FAST AND MEMORY EFFICIENT JBIG2 ENCODER

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

In this paper we propose a fast and memory efficient encoding strategy for text image compression with the JBIG2 standard. The encoder splits up the input image into horizontal stripes and encodes one stripe at a time. Construction of the current dictionary is based on updating dictionaries from previous stripes. We describe separate updating processes for the singleton exclusion dictionary and for the modified-class dictionary. Experiments show that, for both dictionaries, splitting the page into two stripes can save 30% of encoding time and 40% of physical memory with a small loss of about 1.5% in compression. Further gains can be obtained by using more stripes but with diminishing returns. The same updating processes are also applied to compressing multi-page document images and shown to improve compression by 8-10% over coding a multi-page document as a collection of single-page documents.

1. INTRODUCTION

The JBIG2 standard [1,2] is the new international standard for *lossless* and *lossy* compression of bi-level images. It is meant for both text and halftone data; a JBIG2 encoder is expected to segment an image into different regions [3] and use different coding mechanisms for text and for halftones. We only consider coding text images with JBIG2.

On a typical page of text, there are 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. We then code all the symbols by reference to their closest match in the dictionary. In our work we use the Hamming distance based matching criterion.

In JBIG2, coding of text is based on either of two modes: *pattern matching and substitution* (PM&S) [4] or *soft pattern matching* (SPM) [5]. We focus on SPM-based JBIG2 encoder design. Choosing a proper set of dictionary symbols is essential to coding efficiency. Previously we proposed two symbol dictionary design techniques called class-based design [6] and tree-based design [7]. In this paper we combine these two techniques to form a new technique: the modified-class design. This is more efficient than the class-based technique and simpler than the tree-based technique.

To save physical memory,JB IG2 allows the encoder to split a whole page image into horizontal stripes and process one stripe at a time. Since there is strong correlation between symbols on the same page, when coding the current stripe, the encoder can reuse some of the previous dictionary symbols from previous stripes. For this purpose, in JBIG2, at the end of each dictionary, the encoder sends a 1-bit flag for each dictionary symbol to tell the decoder if this symbol is to be retained or discarded after the current stripe is decoded. In this paper we propose a dynamic dictionary update procedure to retain useful dictionary symbols and discard obsolete ones. Because the encoder deals with fewer symbols at a time, this dynamic procedure is more memory efficient and faster. Furthermore, this procedure can be directly applied to compressing multi-page document images. We will show that using dictionaries from previous pages leads to 8-10% improvement in compression compared to treating the pages as single-page documents.

This paper is arranged as follows. In Section 2, we first propose the modified-class dictionary design. We then explain how to construct the current dictionary by dynamically updating previous dictionaries. In Section 3, we present our experimental results. In Section 4, we draw conclusions.

2. STATIC AND DYNAMIC DICTIONARY DESIGN

In this section we first briefly review the class-based and tree-based symbol dictionary design for JBIG2 encoders [6, 7]. Compared with simpler dictionary formation methods such as one-pass and singleton exclusion dictionaries, these two techniques can improve compression by up to 8% for lossless and 17% for lossy compression, respectively.

To form the one-pass dictionary, the encoder matches

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

0-7803-7041-4/01/$10.00 ©2001 IEEE

the current symbol with all previous symbols and encodes it by reference to its best match. Then it adds the new symbol to the dictionary. One-pass dictionaries contain many singletons which are symbols never referenced by any subsequent symbol [8]. They are detrimental to coding efficiency because dictionary indices are assigned to them unnecessarily, thus increasing the average length of all indices. By excluding singletons from the dictionary, we obtain the singleton exclusion (SE) dictionary. Bitmaps of the singletons are coded by reference to their best dictionary match when they occur on the page; this is referred to as embedded coding in text regions [1].

