Logic Minimization Based Approach for Compressing Image Data

Jacob Augustine
Dept. of Electrical Comm. Eng.
Indian Institute of Science
Bangalore 560 012 India

Wen Feng*
Department of Electrical Eng.
Anhui Mechanical and Electrical Institute
Wuhu, Anhui 241 000 China

James Jacob
Dept. of Electrical Comm. Eng.
Indian Institute of Science
Bangalore 560 012 India
james@ece.iisc.ernet.in

Abstract

We propose a novel approach for the lossless compression of binary images using logic minimization. The image is divided into windows or blocks of size $r \times c$ pixels and 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 logic minimizer. To reduce the bits required to encode the minimized cubes (product terms), a code set which satisfies the prefix property is used. If this technique fails to produce compression for a window, the pixels are stored as such. The main motivation of the work has been to investigate the potential of logic minimization as a tool for image data compression. Our technique outperforms UNIX compress in terms of compression ratio on most of the test images. The compression scheme is relatively slower while the decompression time is comparable to that of UNIX compress.

1 Introduction

Compression of image data is important from the point of view of storage and transmission, as it helps to achieve savings in storage space and transmission time. Lossless compression techniques permit the recovery of an exact copy of the original image, whereas the lossy techniques permit only an approximation to be recovered. Well known lossless compression techniques are Huffman coding, Arithmetic coding, Lempel-Ziv(LZ) algorithm [4] and Run-length coding [5]. An implementation of the Lempel-Ziv-Welch(LZW) algorithm is available as a utility called *compress,* with most of the UNIX installations.

Images with no color can be represented mathematically as a function of two spatial variables, i.e., **F**(x,y). The value of this function at a point is called the *gray-level* or *brightness* of the image at that point. *Digital image* is generated by using a sampling process, to extract from the image a discrete set of real numbers or samples, and then by applying quantization process, to yield numbers having a discrete set of possible values. The elements of a digital image are called *pixels.* If there are only two gray-levels, *black* and *white,* they are usually represented by 1 and 0 respectively, and the image is called a *binary* or *bilevel image.* Facsimile data, finger prints and printed materials could be cited as examples for binary images [5].

Two-level logic minimization seeks a logic representation for a given switching function with minimum number of implicants and literals. In digital circuit design this attempt is important because it leads to a greater speed of operation, lower silicon area, higher yield and enhanced reliability [2].

In this paper, we propose a new technique based on logic minimization, for the lossless compression of binary images. Image is divided into blocks of size $r \times c$ pixels and each block is converted into a Boolean switching function in cubical form, which is then minimized using the well known cube-based two-level logic minimizer ESPRESSO [2]. Then the minimized cubes which represent the implicants (product terms) are coded with a code set which satisfies the prefix property, to obtain the compressed data. A decompression scheme using cube *subsuming* operation is also illustrated. The results presented are indicative of the potential of logic minimization as an effective tool for compressing image data.

Performance of the algorithm on several binary images generated from gray-level images, and a comparison with *UNIX compress* is given.

2 Background

For the sake of clarity of the material presented, terms frequently appearing are briefly explained.

Definition 1: A Boolean *switching function* **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 $\mathbf{A} = \{a_1, a_2, ..., a_N\}$, 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 [1, 3].

*During the course of this work, Wen Feng was a visiting Scholar to the Dept. of Electrical Comm. Eng., Indian Institute of Science, Bangalore under the INDO-CHINA Cultural Exchange Program.

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,

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

3 The compression scheme

Block diagrams of our data compression and decompression schemes are given in Figure 1. The compression scheme consists of the steps of image splitting, function generation, logic minimization and cube encoding.

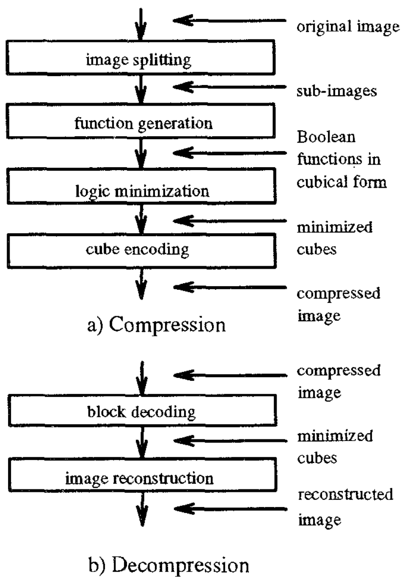
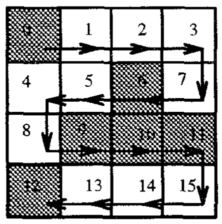


Figure 1: Image compression and decompression schemes.

In image splitting an image is divided into sub-images (windows or blocks) of size $r \times c$ pixels. This is done because of the fact that the correlation on the image is generally local so that smaller blocks often result in better compression. It also helps the logic minimizer to minimize the functions faster as the search space is reduced.

