A Fast Lossless Compression Scheme for Digital Map Images Using Color Separation

Saif Zahir and Arber Borici
Graphics and Image Processing Lab, Computer Science
University of N. British Columbia, BC, Canada
Email: zahirs@unbc.ca and borici@unbc.ca

bstract
In this paper, we present a fast lossless compression scheme for digital map images in the raster image format. This work contains two main contributions. The first is centered around the creation of a code book that is based on symbol-entropy. The second contribution is the introduction of a new row-column reduction algorithm. Our scheme proceeds as follows: we determine the number of different colors in a given map image. For each color, we create a separate bi-level data layer, one for the color and the second is for the background. Then, we compress each bi-level layer individually using the proposed method, which is based on symbol-entropy in conjunction with our row-column reduction coding algorithm. Our experimental results show that our lossless compression scheme achieved on average a compression equal to 0.035 bits per pixel which is better than most reported results in the literature or comparable to some. Moreover, our scheme is simple and fast.

Keywords – Lossless image compression, entropy, symbol entropy, map images, map image compression, GIS map compression.

I. Introduction

Digital map images may be stored in vector formats, such as Scalable Vector Graphics (SVG), or as raster maps, i.e. as rectangular grids of pixels. The latter is a preferable format because of the simplicity of directly accessing the stored map [1]. However, storing maps as raster images comes with high storage and transmission costs. For instance, a digital raster map of the province of British Columbia - Canada with a scale factor of 1: 20000 occupies nearly 50 GB [2]. Therefore, the reduction in the amount of raster map images is a crucial step, for both memory and transmission requirements to facilitate map distribution over the web. In addition, to transmit and visualize maps, they need to be in raster format [3].

Lossless graphics compression methods are generally classified into two categories: (i) methods that are applied directly on the image, such as the graphics interchange format (GIF) and the portable network graphics (PNG); (ii) and methods applied on every layer extracted (or separated) from the image, such as TIFF-G4 and JBIG. In this research, we focus on the second category.

Previous work in literature amounts to several lossless compression methods for map images based on layer separation. The standard JBIG2, which is specifically designed to compress bi-level data, employs context-based modeling along with arithmetic coding to compress binary layers. In [4], a lossless compression technique based on semantic binary layers is proposed. Each binary layer is compressed using context-based statistical modeling and arithmetic coding, which is slightly different from the standard JBIG2. In [5], a method which utilizes interlayer correlation between color separated layers is proposed. Context-based modeling and arithmetic coding are used to compress each layer. An extension of this method applied on layers separated into bit planes is given in [6]. Moreover, other lossless compression standards, such as GIF, PNG, or JPEG-LS are applied directly on the color image.

In this paper, we propose a new hybrid compression scheme for raster map images containing a predefined number of discrete colors. Our method works as follows. First, we extract as many layers from the image as the number of unique colors in the image. Each layer represents bi-level data, where the color is denoted with bit “1” (interchangeably referred to as “black”) and the background is denoted with bit “0” (referred to as “white”). Second, we convert each bi-level layer into 0-1 matrices. We then partition each test matrix into 8×8 blocks. For each 8×8 block, we search a previously constructed codebook to check if the block is found; if yes, we replace the block data (the 64 bits) by its corresponding code in the codebook (the maximum of which is 17 bits). If the block is not found in the codebook, we employ our row-column reduction coding algorithm (RCRC), to compress the block. If the compressed data by RCRC is smaller than its original size (i.e., < 64 bits), we use the compressed data; otherwise, we use the block’s original bits.

The rest of the paper is organized as follows. In Section II, we explain how the codebook is constructed and describe the RCRC algorithm. We also present the decoding process in this section. In Section III, we briefly describe the test data sample and we present our experimental results. Conclusions are incorporated in Section IV.

