Sādhanā, Vol. 21, Part 1, February 1996, pp. 55–64. © Printed in India.

Lossless and lossy image compression using Boolean function minimization

R P DAMODARE, J AUGUSTINE and J JACOB

Department of Electrical Communication Engineering,
Indian Institute of Science, Bangalore 560 012, India

Abstract. A novel approach for lossless as well as lossy compression of monochrome images using Boolean minimization is proposed. The image is split into bit planes. Each bit plane is divided into windows or blocks of variable size. Each block is transformed into a Boolean switching function in cubical form, treating the pixel values as output of the function. Compression is performed by minimizing these switching functions using ESPRESSO, a cube based two level function minimizer. The minimized cubes are encoded using a code set which satisfies the prefix property. Our technique of lossless compression involves linear prediction as a preprocessing step and has compression ratio comparable to that of JPEG lossless compression technique. Our lossy compression technique involves reducing the number of bit planes as a preprocessing step which incurs minimal loss in the information of the image. The bit planes that remain after preprocessing are compressed using our lossless compression technique based on Boolean minimization. Qualitatively one cannot visually distinguish between the original image and the lossy image and the value of mean square error is kept low. For mean square error value close to that of JPEG lossy compression technique, our method gives better compression ratio. The compression scheme is relatively slower while the decompression time is comparable to that of JPEG.

Keywords. Switching theory; Boolean minimization; image compression.

1. Introduction

Conventional lossless image compression techniques employ a decorrelation technique such as DPCM or Hierarchical Interpolation followed by a coding scheme such as Huffman or arithmetic coding (Roos *et al* 1988). In this paper we propose a radically different approach to coding based on finding the minimal Boolean function representation for sub-blocks of an image bit plane. The emphasis in this paper is on the coding scheme and further investigation is needed to determine the decorrelation method that best suits our coding scheme.

In lossless as well as lossy image compression, the state of the art is the JPEG standard (Pennebaker & Mitchell 1993). For lossless compression, JPEG employs a decorrelation

scheme based on DPCM (several predictors are provided to the user) followed by adaptive Huffman or arithmetic coding. The lossy JPEG processes are based on the Discrete Cosine Transform (DCT) and entropy coding of the quantized DCT coefficients based on adaptive Huffman or arithmetic coding. Our techniques for lossless and lossy compression are based on the minimization of Boolean switching functions.

In lossless compression an error file is obtained by applying a simple linear prediction on the original image file. The error file is now split into bit planes. Each bit plane is a binary image which is divided into variable sized blocks using the quad tree approach and each block is converted into a Boolean switching function in cubical form. The functions are then minimized using the well known cube-based two-level logic minimizer ESPRESSO (Brayton *et al* 1984). The minimized cubes which represent the implicants (product terms) of the function are then coded with a code set which satisfies the prefix property, to obtain the compressed data. The results are compared with JPEG lossless mode of compression.

In lossy compression we reduce the number of gray levels in the original image to less than or equal to 32 by applying the *Centre of Mass technique* . The image containing at most 32 gray levels will be recoded by mapping the gray values to a 5 bit gray code, thus reducing the number of bit planes from 8 to 5. The mapping function leads to a fixed overhead of only 96 bits. These 5 bit planes will be processed in the same manner as we do for the lossless compression scheme. The results are compared with JPEG lossy compression technique.

2. Background

The basic idea of employing Boolean minimization for image compression has been reported in an earlier work (Augustine *et al* 1995). However, in that paper, only lossless compression of bi-level (black and white) images was considered. In the present work, we have extended the approach to both lossless as well as lossy compression of gray level images and our technique can be easily applied to full colour images as well. We also consider the division of each bit plane into variable sized windows or blocks using a quad tree structure, leading to higher compression efficiency, as opposed to uniform division into fixed sized blocks (Augustine *et al* 1995). For the sake of clarity of presentation, some basic definitions are reproduced below from Augustine *et al* (1995).