Function generation generates a switching function in cubical form for each window, 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 as shown in Figure 2a, to ensure that pixels at the ends of consecutive lines are mapped to logically adjacent codes. An image with M pixels is converted to a function of $\lceil \log_2 M \rceil$ variables. Figure 2 shows the conversion of a binary image with 16 pixels into a 4 variable switching function according to the scheme. Fig. 2a shows the binary image where the shaded squares correspond to black pixels. Fig. 2b illustrates the assignment of minterms to the pixels, 2c the Karnaugh map of the function and 2d and e the unminimized and minimized ON-set cubes respectively.

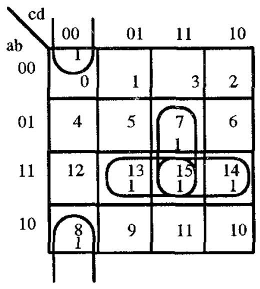
Logic minimization is performed on the generated functions using the two-level cube-based logic minimizer ESPRESSO to find the equivalent minimized cubical representation. For a particular function, in general, the number of cubes in its minimized ON-set and OFF-set are different. Better compression can be achieved by choosing the set with lesser number of cubes, since both represent the same function. An additional bit is needed to encode this information.



a) Binary image

pixel no.	minterm abcd	f
0	0000	1
1	0001	0
2	0011	0
3	0010	0
7	0110	0
6	0111	1
5	0101	0
4	0100	0
8	1100	0
9	1101	1
10	1111	1
11	1110	1
15	1010	0
14	1011	0
13	1001	0
12	1000	1

b) Switching function



c) Karnaugh map

0000
0111
1101
1111
1110
1000

d) ON-set

x000
x111
111x
11x1

e) Minimized ON-set

Figure 2: Function generation following cyclic code and line reversal.

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. If the number of cubes for an N variable function is $C\,(0 \leq C \leq 2^{N-1})$, then the worst case occurs when these three symbols have the same frequency of occurrence, i.e., $CN/3$. In this case, the total number of bits needed to represent the function is,

$$CN/3 + 2CN/3 + 2CN/3 = 1.67CN \text{ bits.}$$

Reduction occurs in the number of bits if $1.67CN < 2^N$. Additional two bits are needed to encode the information about the allotment of 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 original image of size $R \times C$ pixels is divided into windows of size $r \times c$ pixels. The compressed image has a global header portion followed by the encoded windows. The global header has N1, N2, n1 and n2 bits to indicate R, C, r and c respectively. For the images considered here N1, N2, n1 and n2 are set to 7 bits, because the bit-maps are of size 128×128 pixels. From the knowledge of the image size $R \times C$ and the window size $r \times c$, the number of encoded windows and the number of variables required for the function that represents a window can be known. Each window is encoded according to a specific format. The first two bits, as given below, indicate the encoding scheme used for each window.

- 00 : Window without minimization. $r \times c$ bits are stored as in the original window. Algorithm has failed to produce compression for this window.
- 01 : Window is identical to the previous window. For a window on the left side of the image the one just above is taken as its previous window.
- 10 : Window is minimized and encoded as cubes.
- 11 : Difference (xor) of the current and the previous windows is minimized and encoded as cubes because the difference has resulted in better compression.

In the last two cases the next bit called *phase* is used to indicate whether ON-set/OFF-set is coded. In these cases the next two bits as shown below, are used to indicate the allotment of prefix codes.

- $00: 0 \rightarrow 0,\ 10 \rightarrow 1,\ 11 \rightarrow X.$
- $01: 0 \rightarrow 1,\ 10 \rightarrow X,\ 11 \rightarrow 0.$
- $01: 0 \rightarrow X,\ 10 \rightarrow 0,\ 11 \rightarrow 1.$

The unused code 11 is used to indicate the case where the number of minimized cubes is zero. This situation arises when a window consists of all zeros or all ones. In such cases no more bits are needed to encode the window.

If the number of cubes of the minimized function is not zero and if a data expansion has not occurred, then the next m bits are used to indicate the number of cubes. Encoded cubes are placed after this. Figure 3 shows the format of the encoded window.

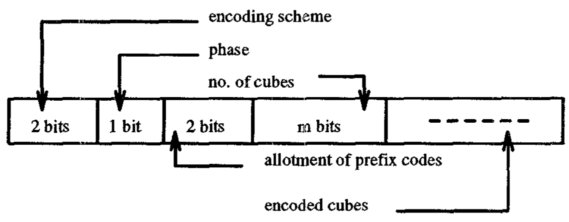


Figure 3: Window format

5 Decompression scheme

Decompression procedure consists of the steps of block decoding and image reconstruction. In block decoding a block of data corresponding to a window is located on the compressed image and retrieved. If the data is in the form of minimized set of cubes, image reconstruction determines the pixel values of the corresponding window from the given minimized ON-set (OFF-set) of the function. This is done by computing the value of the function for each of its possible minterms (following the cyclic code) using the cube *subsuming* operation given in Definition 6, and assigning the values to the corresponding pixels of the window. If a minterm subsumes the set of minimized cubes of the ON (OFF)-set of a window, then the pixel corresponding to the minterm evaluates to 1(0). For the windows encoded without minimization, image reconstruction is straightforward and the subsuming operation is not invoked.

