International Conference on
Information, Communications and Signal Processing
ICICS '97
Singapore, 9-12 September 1997

Lossless Compression of Images Using Minterm Coding

Debashish Pramanik *, James Jacob and Jacob Augustine †

Department of Electrical Communication Engineering
Indian Institute of Science,
Bangalore 560012, India

Abstract

Lossless compression of images is important in the fields like medical imaging and remote sensing. There are only a small number of good algorithms known to date for lossless image compression. We present here a lossless image compression scheme which uses concepts of logic coding and auto-adaptive block coding to obtain a scheme which performs comparable to JPEG standard. The scheme reaches the best possible value of compression automatically, unlike the case of JPEG which gives a range of compression, based on available predictors.

1 Introduction

Lossless compression is mandatory in the case of medical and satellite images due to well known reasons. Kunt and Johnsen [5] have proposed a lossless compression technique called *Block Coding* for binary images and have subsequently extended their approach to gray-level images, by applying it on the bit planes. In *Block coding,* each bit plane is divided into smaller blocks of size $n \times m$, which are classified into three types namely, *all-black, all-white,* and *mixed,* which are coded using prefix codes '1', '10' and '11'. In the case of mixed type block, the nm pixels of the block are put after the prefix code. The 2^{nm} different possible bit patterns of a mixed block can be considered as source messages and coded using a variable length code (VLC) such as Huffman code. By making n and m as large as possible, better result can be obtained with Huffman coding. However, for large alphabet size the design and implementation of Huffman code is complicated as it requires the measurement of 2^{nm} probabilities and table look-up involving a dictionary of large size. In [5], if the size of the mixed type block is small enough, it is coded using VLC; otherwise the block is transmitted as it is. The mixed type blocks have also been arithmetic coded to increase the compression ratio in the case of binary images [4].

In an earlier work, Augustine *et al* [1] demonstrated the possibility of compressing mixed type blocks using *logic coding.* The compression obtained using the technique on gray-level images were comparable to that of the lossless mode of JPEG. The approach consists of the steps of recoding the pixels and Gray coding, bit plane segmentation and logic coding. Recoding consists of arranging the intensity values in an image contiguously and Gray coding substitutes the recoded intensity values by their equivalent Gray codes. The image is then decomposed into its individual bit planes and each bit plane is divided into blocks of size $n \times m$. Each mixed type block in a bit plane is converted to a switching function, treating the binary pixel values as the output of the function. Each function is then minimized using a two-level logic minimizer such as ESPRESSO [3] and if minimization results in compression, the minimal two-level sum of products form of the function is encoded to get the compressed image.

Logic minimization can be used to handle mixed type blocks of larger sizes than is practically possible with Huffman coding and in combination with block coding can yield significant compression on gray-level images [2]. However, the main bottleneck in the use of the logic minimization based method is the large compression time required by the method, because of the use of the logic minimizer ESPRESSO and the file manipulation overheads associated with it.

In this paper we will show the potential of simple minterm coding of the pixel values without going for logic minimization, thus reducing the time requirements of compression. The results obtained are better as compared to our previous results of logic coding both in terms of compression ratio and compression time. The compression time has improved drastically over our previous methods. The results are also comparable to the lossless mode of JPEG in terms of compression ratio. Unlike JPEG, where one has to experiment with different available predictors to obtain the best compression possible, our scheme automatically adapts to the

*Currently with Silicon Automation Systems Ltd., Bangalore 560 008, India, *debu@sas.soft.net*

†Currently with Department of Electrical Engineering, Regional Engineering College, Calicut 673 601, India, *jacob@vishak.reccal.ernet.in*

0-7803-3676-3/97/$10.00

bit plane statistics to reach the best result.

2 Basic Definitions

We first define a few important terms required for explaining our compression scheme. A Boolean *switching function* $\mathbf{F}$ is a mapping $\mathbf{F} : \mathbf{B}^N \rightarrow \mathbf{B}$, where $\mathbf{B} = \{0, 1\}$. 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*. 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. We define *compression ratio* as,

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

3 Auto-adaptive Minterm Coding

We present here a scheme of compression, which uses the concepts of minterm coding [2] derived from logic coding, and a variable block sized approach to adapt to local statistics, derived from auto-adaptive block coding [5]. An image compression scheme using variable block size segmentation is presented in [6].

In an image bit plane the local statistics vary from place to place. As a result, the optimum choice of block size for the purpose of encoding also varies with the local statistics. Furthermore the block size to be used should be different for the different bit planes of the image, for the reason that the activity within a bit plane depends on the relative position of the bit plane. More specifically, as we move toward the MSB (Most significant bit) bit plane, the activity within the bit planes decreases and hence we can use larger block sizes. In contrast, on moving towards the LSB bit plane, the activity, in general, increases and hence using smaller blocks will achieve a higher percentage of compressible blocks, which can in turn result in higher compression.

