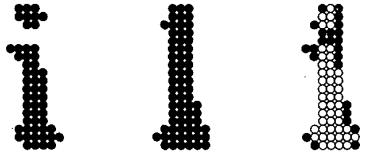
2.2. Feature monitored shape unifying

The advantage of permitting only isolated errors in shape unifying is that visual information loss at the receiver's end is almost imperceptible. However, such a restriction also puts a limit on the lossy coding efficiency. To improve the coding efficiency, shape unifying should allow not just isolated errors, but some clustered ones as well, as long as the risk of character substitutions is kept low. To limit the risk of substitutions, we introduce the idea of monitoring the shape unifying procedure with features.

We can compute geometrical features (e.g., centroid, number of holes, number of connected components) for binary patterns [7, 8]. If two bitmaps represent the same character, then their features will have the same or very similar values; otherwise, their features are likely to be different even though they may look somewhat similar. In Fig. 2 we show two pairs of similar characters with different features and the difference maps between them. In our experiments, these character pairs will very likely find each other as the reference symbol because their Hamming distances are rather small. Note that the character "b" has one internal hole while the character "h" has none; "i" has two connected components (the dot and the stem) while "l" has only one. Such features are essential for a human being to distinguish between "b" and "h" or "i" and "l." Thus, we can use the number of holes and the number of connected components to monitor the shape unifying procedure. That is, for each cluster of differences between the current bitmap and its reference, regardless of whether it is small or not, if eliminating it will not cause the features to change abruptly, we go on with shape unifying and eliminate this difference cluster; otherwise, we preserve it because a substitution error will likely occur. For example, in Fig. 2 (a), we can change the "b" bitmap not only at the isolated single-pixel location, but at all the gray pixel locations, as they will not cause the one internal hole in "b" to disappear. But, the 10-pixel cluster of differences down at the bottom (painted in black) must be preserved. Otherwise a human being will perceive an "h" instead of a "b". Similarly, in Fig. 2 (b), we can change the "i" bitmap at all the gray locations but not at the black ones as changing the black locations will cause the "i" bitmap to be connected into one whole piece, resulting in a substitution error. Modifying the current symbols at more locations makes symbols more similar to their references. Consequently, the refinement coding efficiency can be improved. At the same time, ensuring certain features do not change abruptly allows us to prevent many cases of character substitutions.



(a) characters "b" and "h" and the differences between them



(b) characters "i" and "l" and the differences between them

Figure 2: Examples of similar symbols with different features. The rightmost figures show the difference bitmaps. Black error clusters cannot be ignored because otherwise the features will change. Gray error clusters can be ignored.

3. EXPERIMENTAL RESULTS

In this section we show how the feature monitored shape unifying can improve the coding efficiency of lossy SPM systems while effectively suppressing substitution errors. Our test set contains six images from two sources: two 200-dpi CCITT standard images (f01 and f04); and four 300-dpi images (IG0H, J00O, N03F, N03H) from the University of Washington Document Image Database I [9].

In our experiments, we use two features, number of holes and number of connected components, to monitor the shape unifying procedure. According to our experiments, in English and other Latin-based languages (image f04 contains French text), characters that are most easily confused under the Hamming distance based matching criterion are "b" and "h", "i" and "l", "u" and "n", "c" and "o" and "e", etc.. Clearly, the feature number of holes can distinguish between "b" and "h", "c" and "o", and "c" and "e"; the feature number of connected components can distinguish between "i" and "l". It is less obvious but true that these two features can also help prevent confusion between "u" and "n" and "e" and "o". Take "u" and "n" for example. In comparing the bitmaps of "n" and "u", there are basically two areas where a substantial number of clustered pixels differ: the center top and the center bottom. Modifying the upper cluster of pixels in the "n" bitmap to match the "u" bitmap will cause the "n" to split into two separate connected components. Monitoring based on number of connected components will prevent this. Likewise, modifying the lower cluster of pixels in the "n" bitmap to match the "u" will cause the lower opening in the "n" to close and will generate one internal hole. Monitoring based on the number of holes will prevent this. This way a substitution error is prevented. Furthermore, experiments show that an error cluster that is too big should not be permitted even if it would not change the binary bitmap's features. Otherwise, visual information loss caused by shape unifying will be too significant and the reconstructed image quality will become objectionable; it will contain a large number of distorted text characters in addition to substitution errors. Conceptually, a bigger symbol can tolerate a bigger error cluster. Therefore, we set the error size threshold to be proportional to the symbol size, i.e., error clusters smaller than a certain percentage of the symbol size are deemed ignorable. These smaller error clusters are put through the feature monitoring process to test if modifying the current bitmap at those locations will change the features.

In Table 1 we show results for three error size thresholds, 2%, 4% and 6% of the symbol size. We compare the coded file sizes and percentages of substitution errors for shape unifying with and without feature monitoring. At a low error threshold of 2%, the monitored version of shape unifying is not very different from the unmonitored version. Using the more restricted shape unifying described in Section 2.1, the average compressed file size for the test set is 17,030 bytes. The monitored and unmonitored versions at a threshold of 2% both achieve an additional coding gain of about 32%. At this threshold, the monitored and unmonitored versions also both suffer very rare substitution errors; the highest number of character substitutions observed is 3 for image f04, which is a dense text image with over 4,000 characters. As the error threshold goes up, the feature monitored version begins to show a clear advantage over the unmonitored version. For error thresholds of 4% and 6%, feature monitoring can successfully restrict the occurrences of substitutions to less than 1/3 of that of the unmonitored version. We note that besides character substitutions (e.g., a "b" turned into an "h"), shape unifying sometimes also generates "garbage" symbols which are not meaningful characters. These "garbage" symbols are also counted as substitutions and included in the numbers shown in Table 1. Interestingly, at an error threshold of 6%, the compressed file size of the monitored version seems to have already hit a bottom. This does not hold true for the unmonitored version; its compression improvement slows down but does not stop. The feature monitored version will refrain from making many bigger errors since they will most likely change the features, while the unmonitored version does not have this constraint and hence will go on with shape unifying, resulting in higher coding efficiency but also more substitution errors. Fig. 3 compares the monitored and unmonitored versions at different error thresholds by showing a portion of the original and the reconstructed images (from image N03H). Ignoring only isolated errors (Fig. 3 (b)) as described in Section 2.1 causes very little visual informa-