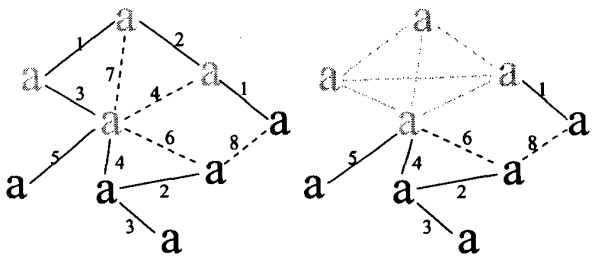
To design the class-based (CLASS) dictionary [6] we follow two steps. First we group all the symbols into *classes* by pointing them to their closest matches. For each class we choose one representative to go into the dictionary; all other symbols will be coded with embedded coding. Then we follow a recursive procedure to group all the dictionary symbols into *super-classes*. We put all super-class leaders into the direct dictionary (bitmaps coded without reference to any other dictionary symbol) and other dictionary symbols into the refinement dictionary (bitmaps coded with reference to another dictionary symbol) [1].

To design the tree-based (TREE) dictionary [7] we first compute the matching graphs between all extracted symbols. From these graphs we construct minimum spanning trees (MSTs) using Kruskal's algorithm [9]. We put all root nodes into the direct dictionary,all intermediate nodes into the refinement dictionary, and all leaf nodes into embedded coding. We can almost arbitrarily change the TREE dictionary's size to obtain the best compression. We do this by increasing the number of leaf nodes by relocating certain nodes' children to their new parents.

In this paper we consider a new dictionary design which combines the CLASS and TREE design ideas. We call it the modified-class (MC) design. First we group all symbols into classes and choose the representatives as in the CLASS design. Then we construct MSTs for all the representatives. This improves over the CLASS design because MSTs give better reference relationships among dictionary symbols than those given by super-classes [7]. The MC design is also computationally less complex than the TREE design. Our experiments on twelve test images show that, for lossless compression, the MC design is basically the same as the CLASS design while slightly worse than the TREE design. However, in the TREE design the encoder has to exhaustively search for the optimal dictionary size to achieve the best compression. For lossy compression, the MC design achieves the best compression.

Dynamic dictionary update: The fonts and sizes of text characters in one input page are usually very similar. Therefore, in page striping, we dynamically update the current dictionary from previous ones.

Updating a SE dictionary is straightforward. For each new symbol in the current stripe, the encoder matches it with all dictionary symbols from previous stripes and with all previous symbols in the current stripe. The encoder then points it to its closest match and adds it to the dictionary. After the current stripe is processed, the encoder examines the new dictionary and excludes all singletons from it. Those dictionary entries from previous stripes, if not used by any symbol in the current stripe, are expunged. This way new symbols useful for the current stripe get included in the new dictionary, and old symbols that are obsolete are removed.



(a) classic Kruskal's (b) modified Kruskal's

Fig. 1. Modified Kruskal's algorithm.

For the MC design, the first step is to form classes and choose representatives. This step is carried out on all new symbols and all existing dictionary symbols. This way if a pre-existing dictionary symbol is selected as the class representative, we do not need to encode its bitmap again. At the next step we use the modified Kruskal's algorithm to construct MSTs. In Fig. 1 we illustrate the classic and the modified Kruskal's algorithms. We show previous dictionary symbols in gray and newly added ones in black. The problem with the classic algorithm is that the MST in Fig. 1(a) includes the four gray nodes; but in fact the reference relationship among pre-existing symbols is meaningless because the decoder has those bitmaps already. We modify it as in Fig. 1(b). We first assume each pair of existing symbols has zero mismatch and is therefore connected by the dash-dotted gray edges. We then go on and apply the classic Kruskal's algorithm. With the modified algorithm each resulting MST is guaranteed to have at most one gray node representing a previous dictionary symbol; some MSTs may have no gray nodes if they contain symbols only from the current stripe. If a gray node exists in an MST,we use it as the root; otherwise we decide the root as in [7].

3. EXPERIMENTAL RESULTS

3.1. Fast and memory-efficient encoder

In this section we show the savings in encoding time and memory usage when page striping is applied. Our test im-