In auto-adaptive block coding approach similar concepts are used, but only black, white and mixed blocks of size 2×2 pixels are the ones which are considered. Inclusion of other block types makes the Huffman tree too large and hence mixed block sizes other than 2×2 are not considered. We present an approach to include different sized mixed blocks along with blocks of black or white type, without increasing the length of the Huffman tree significantly. Although we do not cover all the possibilities of the mixed block type for any given block size, we will be covering a large proportion of the blocks which are compressible.

4 Compression Scheme

In order to restrict the set of block size choices we will be using only 8×8, 8×4, 4×4 and 2×2 block sizes. We start with 8×8 block size and if required go down to 2×2 block size to adapt to the local statistics. Individual blocks of size 8×8, 8×4 and 4×4 are compressed by using our minterm coding scheme. We do not use minterm coding for the 2×2 blocks, as the relative overhead required is high. Instead we use Huffman coding for the 2×2 blocks.

A block (other than 2×2) is categorized into one of the four possible classes:

- completely black
- completely white
- compressible by minterm coding
- incompressible

For a block which is neither black nor white we try minterm coding. If the number of black or white pixels (either ON-set or OFF-set minterms) within the block is less than or equal to a certain maximum (typically one or two), then we encode the block as a minterm block and also write the minterms on the coded stream. As is evident the minterm coding scheme is a form of co-ordinate coding scheme. The minterms represent the distance of the ones or the zeros from the apex of the block. Since the concepts have been derived from logic coding, we call it minterm coding.

The use of minterm coding has the effect that it covers a large proportion of blocks which are neither black nor white, but are nearly uniform (resulting in small number of either ON-set or OFF-set minterms). Since a limit of small number of minterms is used for any block size, the length of Huffman tree is also not large.

We start with a 8×8 block, check whether the block is completely black, white or mixed type. If the block is not of mixed type then the block is encoded as black or white, as the case may be. If the block is of mixed type, we try minterm coding. If the number of minterms of any one color (black or white) within the block is found to be less than the maximum number of minterms allowed for that block size, then the block is classified as a (minterm) compressible block and we go for minterm coding of that block. Otherwise the block is segmented into two 8×4 blocks and the above steps of classifying into black, white, compressible or incompressible block is repeated. The 8×4 blocks are encoded if found black, white or compressible, otherwise each incompressible block is further segmented into blocks of size 4×4 and the process is repeated. If a 4×4 block is found incompressible, then we divide it into four blocks of size

2×2 each. Each of these four blocks are Huffman encoded as events from among the set of 16 possible events for a 2×2 block.

For optimum bit allocation, Huffman coding is used for the various possible events. The coding is done with two different schemes, one generally suited for the higher bit planes and the other one suited for the lower bit planes. We discuss below these two coding schemes and the choice of the scheme used for the different bit planes.

4.1 Coding Scheme I

In this scheme each block is mapped as a possible event from among the set of the following events.

- completely black
- completely white
- minterm compressible with 1 or 2 ON-set minterms (two events)
- minterm compressible with 1 or 2 OFF-set minterms (two events)
- incompressible

As can be seen, we have put a limit of two minterms on the maximum allowed minterms in a block, to qualify the block as a minterm compressible block. These events are Huffman coded using the statistics of the bit plane and then used to encode the type of the block. In this coding scheme we go level by level. We start with 8×8 block size. If it is not an incompressible block then we simply encode it and then proceed on to the next 8×8 block. In case of the block being minterm compressible the minterms are also written on the encoded stream. Otherwise if the 8×8 block is incompressible then we encode it as an incompressible block, segment it into two 8×4 blocks and then encode the two 8×4 blocks individually. If any of the 8×4 blocks is encoded as an incompressible block then for that block we go on for further segmentation and encoding. The scheme thus uses a tree structured coding scheme. The segmentation and hence encoding continues on a branch until we reach a compressible block on that branch, or until the block size reduces to 2×2. In the latter case we do not try to determine the nature of the block in the above manner. The four 2×2 blocks of the incompressible 4×4 block are encoded as Huffman events.