DEFINITION 1
A Boolean *switching function* $\mathbf{F}$ is a mapping $\mathbf{F} : \mathbf{B}^N \rightarrow \mathbf{B}$, where $\mathbf{B} = \{0, 1\}$.

DEFINITION 2
In the truth table of a switching function of N variables, there are 2^N rows. Each of these rows which represents an input state vector is called a *minterm.*

DEFINITION 3
In a switching function, the *ON-set* is the set of minterms whose outputs are mapped to 1 and the *OFF-set* is the set of minterms whose outputs are mapped to 0.

DEFINITION 4
A *cube* is an N-tuple $\mathbf{A} = \{a_1, a_2, \ldots, a_N\}$, where $a_i \in \{0, 1, X\}$. The dimension of the cube is the number of Xs in it. An α cube has 2^α minterms (zero cubes) within it (Biswas 1993; Breuer 1972).

DEFINITION 5
The performance of a lossless data compression algorithm is assessed by the parameters, *compression ratio* and *run time*. We define compression ratio as,

$$\frac{(\text{no. of input bytes} - \text{no. of output bytes})}{\text{no. of input bytes}} \times 100\%.$$

3. The Lossless compression scheme

Block diagram of our lossless data compression scheme is given in figure 1. The compression scheme consists of the steps of linear prediction, function generation, function minimization and cube encoding.

3.1 *Linear prediction*

First we apply linear prediction on the original image file. A simple predictor of averaging two adjacent pixel values A and B as shown in figure 2 is used. The error values obtained are recoded by adding the magnitude of the maximum negative error value to each error to obtain only positive values in the error file.

3.2 *Function generation*

The values in the error file are replaced by their equivalent gray code, so that adjacent error values are advantageously mapped onto logically adjacent codes leading to fewer transitions between 0s and 1s on the bit planes. The error file is now split into bit planes. The number of bit planes can be 8 or 9 depending on the number of error values obtained. If the number of error values are more than 256, then there will be 9 bit planes. The bit planes will be divided into variable window sizes using the Quad tree approach. A switching function in cubical form for each window is generated by assigning the pixels to minterms according to Gray code. Gray code is chosen because of its unit distance property, to capture the correlation likely to be present among adjacent pixels, by assigning geometrically adjacent pixels of a window to logically adjacent minterms. This assignment helps in minimization since any 2^{α} logically adjacent minterms combine to form a single α-cube. Pixels are scanned row wise with a reversal of the direction for adjacent rows, to

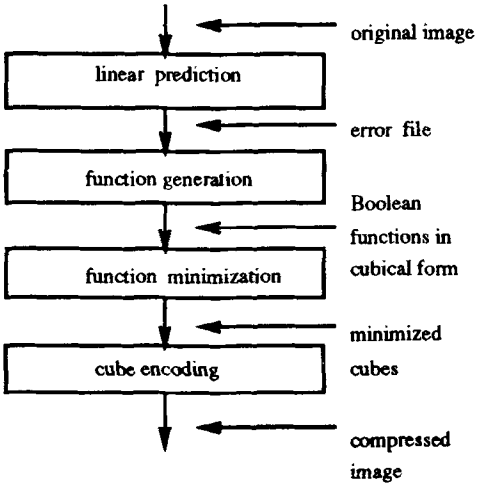

Figure 1. Lossless image compression scheme.

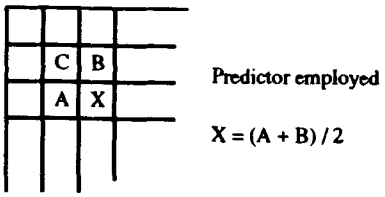

Figure 2. Sample predictor pixels.

ensure that pixels at the ends of consecutive lines are mapped to logically adjacent codes. A more detailed explanation on function generation can be found in Augustine *et al* (1995).