II. The Proposed Method

2.1 Block Entropy and Codebook Generation Process

The concept of entropy, borrowed from thermodynamics and introduced in Information Theory by Shannon, is a quantitative measure of the uncertainty contained in a particular message or system, in general [7]. The more random (or disordered) a system, the more information is contained in that system and the higher its entropy is. Thus, entropy provides a theoretical lower bound for coding and serves as a compression target. If the entropy of a particular message is H, then the highest compression ratio that can be achieved for that message is $(S-H)/S$, where S is the size (usually in bits) of the message. This implies that the smaller the entropy, the higher the compression ratio, and conversely.

Our compression method uses a fixed–to–variable codebook that is generated using Huffman tree codes. In order to construct an efficient and practical codebook, we performed a frequency analysis on a sample of more than 250,000 blocks of 8×8 bits. The data were obtained by partitioning 120 randomly chosen binary image samples. In our view, these bi-level samples strongly represent binary layers of map images. We proceeded as follows: (1) we clipped the image margins in such a way to include all content and background bits at a minimum. We did this preprocessing to minimize the impact of background margins redundancies on the code construction of the codebook (dictionary). That is, image margins would bias the result of 8×8 block that represents the image background, which happens to have the shortest Huffman code in the codebook; (2) we append the $r \times c$ test data matrix by either zeros or ones, depending on the background color, to become a $r^* \times c^*$ matrix such r^* and c^* are divisible by 8. For example, a 100×100 matrix will be modified to become a 104×104 matrix using the two steps above. This is achieved via appending 4 rows and 4 columns with bits that are equal to the bit value of the image background; (3) we partitioned the image sample matrices into 8×8 blocks which produced nearly 250,000 blocks. At this point, we calculated the frequency of occurrence of each block and identified all unique blocks in the sample; (4) finally, we built a codebook that contains high-frequency 8×8 bit blocks and their corresponding Huffman codes.

Due to the nature of binary images, where white (encoded with bit '0', by convention) or black (encoded with bit '1') dominate the background and other parts of the images, it is normal to expect all-zero (or all-one) 8×8 blocks to have the largest frequency (and, hence, the shortest Huffman code). We observed this phenomenon in our data sample, where the all-zero block (representing white background) has a Huffman code length equal to 1 bit, and the all-one block (representing black) has a Huffman code length of 2 bits.

In our sample, we have identified a total of 65,534 unique blocks. From this total, we have chosen 6,952 blocks that occurred more than once in order to construct our codebook, and we discard the other blocks, which appeared only once. We also noticed that the entropy of the entire system (sample set of the 65,534 blocks) is higher than the entropy of the codebook, which contains 6,952 blocks. This is an expected result because the system entropy decreases with a decrease in the possible outcomes of that system.

For a codebook that includes all 65,534 sampled blocks, the entropy would be equal to 7.61 bits per block, thus giving an 88.11% theoretical compression upper bound. For our dictionary with 6,952 blocks, the entropy value is found to be 4.08 bits per block, yielding a 93.62% compression upper bound. There are three reasons we limit our dictionary to 6,952 entries. First, the Huffman algorithm produces codes in a reasonable amount of time with a smaller number of initial codes. Second, the 6,952 blocks were observed to be the most frequently occurring blocks in our sample. Third, we get very good results.

To employ this codebook, we compare each 8×8 block of an input layer matrix to the 8×8 block entries of the codebook. If a match occurs, we assign the corresponding Huffman code to the block and continue until the blocks of the bi-level layer are all processed.

Based on the constructed codebook, the average Huffman code length is 4.094 bits per block, which is greater than the entropy value of 4.084 bits per block. The difference between the average Huffman code length and the entropy value, which in our case equals 0.01, is referred to as the redundancy value. In percentage, the redundancy is found to be 0.252% of the entropy. This means that, on average, we need 0.252% more bits than the minimum required in order to code the sequence of blocks in the codebook. To calculate the frequencies of all unique 8×8 blocks observed in our sample, it took the program almost 20 days to terminate on an Intel Dual Core machine (1.6 GHz per processor) with 2.49 GB of RAM.