4.2 Coding Scheme II

In the second coding scheme, we do not follow a level by level approach. Instead the information of the size and the type of the block is stored in the same code, which enables us to point directly to the required block size. The code tree has all the events of the previous code tree, except for the incompressible block event, but all the events are repeated for the block sizes of 8×8, 8×4 and 4×4. To point to the 2×2 block we also include the 16 possibilities of the 2×2 block as events. The first 2×2 block of the incompressible 4×4 block is an event in the code tree discussed above, and the subsequent three 2×2 blocks are events in a separate code tree of length 16. Here we have now the following set of events in the first code tree:

- completely black 8×8 block
- completely white 8×8 block
- 8×8 block, minterm compressible with 1 or 2 ON/OFF set minterms (4 events)
- completely black 8×4 block
- completely white 8×4 block
- 8×4 block, minterm compressible with 1 or 2 ON/OFF set minterms (4 events)
- completely black 4×4 block
- completely white 4×4 block
- 4×4 block, minterm compressible with 1 ON/OFF set minterm (2 events)
- 16 events of the first 2×2 block in a 4×4 incompressible block

As would have been noticed, we have used a limit of one minterm only for the 4×4 block, unlike the previous coding scheme where a limit of two minterms is imposed on all block sizes. The modification helps in getting a better encoding in case of less compressible bit planes for which this coding scheme is more suitable.

For encoding any block we do not have to go level by level. Thus if there is an 8×8 incompressible block, then we don't represent explicitly that its incompressible, instead encode the 8×4 blocks if it is found compressible. Unlike the previous case, the size and the nature of the block are both absorbed in the Huffman code. The position of the block in the bit plane is determined by its context in the coded stream, as the encoding (as well as decoding) proceed along a predetermined structure.

4.3 Choice of the Coding Scheme

In the last two sections we have proposed two coding schemes to be used with the compression scheme. In the first scheme the individual codewords are smaller,

Image		lena	boats	girl	baboon
Auto-adaptive Logic Coding	compression ratio(%)	27.8	31.3	**43.7**	**14.2**
	compression time(sec)	5.1	5.9	4.9	5.1
	decompression time(sec.)	6.3	8.0	5.9	6.9
PVRG-JPEG (lossless mode)	compression ratio(%)	22.7- **29.4**	24.9- **32.9**	30.7- 38.1	6.7- 13.7
	compression time(sec)	0.4	0.4	0.4	0.4
	decompression time(sec.)	0.3	0.3	0.3	0.3

Table 1: Results of the compression experiment

since the number of possible Huffman events are smaller, but at the same time the effective code length for the smaller block sizes is large as we have to go level by level. In the second scheme the effective codeword length is smaller for the smaller blocks (than the first scheme) and for the larger blocks it is comparatively larger (owing to the larger code tree in the second case). Thus for the higher bit planes, where there is a chance of having large number of bigger sized compressible blocks, it is advantageous to use the first scheme. On the other hand the second scheme is more suited for the lower bit planes, as the compressible block size, in general, is smaller for the lower bit planes.

For any image, we use a combination of the two coding schemes. Typically the first three bit planes (towards the MSB bit plane) are encoded by the first scheme and the rest by the second scheme, if they are found compressible.

To decide the coding scheme to be used for a certain bit plane, we need to determine the activity within a bit plane. A good representation of activity within a bit plane has been found to be the *transition product count*, defined as the product of vertical transition count and the horizontal transition count. The vertical transition count is the sum of the transitions along the columns of the image bit plane. Similarly the horizontal transition count is the sum of transitions along the rows of the image bit plane. Here we have used the transition product count as the parameter to decide the coding scheme to be used. For 256×256 images we have empirically determined the value of 120×10^6 *transitions* as the transition product count threshold for deciding the coding scheme to be used. The threshold value can be decided for any given image size by considering the transition product count statistics of the bit planes from a representative set of images of this size. Thus the low activity bit planes (with transition product count below the threshold) are compressed with coding scheme I and the coding scheme II is used to compress the high activity bit planes (with transition product count above the threshold). It should be further noted that the loss in compression ratio due to a wrong choice of the coding scheme for a certain bit plane goes on decreasing as we move towards the threshold. This makes it clear that the threshold is not a stringent one and a small change in the threshold itself would not effect the compression significantly.

Further for the very high activity bit planes (say, with a value of transition product count greater than 800×10^6 *transitions*), the compression obtained is almost negligible (the coding scheme II being used for such bit planes). So we have set a threshold of 850×10^6 *transitions*, to decide as to whether to try compression on that bit plane or not. In case of bit planes with a transition product count greater than this value, we store the bit planes as such. This reduces the total compression time requirements.

5 Experimental Results

The compression and decompression techniques have been implemented in 'C' on an IBM RS-6000/580 workstation running under UNIX, and tested on standard images *lena*, *baboon*, *boats* (nic.funet.fi:/pub-/graphics/misc/test-images) and *girl* (eedsp.gatech.edu:/database/images), of sub-sampled size 256×256 pixels and 8 bits/pixel. The preprocessing step of recoding and Gray coding of the pixel values are applied before splitting each image to its individual bit planes.