tion loss. In fact, the text appears to be even smoother due to edge smoothing. In Fig. 3 (c)-(h), we compare the monitored and unmonitored shape unifying under different ignorable error thresholds. Clearly, the monitored version prevents the "u" from turning into an "n" at all thresholds.

flip. thres.	w/ feature		w/o feature	
	size	subst.	size	subst.
2%	11,665	0.04	11,452	0.05
4%	10,073	0.42	9,666	1.34
6%	9,757	1.12	9,075	3.51

Table 1: Applying shape unifying with and without feature monitoring to a lossy SPM system. Compressed file sizes (in bytes) and percentages of substitution errors are shown. Results are averaged over six test images.

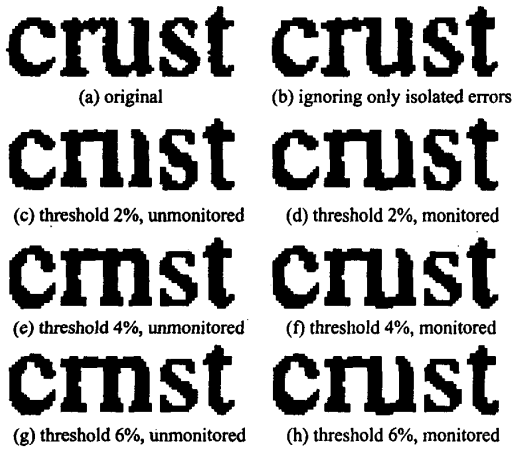


Figure 3: A portion of the original image N03H and the reconstructed images under different error thresholds.

Although feature monitored shape unifying can effectively lower the risk of substitution errors, it is more computationally demanding because every cluster of differences with size below the threshold has to be checked to see if this will result in different features. In Table 2 we list the average running times for lossy SPM with and without feature monitoring. We use the Unix command "time" to record the total execution time. The computer system used is a Pentium Pro 200MHz, running Red Hat Linux 6.0, with 64MB memory. Such a system is far from state-of-the-art, however, we can still obtain a valid comparison. Our code was not optimized for speed. On average the feature monitored shape unifying takes 40% more time to encode than its unmonitored counterpart.

flip. thres.	2%	4%	6%
w/ feature	141.3	137.8	137.3
w/o feature	103.0	98.7	97.7

Table 2: Average running time (in seconds) for lossy SPM with and without feature monitoring.

4. CONCLUSION

In this paper we proposed a feature monitored shape unifying procedure for lossy SPM systems. For bigger areas of differences between the current bitmap and its reference, the monitored shape unifying examines whether ignoring such differences will cause the features of the original bitmap to change. If not, these differences are ignored by modifying the current bitmap at corresponding locations. Otherwise, they are preserved. Experiments show that compared to the unmonitored shape unifying, the monitored version can effectively suppress 2/3 of all substitution errors. Furthermore, it improves the coding efficiency of lossy SPM by 30-40% compared to ignoring only isolated errors.

5. REFERENCES

[1] ISO/IEC JTC1/SC29/WG1 N1545. *JBIG2 Final Draft International Standard*, Dec. 1999.

[2] P. Howard, F. Kossentini, B. Martins, S. Forchhammer, W. Rucklidge, F. Ono. The Emerging JBIG2 Standard. *IEEE Trans. on Circuits and Systems for Video Technology*, pages 838–848, Vol. 8, No. 5, September 1998.

[3] R.N. Ascher and G. Nagy. Means for Achieving a High Degree of Compaction on Scan-digitized Printed Text. *IEEE Trans. on Computers*, Vol. 23, pages 1174–1179, Nov. 1974.

[4] P. Howard. Lossless and Lossy Compression of Text Images by Soft Pattern Matching. *Proc. 1996 IEEE Data Compression Conf. (DCC)*, pages 210–219, Snowbird, Utah, March 1996.

[5] Y. Ye and P. Cosman. Dictionary design for text image compression with JBIG2. To appear in *IEEE Trans. on Image Processing*, June, 2001.

[6] Y. Ye and P. Cosman. Fast and memory efficient JBIG2 encoder. To appear in *Proc. 2001 IEEE Intl. Conf. on Acoustics, Sound, and Signal Processing (ICASSP)*, Salt Lake City, Utah, May 2001.

[7] B. Horn. Robot Vision, Chapter 3. MIT Press. 1986.

[8] A. Jain. Fundamentals of Digital Image Processing, Chapter 9. Prentice Hall. 1989.

[9] E. S. Askilsrud, R. M. Haralick and I. T. Phillips. A quick guide to UW English Document Image Database I, version 1.0. CD-ROM. Intelligent Systems Lab, Univ. of Washington. August 1993.