ages are from two sources: two 200-dpi CCITT standard images (f01 and f04); and ten 300-dpi images (IG0H, J00O, N03F, N03H, N03M, N046, N04D, N04H, N057 and S012) selected from University of Washington Document Image Database I [10]. We compare coding results using the proposed updating procedures for SE and MC dictionaries. Our experiments are done on a Pentium Pro 200MHz, running Red Hat Linux 6.0, with 64MB physical memory. Execution time (in sec) is measured with Unix "time" and peak memory usage (in % total system memory) with Unix "top". Our code was not optimized for speed or memory efficiency. Fig. 2 plots coded file size, encoding time, and peak memory usage as functions of the number of stripes into which a page is split. The results shown are for image f04; very similar results are obtained for all other test images. As the number of stripes increases, the **compressed file size** goes up at a slow and relatively constant pace. The slope of each line segment in Fig. 2 (a) represents coding loss per stripe added. The mean value of all the slopes stands for the speed at which the coding loss is being incurred; the variance stands for how steadily the loss is being incurred. For all test images, the mean value of the slopes is 1.6% for the MC dictionary and 1.3% for the SE dictionary; hence the coding loss per stripe is small. The SE dictionary has very low variances (on the order of 10^{-2} or 10^{-3} percent) for all twelve images. The MC dictionary shows the same results on all images but one. For **encoding time**,o n average 29% of encoding time is saved when we treat the page as two stripes instead of one; and returns diminish with more stripes. For **peak memory usage**, the average saving with 2 stripes is 40% compared to no striping. After 6 stripes, the curves flatten out because each stripe becomes small enough that the memory used to buffer it no longer dominates the total memory usage. In Fig. 2, we see that the "MC dict+3 stripes" scheme (33543 bytes) achieves the same compression as the "SE dict+1 stripe" scheme (33545 bytes) while the former scheme encodes 53% faster and uses 54% less memory. In general, MC dictionaries with $n+1$ or $n+2$ stripes achieve approximately the same compression as SE dictionaries with n stripes.

3.2. Multi-page document compression

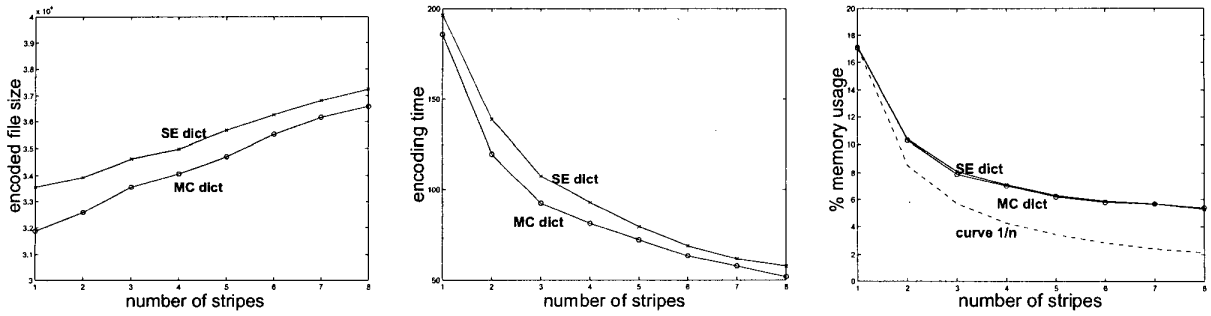
Multi-page document images are a set of images scanned from the same source, preferably from consecutive pages. The issues of compressing multi-page document images are addressed in [11]. In this section we try the proposed dictionary updating strategy on multi-page document compression. We use three test sets. Two are from University of Washington Document Image Database I, one of 4 pages (N04H, N04I, N04L and N04M) and the other of 5 pages (N01F, N01G, N01H, N01I and N01J). They are from the same source, but not from consecutive pages. The third set is an 11-page document we scanned in from [12], at 300 dpi. In Table 1 we show compression results for all three sets using three coding strategies combined with SE or MC dictionaries. Strategy A encodes the images as an uncorrelated set of single-page documents. Strategy B uses the dictionary designed from the first page to encode all the pages. Strategy C uses the proposed dictionary updating techniques from page to page, taking in new symbols in the current page and discarding useless symbols from previous pages. For both dictionaries, we see the biggest improvement from strategy C over strategy A is obtained with our 11-page set, amounting to 8% for the MC dictionary and 10% for the SE dictionary, respectively. This is because this set is scanned in under the same conditions and from consecutive pages, therefore the page correlation is the strongest. For the other two test sets, the scanning conditions are unknown. Another interesting phenomenon is, when using the SE dictionary, strategy B also achieves 4-5% of improvement over strategy A. In [6] we showed that SE dictionaries are usually twice as big as CLASS dictionaries. While this is a disadvantage when coding a single-page document (because index coding is too costly), it is advantageous to use a bigger static dictionary throughout all the pages (because all the symbols from later pages have a broader range of choices). On the contrary,we hardly see any improvement from strategy B over strategy A using MC dictionaries because they are too small and too specific for the first page. Note that the complexity of option SE+B is very low because the dictionary is designed only once. In Figure 3 we show the dictionary size growth from page to page. For our 11-page test set, from the fourth page on, the dictionary no longer grows, meaning that the encoder has gathered most useful bitmap information contained in this document set. The other two test sets do not contain enough pages to show this trend.