Table 1 gives the results of the compression experiment as well as a comparison with the lossless mode of JPEG. We have used the PVRG-JPEG

(Portable Video Research Group) Codec1.1 (havefun.stanford.edu:pub/jpeg/JPEGv1.2.tar.Z). The compression ratios obtained by auto-adaptive logic coding are comparable to those obtained by JPEG. The result of logic coding are superior to the best of JPEG results for *girl* and *baboon*, while for other two images our results are within 1.6% of the best JPEG results.

It should be noted that in our scheme we need not worry about the encoder to be used, whereas JPEG does not provide any method of knowing beforehand a suitable predictor for the image. Variance of the compression obtained by the set of predictors in case of JPEG is very large, whereas auto-adaptive logic coding gives a single value. The time requirements of our prototype encoder and decoder are presently higher than JPEG, but can be reduced significantly by efficient coding.

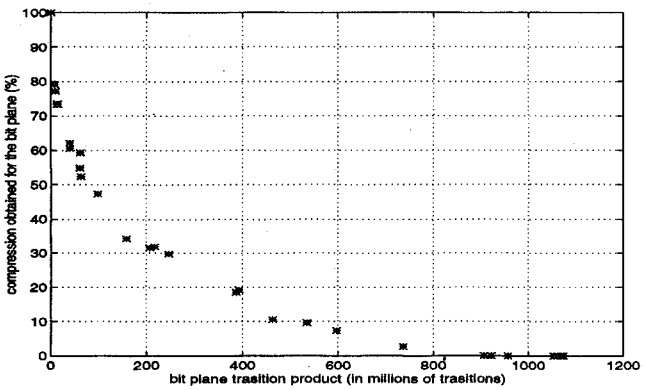

Figure 1: Relationship between transition product count and compression ratio

We have plotted the compression obtained versus the transition product count for the 32 bit planes of the four 256×256 pixel (8 bits per pixel) experimental images in Figure 1. As can be seen there is a good correlation between the amount of compression achievable and the transition product count for the bit plane. The figure further emphasize the fact that, irrespective of the image, transition product count is closely related to the compression achievable on a certain bit plane.

6 Conclusion

The method of image compression discussed above shows a practical way of encoding a large proportion of blocks other than the black or white blocks, without increasing the length of Huffman tree to large values. The performance of the scheme is comparable to JPEG in terms of compression ratio. Time requirements of the scheme are presently higher than JPEG but can be reduced by efficient coding. The advantage of the scheme is that it automatically chooses the best possible encoder (by choosing one of the two coding schemes for a bit plane), and reaches the optimum value of compression achievable with the scheme, whereas in the case of JPEG, the variance of compression that can be obtained with the seven predictors is large and the best possible predictor for a certain image can not be determined beforehand.

The link between the transition product count and the compression ratio obtained for a bit plane suggests the use of some reversible Boolean transform techniques to reduce the product of transitions within a bit plane. This can further enhance the compression ratio. The compression ratio can be further improved by inclusion of larger block sizes (16×16, 16×8), especially for the higher bit planes.

In the above scheme we had used minterm encoding only. The use of minimization with *ESPRESSO* can help to further improve compression but was not used due to the high time overhead associated with logic minimization. The use of some form of relative encoding for the correlated minterms is another possibility which can enhance the performance without affecting the time requirements. These are some of the issues which need to be investigated further.

References

[1] J. Augustine, W. Feng, A. Makur and J. Jacob, "Switching Theoretic Approach to Image Compression," *Signal Processing (Elsevier Science)*, vol. 44, pp. 243-246, June 1995.

[2] J. Augustine, "Switching Theoretic Approach to Image Compression," *Ph.D thesis*, Department of Electrical Communication Engineering, IISc. Bangalore, May 1996.

[3] R. K. Brayton, G. D. Hachtel, C. T. McMullen, and A. L. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis*, Kluwer Academic Publishers, Boston, 1984.

[4] P. Fränti, "A fast and efficient compression method for binary images," *Signal Processing: Image Communication*, 6(1994), pp. 69-76.

[5] M. Kunt and O. Johnsen, "Block Coding of Graphics: A Tutorial Review," *Proc. of The IEEE*, vol. 68, No. 7, pp. 770-786, July 1980.

[6] J. Vaisey and A. Gersho, "Image Compression with Variable Block Size Segmentation," *IEEE Trans. on Signal Processing*, vol. 40, no. 8, pp. 2040-2060, 1992.