3.3 *Quad tree approach to bit plane segmentation*

Each bit plane is divided into sub-planes (windows or blocks) of variable sizes using the quad tree approach. This is done because of the fact that the region of correlation on the bit planes can be large or small. Capturing such variable size regions of high correlation on the bit planes results in better compression.

In the quad tree approach of dividing the bit plane into variable sized windows, a bit plane is treated as a collection of leaf nodes. Given a $2^n \times 2^n$ array of pixels, a quad tree is constructed by repeatedly subdividing the array into quadrants, subquadrants etc., until we reach the smallest window size that we have decided on. We have employed a maximum window size of 32×32 pixels and a minimum of 4×4. To decide whether a given block has to be further subdivided or not, we have used a simple heuristic criterion. If a given block consists of 98% or more black or white pixels, we retain this block without further division; otherwise it is divided into four equal sub-blocks and the same criterion is applied to each sub-block to decide whether to divide them further. Once the sub-block size reaches 4×4 pixels, we retain it without further division, even when our criterion is not satisfied.

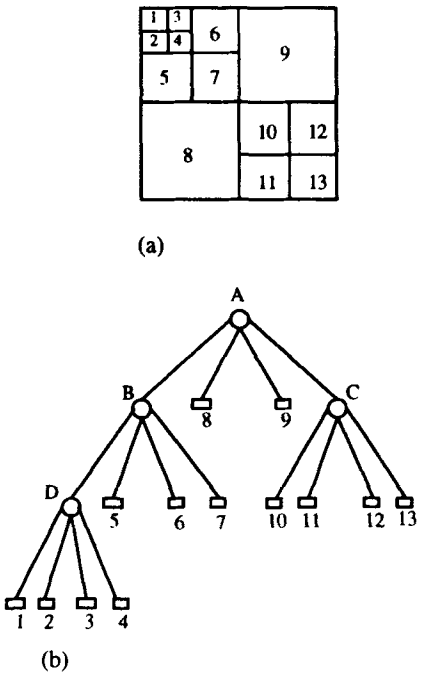

Figure 3. A bit plane and its Quad tree representation. **(a)** A bit plane of size 32×32, **(b)** Quad tree representation of the above bit map.

The process of subdividing the bit plane into quadrants, subquadrants etc., can be represented by a tree of out degree 4 in which the root node corresponds to the entire bit plane, the four sons of the root node correspond to the quadrants, and the terminal or leaf nodes correspond to those windows of the bit plane for which no further subdivision is necessary (Samet 1985). The nodes at level k (if any) represent blocks of size $2^k \times 2^k$ and are referred to as nodes of size 2^k. The tree is created in a depth first manner and hence is coded in preorder form in the compressed file.

An example of a bit plane being divided using quad tree approach is shown in figure 3. The bit plane is of size 32 × 32. The quad tree representation of the bit plane has been shown in figure 3b, where a small box indicates a terminal or a leaf node. The preorder listing of the quad tree is

A B D 1 2 3 4 5 6 7 8 9 *C* 10 11 12 13.

The preorder listing of the quad tree indicates the manner in which the bit plane was divided. The alphabets A, B, C etc., indicate the nodes which are not terminal nodes. The terminal nodes 8 and 9 are of size 16 × 16, nodes 5, 6, 7, 10, 11, 12 and 13 are of size 8 × 8 and the nodes 1, 2, 3 and 4 are of size 4 × 4. The quad tree structure can be effectively represented using only 1 bit for each node (Samet 1985).

3.4 *Function minimization*

Boolean function minimization is performed on the function generated for each block using the two-level cube-based logic minimizer ESPRESSO (Brayton *et al* 1984) to find the equivalent minimized cubical representation. For a particular function, in general, the number of cubes in its minimized ON-set and OFF-set are different. Better compression can be achieved by choosing the set with lesser number of cubes, since both represent the same function. The information regarding the choice of the ON/OFF set is passed on to the decoder via the header for each block, and is discussed in § 4.