2.3 The Row–Column Reduction Coding (RCRC)

The second component of our lossless compression scheme is our improved RCRC algorithm. This component is designed to deal with those blocks that are not found in the codebook. The RCRC proceeds as follows: For each 8×8 block, generate a row reference vector (RRV), a column reference vector (CRV), and a reduced block (RB). The RRV and CRV are constructed in a similar way. We construct the RRV as follows: (i) we create an empty 8×1 vector; (ii) we compare the first row of bits in the block with the second row. If they are identical (as a result of their XOR-ing), we place a '1' in the first row and a '0' in the second bit locations in the RRV; (iii) we compare the third row with the first row. If they are identical we assign '0' for the third row as well. We continue until we encounter a non-identical row or until we reach the last row of the block; (iv) if two consecutive rows are not identical, we place '1' for both rows in their corresponding locations in the RRV. At this point, the second row becomes the current row and we proceed as above; (v) remove all identical rows from the block to obtain a row-reduced block.

We follow the same procedure for the columns of the row-reduced block. In this case, we generate the 1×8 column reference vector (CRV) whose zero-valued entries correspond to columns that have been removed from the block. The result of this procedure will be a row-column-reduced block or simply the reduced block, (RB). In short, the output of the RCRC algorithm will be the RRV, CRV, and RB.

To explain this process, consider Figure 1 below, the row reduction operation is applied on a selected block. The row reference vector (RRV) is shown on the left of the block. In this case, the first row is identical to the second row, which is removed from the block. Therefore, a value of '1' is placed in the first location of the RRV (for the first row), and a value of '0' is placed in RRV second location for the second (eliminated) row. Finally, a value of '1' is placed for the third row in the third RRV location, and a value of '0' is placed for the corresponding RRV locations of rows 4 to 8, which are eliminated since they are identical to row 3.

The column-reduction operation is applied on the row-reduced block, as depicted in Figure 2. The column-reference vector (CRV) is shown on top of the block. In this case, the first column is identical to and eliminates columns 2 to 6. Also, column 7 eliminates column 8. This results in the reduced block, RB, shown on the right of the column-reduced block. For this example, the output of the RCRC is a concatenated string composed of the RRV (the first group of 8 bits), CRV (the second group of 8 bits), and the RB (the last 4 bits) displayed as a vector: 1010000010000010**1011**, for a total of 20 bits. The compression ratio achieved for this block is (64 – 20)/64 = 68.75%.

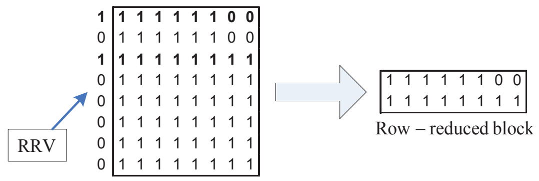

Figure 1. The row-reduction operation applied on a block

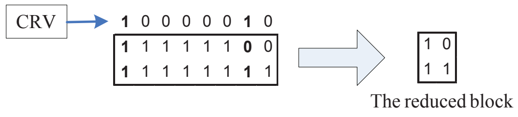

Figure 2. The column-reduction operation applied on the row-reduced block in Figure 1

To further clarify the decoding process, we consider the row-column reduced block of the preceding example. The number of ones in the row-reference and column-reference vectors shows the number of rows and columns of the reduced block respectively. The output 1010000010000010**1011** contains two ones in the first group of 8 bits (the RRV), and two ones in the second group of 8 bits (the CRV). This means that there are 2 rows and 2 columns in the reduced block. That is, the first two bits of the reduced block i.e., '10' of the underlined bits, represent the first reduced row, and the second two bits, '11' of the underlined portion, represent the second reduced row. Then, given the ones and zeros in the reference vectors, we construct the rows and columns of the original block. Figure 3 shows the column reconstruction based on the column-reference vector.

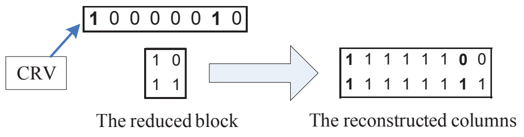

Figure 3. Column reconstruction based on the column-reference vector (CRV)

In Figure 3, the CRV tells the decoder that columns 2 to 6 are exact copies of column 1, and column 8 is an exact copy of column 7. The block on the right depicts this operation. Figure 4 shows the row reconstruction process to obtain the original block.