Table 1. Coding multi-page documents using the 3 strategies combined with the 2 dictionaries. Strategy A rows are compressed file size in bytes; the other rows are % improvement over strategy A.

	UW 4-pg	UW 5-pg	Our 11-pg
SE+A	126056	72237	222430
SE+B	5.3%	3.7%	5.4%
SE+C	3.9%	4.2%	10.3%
MC+A	121842	70420	214506
MC+B	1.8%	0.3%	0.4%
MC+C	2.4%	3.2%	8.2%

4. CONCLUSION

In this paper we first introduced a new dictionary design technique called the modified-class design. Then we pro-



(a) File size (in bytes) (b) Coding time (in sec) (c) Peak % system memory used

Fig. 2. File size, encoding time and peak memory usage curves: image f04.

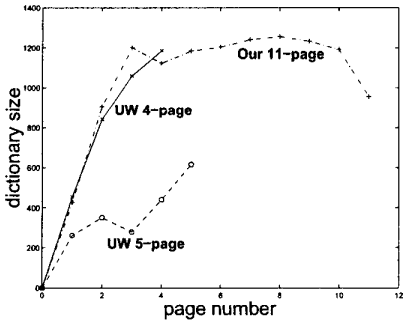


Fig. 3. Dictionary size growth from page to page.

posed separate dictionary updating procedures for the singleton exclusion and the modified-class dictionary. Using such procedures we developped a fast and memory efficient JBIG2 encoding strategy which splits up the input image into horizontal stripes and encodes one stripe at a time. Experiments show that, for both dictionaries, by using two stripes, the encoder can encode 30% faster and use 40% less memory while suffering a small 1.5% penalty in coding efficiency. Further savings in time and memory usage can be obtained when more stripes are used. We also applied the same updating procedures directly to multi-page document compression. Compared to coding the images as single-page documents, this can improve compression by up to 8-10%.

5. REFERENCES

[1] ISO/IEC JTC1/SC29/WG1 N1359. *JBIG2 Final Committee Draft*, July 1999.

[2] P. Howard, F. Kossentini, B. Martins, S. Forchhammer, W.Ruck lidge, 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] D. Tompkins and F. Kossentini. A Fast Segmentation Algorithm for Bi-level Image Compression Using JBIG2. *Proc. 1999 IEEE Intl. Conf. on ImagePr ocessing (ICIP)*, Kobe, Japan, October 1999.

[4] R.N. Ascher and G. Nagy. Means for Achieving a High Degree of Compaction on Scan-digitized Printed Text. *IEEE Trans. on Computers*,V ol. 23, pages 1174–1179, Nov.19 74.

[5] 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.

[6] Y. Ye, D. Schilling, P. Cosman, and H. H. Koh. Symbol dictionary design for the JBIG2 standard. *Proc. 2000 IEEE Data Compression Conf. (DCC)*, pages 33–42, Snowbird, Utah, March 2000.

[7] Y.Y e and P. Cosman. JBIG2 symbol dictionary design based on minimum spanning trees. *Proc. of the First Intl. Conf. on Image and Graphics (ICIG)*, pages 54–57, Tianjin, China, Aug. 2000.

[8] ISO/IEC JTC1/SC29/WG1 N339. *Xerox Proposal for JBIG2 Coding*,Jun e 1996.

[9] R. Gould. Graph Theory. Chap. 3, pages 68-72, The Benjamin/Cummings Publishing Co., Inc., 1988.

[10] 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.

[11] S. Inglis. Lossless document image compression. Ph.D. dissertation. Chap. 7. Univ.of Waikato, New Zealand, 1999.

[12] R. Habib. The early T.S. Eliot and western philosophy. Chap. 1, pages 1-11, Cambridge University Press, 1999.