3.5 *Cube encoding*

In cube encoding the set of minimized cubes of the function corresponding to each window is coded separately. A code set 0, 10, 11 which satisfies the prefix property is used for the cube symbols 0, 1 and X by allotting the one bit code to the symbol with maximum frequency of occurrence. One or two bits are needed to encode the information about the allotment of prefix code to the cube symbols. If minimization fails to achieve compression for any window, we choose to represent the original window as such in the compressed image and this is indicated in the window header.

4. Format of the compressed image

The compressed file format has a global header containing the information of the size of the image and the information required for reverse prediction. For each bit plane the windows or blocks will be encoded in a specific format given in figure 4. Since we have considered only window sizes of 32 × 32, 16 × 16, 8 × 8 and 4 × 4, the *quad tree bits* associated with any terminal node will have 1, 2 or 3 bits depending on the level of the node in the quad tree structure. The one bit *status code* for the window is interpreted as below.

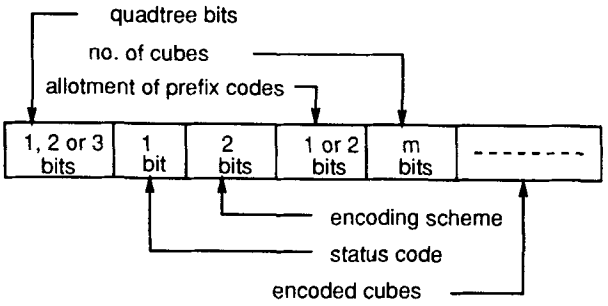

Figure 4. Window format.

- 0 : Window without minimization. Minimization algorithm has failed to produce compression for this window, and bits are stored as in the original window.
- 1 : Window is minimized and encoded as cubes.

In the latter case the next two bits are used to indicate further characteristics of the windows. These two bits denoting the *encoding scheme* have the following interpretation.

- 00 : ON set is minimized and stored as cubes.
- 01 : OFF set is minimized and stored as cubes.
- 10 : window consists of all black pixels.
- 11 : window consists of all white pixels.

For the last two cases, we need no further bits to encode the window. However, in the first two cases the next one or two bits as shown below, are used to indicate the *allotment of prefix codes* for the cube alphabets.

- $0:0 \rightarrow 0,\ 10 \rightarrow 1,\ 11 \rightarrow X.$
- $10:0 \rightarrow 1,\ 10 \rightarrow X,\ 11 \rightarrow 0.$
- $11:0 \rightarrow X,\ 10 \rightarrow 0,\ 11 \rightarrow 1.$

After the prefix code allotment bits, the next m bits (m varies between 0 and 6 depending on the window size) are used to indicate the number of cubes. Encoded cubes are placed after this (figure 4).

It is possible that although many windows in an image bit plane can be compressed by logic minimization or by virtue of the fact that they are *all black* or *all white* windows, several other windows may not yield any compression, leading to a net expansion for an entire bit plane. This usually happens for two or three least significant bit planes of most images, which have more random characteristics compared to the higher bit planes. In such a case, the entire bit plane is stored as it is in the compressed file. A one bit global header is associated with each bit plane to indicate whether the entire bit plane is in the original or compressed form.

5. Decompression scheme

Decompression procedure consists of three steps, namely,

1. Window recovery
2. Bit plane recovery
3. Image recovery

Table 1. Results of the lossless compression experiment.

Test Image	Logic coding			JPEG Lossless mode		
	comp. time*	decomp. time*	compr. ratio %	comp. time*	decomp. time*	compr. ratio %
baboon	195	2.8	10.7	0.4	0.3	13.7
boats	132	2.6	31.5	0.4	0.3	32.9
girl	125	2.6	37.0	0.4	0.3	38.1
average	151	2.7	26.4	0.4	0.3	28.2

* CPU s on IBM RS-6000/580

Window recovery consists of locating a block of data in the compressed file corresponding to a bit plane window and reconstructing the original bits of this window. If the window has not been compressed, recovery is straight forward; else the two bits for encoding scheme will indicate whether the ON/OFF set has been minimized and stored as cubes or whether the window is all black/white. In case of minimized ON/OFF set representation, the cubes of the minimized function are recovered from the knowledge of the number of cubes and the allotment of prefix code to the cube alphabets. Once the minimized ON/OFF set of the function is obtained in the cubical form, the bits of the original window can easily be obtained by expanding the function to its truth table form.