If the output of the row–column reduction routine has a size larger than 64 bits, we keep the original block. Such cases represent less than 5% of the total block count in the test samples. Moreover, the minimum number of bits that can be achieved by RCRC is 17 bits, for a maximum compression ratio of 73.44%, and occurs in cases where all 7 rows and 7 columns of the block are eliminated which leads to 1 bit reduced block.

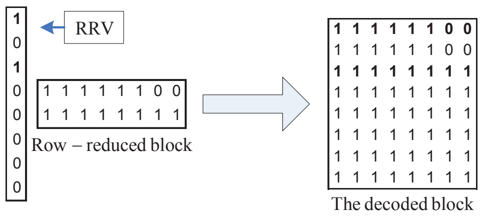

Figure 4. Row reconstruction based on the row-reference vector (RRV)

The schematic diagram shown in Figure 5 illustrates the operation of the proposed compression scheme. Each map image is appended in both dimensions to become divisible by 8. Then, layers are extracted through color separation yielding a set of bi-level matrices. We partition each layer into 8×8 blocks. Each 8×8 block of the original data is searched in the codebook. If it is found, the corresponding Huffman code is selected and added to the compressed data stream. If it is not found, the row–column reduction coding attempts to compress the block. If the output of RCRC is smaller than 64 bits, we add the reduced block to the compressed data stream. Otherwise, we keep the original block.

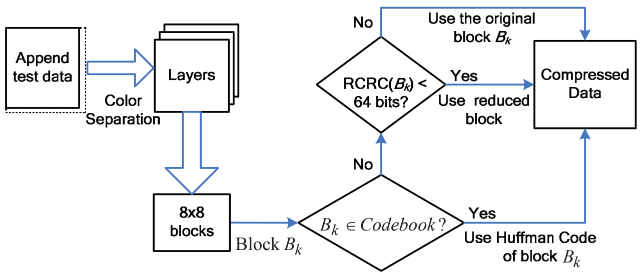

Figure 5. Diagram of the proposed compression technique

2.4 The Encoding/Decoding Process

The encoding process is simple and straightforward. In order to distinguish blocks compressed by the codebook, or blocks compressed by RCRC, or uncompressed blocks, we consider the following cases.

Case 1: If the block is found in the codebook, then use the corresponding Huffman code. We consider two sub-cases.

- *Case 1a*: If the corresponding Huffman code is the shortest in the codebook (i.e. 1 bit in the case of our codebook) then assign '11' to encode that block.

- *Case 1b*: If the corresponding Huffman code has a length > 1 bit, assign '00' to encode the block. As the length of the Huffman codes in our codebook varies from 1 bit to 17 bits, we use an additional 5 bits to encode the length of the Huffman code immediately after '00' bits. For instance, if a block that is found in the codebook has a Huffman code of length 7, then the block will be encoded as 00 + 00111 + *The Huffman Code*. In this case, the 5 bits (00111) tell the decoder that the Huffman code that follows immediately has a length equal to 7 bits and, thus, the decoder will read the next 7 bits.

The reason we use two overhead bits for the block that has the shortest Huffman code length in the codebook is based on the empirical result that this entry comprises 73.74% of the total blocks found in the codebook.

Case 2: If the block is compressed by RCRC, we use ‘01’ followed by the output bits of RCRC. The decoding process for RCRC is explained in Section 2.3.

Case 3: If the block is neither found in the codebook, nor compressed by RCRC, we use the two bits ‘10’, after which the 64 bits of the original block data are appended. The decoding process is straightforward and follows the same encoding process explained above.