Bit plane recovery consists of reconstructing an entire bit plane from the recovered windows of this bit plane, using information from the quad tree representation for the different windows.

Image recovery consists of combining the individual bit planes of the image. Since linear prediction was employed as a preprocessing step, image recovery first yields the error file from which the original gray values are obtained by reverse prediction.

6. Experimental results

The proposed compression and decompression schemes, tentatively called as Logic coding, have been implemented in **C** on an IBM RS-6000/580 workstation with UNIX operating system, and tested on gray level images of size 256×256 pixels (64k bytes). Comparison of the performance of our scheme with that of JPEG lossless mode is given for 3 standard gray scale images *baboon*, *boats* and *girl* in table 1. We have used the PVRG-JPEG (Portable Video Research Group) Codec 1.1 available through internet from havefun.stanford.edu, to carry out the experiments. Our lossless compression scheme is comparable to JPEG lossless mode in compression ratio. The predictor used by JPEG is the same as the one used in our scheme shown in figure 2. The CPU time for encoding for JPEG is better than that of our compression scheme, but the decompression time is comparable to that of JPEG.

7. Lossy compression

Our lossy compression scheme consists of the following four steps.

- Reducing the number of bit planes from eight to five
- Function generation

- Function minimization
- Cube encoding

7.1 *Reducing the number of bit planes*

To reduce the number of bit planes we first reduce the number of gray levels in an image. The number of gray levels are reduced to 32 or less. To achieve this reduction, we use the technique called *Centre Of Mass technique*. We use the fact that the human eye cannot detect very small changes in gray values and have substituted a single gray value for a group of adjacent gray level values, as explained below.

We first divide the range 0 to 255 of gray values into 32 intervals. Each interval will consist of 8 gray values. The intervals are 0 to 7, 8 to 15, 15 to 23, ..., 248 to 255. The frequency count of each gray value in the original image is computed. Then for each interval k, ranging from 0 to 31, we calculate a gray value gl_{new} (representing the centre of mass of that interval) using the formula

$$gl_{new} = \sum_{i=8k}^{8k+7} i.fr_i \Big/ \sum_{i=8k}^{8k+7} fr_i$$

where i corresponds to the gray level value (between 0 and 255) and fr_i is the frequency of occurrence of ith gray level in the image. Thus we get a gray value gl_{new} for each interval which is substituted for all the 8 gray levels falling within that interval. If the value obtained from the above calculation is a fraction, it will be rounded off to nearest integer value. Thus for each interval we find a gray value gl_{new} which is biased towards those gray levels whose frequency of occurrence is more and thus help to reduce the mean square error value. One major advantage is that the error is bounded and is not localized but is distributed throughout the image.

The recoded image will have at the most 32 gray values and each of these values denote the centre of mass of the interval in which the gray value falls. The five MSBs of each recoded gray value represent the interval while the three LSBs denote the offset from the start of the interval to its centre of mass. We can maintain these offsets in an array of 32×3 bits and recode the intervals by a 5 bit gray code. Thus the entire recoded image now consists of 5 bit planes and the 32×3 bit overhead for denoting the offsets is included in the global header.

This is the only step where we incur loss in the image. Qualitatively one cannot visually distinguish between the original image and the recoded image having only 32 or fewer gray levels.

The five bit planes obtained from the *Centre Of Mass* technique will now undergo steps of function generation, function minimization and cube encoding in the same manner as explained in the lossless compression scheme.

7.2 *Compressed file format*

The compressed image format is same as that of the lossless compression scheme with differences in the global header. There will be a header of 12 bytes containing the offset information with respect to the centre of mass of each interval. There will be only five encoded bit planes in the compressed file. The window format is same as shown in figure 4.

Table 2. Results of the lossy compression experiment.

Test	Logic coding			JPEG lossy mode			
Image	MSE	comp. time*	compr. ratio %	Q-factor	MSE	comp. time *	compr. ratio %
baboon	5.45	150	47.85	95	4.16	0.5	33.37
				94	5.68	0.5	38.07
boats	5.47	94	65.85	93	5.10	0.4	62.39
				92	6.12	0.4	64.98
girl	5.40	85	67.87	93	5.14	0.4	66.88
				92	5.94	0.4	69.60
average	5.44	110	60.52		5.36	0.43	55.88

* CPU s on IBM RS-6000/580

7.3 *Experimental results*

The results of our scheme and that of JPEG are given in table 2. We have kept mean square error (MSE) as a reference to compare the compression ratios. To get MSE of JPEG decompressed file close to that of ours we have tried different values for the quality factor (Q-factor) which is an option available in JPEG. As the MSE of JPEG decompressed image was not exactly identical to ours, we have given results for two Q-factors giving results as close to our MSE as possible. From the results it is clear that for an average MSE close to that of JPEG, our compression ratio is better by 4.64% on an average, but JPEG has a plus point in the compression time required. The decompressed images were visually indistinguishable from the originals both for our scheme as well as JPEG.

8. Conclusions

A novel approach for image data compression using the Boolean function minimization technique has been presented. The images after the preprocessing step are split into bit planes. Each bit plane is split into variable sized blocks and these blocks are converted into a set of Boolean switching functions and minimized using ESPRESSO to get the minimal representation in cubical form. A prefix code set is used for the encoding of cube alphabets to achieve maximum possible compression.

In lossless compression of images our experiments show that our technique gives compression ratio comparable to that of JPEG lossless mode. In lossy compression technique, for an average mean square error close to that of JPEG, we are better in compression ratio by 4.64%. JPEG is better in the execution time required as compared to our technique. As ESPRESSO is used as a stand alone package, considerable time is wasted in the communication and the file manipulation overheads. Time performance is expected to improve once ESPRESSO is integrated into the code.

Two level minimized representation of a function is not the most compact representation for all functions. There are many functions for which the minimum two level representation leads to an expansion, for example the parity function. For such functions, alternative representation such as Binary Decision Diagrams (BDDs) may be employed (Akers 1978). Further work is also required to study the performance of our lossy compression approach at higher MSE levels and compare it with JPEGs performance.

References

Akers S B 1978 Binary Decision Diagrams. *IEEE Trans. Comput.* C-27: 509–516

Augustine J, Feng W, Jacob J 1995 Logic Minimization Based Approach for Compressing Image Data. *Proc. of the* 8^{th} *Intl. Conf. VLSI Design*. 225–228 (New Delhi: IEEE Press)

Biswas N N 1993 *Logic design theory* (Prentice Hall Inc., NJ)

Brayton R K, Hachtel G D, McMullen C T, Sangiovanni-Vincentelli A L 1984 *Logic minimization algorithms for VLSI synthesis* (Boston: Kluwer)

Breuer M A 1972 *Design automation of digital systems Vol-1: Theory and techniques* (Englewood Cliffs, NJ: Prentice Hall)

Pennebaker W B, Mitchell J L 1993 *JPEG Still Image Data Compression Standard* (New York: Van Nostrand Reinhold)

Roos P, Viergever M A, Van Dijke M C A, Peters J H 1988 Reversible intra frame compression of medical images. *IEEE Trans. Medical Imaging* 7: 328–336

Samet H 1985 Data structures for quadtree approximation and compression. *Commun. ACM* 28: 972–993