III. Results

To examine the effectiveness and performance of the proposed scheme, we tested it on a sample of topographic map images of four semantic layers each obtained from the GIS lab at the University of Northern British Columbia [2]. Semantic layers include contour lines, lakes, rivers, roads, etc. Table 1 provides a summary description of three map images used in this process. Table 2 shows the compression results in bits per pixel (bpp) of the proposed method for the three map images shown in Table 1. It is clear that the proposed method achieves very high compression on the selected set of map images. We note that our results are higher than JBIG, which has been reported to compress at a rate of 0.22 to 0.18 bits per pixel [1]. On average, our method achieves a compression of 0.035 bpp on the selected data sample. This result is higher than or comparable to those results reported in [4].

Table 1. Description of selected topographic map images, scale 1:20000 [2]

Map	Dimensions	Size (KB)
Map 1	2200 x 1700	10,960
Map 2	5776 x 13056	220,979
Map 3	5112 x 11600	173,769
	Total Size	406,708

Table 2. Compression results using our scheme

Map Image	Original Size (KB)	Compressed Size (KB)	Compression Ratio (bpp)
1	10960	210.77	0.019
2	220979	7489.27	0.034
3	173769	6626. 27	0.038
Total	**406,708**	**14326.42**	**0.035**

IV. Conclusions

In this paper, we introduced a fast and highly efficient hybrid compression scheme for map images with predefined number of discrete colors. The proposed scheme contains three steps. The first step is to divide the map image into as many bi-level layers as the number of colors. The second step is to implement the proposed scheme on the 8×8 bit blocks of the map bi-level layers. The proposed scheme scored a high compression of 0.035 bits per pixel which is higher than JBIG as well as many other compression algorithms. In fewer cases, the scheme obtained comparable results to those reported in [4]. This scheme is simple, fast and can be used for real time applications.

Acknowledgement

The Authors would like to thank the GIS lab instructors at UNBC (Roger Wheate and Scott Emmons) for their valuable support to this research.

References

[1] Franti, P., Kopylov, P., and Ageenko, E., “Evaluation of Compression Methods for Digital Map Images”, *Int. Conf. on Automation, Control, and Information Technology*, Novosibirsk, Russia, June 2002.

[2] University of Northern British Columbia GIS Lab Database, “Topographic Map of British Columbia,” 2009.

[3] Akimov, A., Kolesnikov, A., and Franti., P., “Lossless Compression of Map Contours by Context Tree Modeling of Chain Codes”, *Pattern Recognition*, 40, 2007, pp. 944-952.

[4] Franti, P., Ageenko, E., Kopylov, P., Grohn, S., and Berger, F. “Map Image Compression for Real-Time Applications”, *Joint Int. Symposium on Geospatial Theory, Processing and Applications, Ottawa, Canada,* July 2002.

[5] Kopylov, P., and Franti, P. “Compression of map images by multilayer context tree modeling”, IEEE Trans. On Image Processing, 14 (1), 2005, pp. 1-11.

[6] Podlasov, A., and Franti, P., “Lossless Image Compression via Bit-Plane Separation and Multilayer Context Tree Modeling”, Journal of Electronic Imaging, 15 (4), 2006.

[7] Sayood, K, “Introduction to Data Compression,” 3rd Ed. *Morgan Kaufmann*, 2005.

[8] Turner, M., and Wiseman, N., “Efficient Lossless Image Contour Coding”, *Comp. Graphics Forum*, 15 (2), 1996, pp. 107-118.

[9] Howard, P.G., Kossentini, F., Martins, B., Forchammer, S., Rucklidge, W.J., “*The Emerging JBIG2 Standard*”, *IEEE Trans. Circuits and Systems for Video Technology*, 8, 1998, pp. 838-848.

[10] Franti, P., Ageenko, E., Kopylov, P., Grohn, S., “Compressing Multi-component Digital Maps Using JBIG2”, *Proc. IEEE Int. Conf. on Acoustics, Speech, and Signal Processing*, Orlando, Florida, May 2002, pp. 2677-2680.

[11] Akimov, A., and Franti, P., “Symbol Representation in Map Image Compression”, *19th Annual Symposium on ACM Computing*, Nicosia, Cyprus, March 2004.

[12] Zahir, S., and Dhou, K., “A New Chain Coding Based Method for Binary Image Compression and Reconstruction”, *PCS*, Lisbon, Portugal, November 2007.