	$\sqsubseteq$	b_i: 0	1	X
a_i	0	e	ϕ	e
	1	ϕ	e	e
	X	ϕ	ϕ	e

Table 1: Subsuming operation

Definition 6: Let **A** and **B** be two cubes of N variables, where **A**$= \{a_1, a_2, ..., a_N\}$ and **B**$= \{b_1, b_2, ..., b_N\}$, **A** *subsumes* **B** (denoted as $\mathbf{A} \sqsubseteq \mathbf{B}$), if $a_i \sqsubseteq b_i = e$ for $i = 1$ to N. In other words, $\mathbf{A} \sqsubseteq \mathbf{B}$, if all minterms of **A** are contained in **B**. The co-ordinate subsuming relation is defined in Table 1.

6 Experimental results

The proposed compression and decompression schemes have been implemented in 'C' on an IBM RS-6000/580 workstation with UNIX operating system, and tested on a set of binary images. The test images of size 128×128 pixels (16k bits) were generated from gray-level images with 256 gray-levels (8 bits/pixel) by applying thresholding. Thresholding is the process of generating a binary image from the corresponding gray-level image by mapping all gray-levels above the

Table 2: Results of the compression experiment

Test image	Logic coding: window size in pixels (number of variables in the corresponding Boolean function)										UNIX compress	
	32 × 32 (10)		16 × 16 (8)		8 × 8 (6)		8 × 4 (5)		4 × 4 (4)			
	CPU sec.	compr. ratio %	CPU sec.	compr. ratio %	CPU sec.	compr. ratio %	CPU sec.	compr. ratio %	CPU sec.	compr. ratio %	CPU sec.	compr. ratio %
girl	18	37.41	8	47.99	7	56.34	8	**56.83**	10	56.41	<0.1	50.24
unix	9	66.37	7	71.22	5	**74.59**	5	73.70	7	70.74	<0.1	68.55
elephant	16	28.95	9	39.10	8	48.33	9	**50.21**	12	48.26	<0.1	43.16
inst1	10	62.16	6	66.26	6	70.17	6	**71.15**	8	66.42	<0.1	59.71
lawn	20	2.22	11	6.21	10	14.03	12	16.52	18	17.57	<0.1	**20.55**
city	9	66.97	6	70.34	6	**73.43**	6	71.95	8	66.35	<0.1	65.08
checker	2	**99.57**	24	98.02	6	94.65	4	91.53	5	83.23	<0.1	92.67
spanner	4	90.69	3	**90.83**	3	89.98	2	87.92	3	81.34	<0.1	84.22

CPU time in seconds is on IBM RS-6000/580

threshold to 0 (white) and below the threshold to 1 (black).

Our compression technique is tentatively called *Logic coding.* Table 2 gives the results of the compression experiment as well as a comparison with the performance of UNIX compress. Test images were divided into 16, 64, 256, 512 and 1024 windows of rectangular shape, such that their sizes are 32×32, 16×16, 8×8, 8×4 and 4×4 pixels respectively. The Boolean functions generated from each window is written to a file in the required cubical input format and logic minimization is performed by invoking ESPRESSO 2.1. Logic coding outperforms UNIX compress in terms of compression ratio, on all the test images except *lawn.* Larger window sizes yield better compression ratio only in the case of relatively simple images like *spanner* and *checker.* The best result obtained for each image is highlighted using boldface in Table 2. The CPU time reported is for the compression of the entire image which is presently rather high. It does not reflect the efficiency of the algorithm, as ESPRESSO is used as a stand alone package and considerable communication overhead is incurred. Results of the UNIX compress were obtained by running it on the 128×128 pixel image files, after packing 8 consecutive bits into a byte. Considering all the test images, the average compression ratio obtained by our approach, taking the best result for each image is **66.77%**, whereas the average turns out to be **60.52%** for UNIX compress.

The runtime of the decompression scheme of Logic coding and UNIX compress is **less than 0.1 seconds** in all the cases. As the cube subsuming operation is relatively simple, the time required to recover the image from its compressed form is significantly less and comparable to that of UNIX compress.

7 Conclusion

A novel lossless data compression technique for binary (bi-level) images using logic minimization has been presented. Our compression technique called Logic coding performs better than UNIX compress for most of the test images in terms of compression ratio. The execution time of the compression scheme of Logic coding is rather high at present, which is expected to improve with further effort. However this work points to the possibility of using logic minimization as an effective tool for digital image compression. Further study is needed to properly understand the behavior of this algorithm on different types of images. Also a method needs to be devised to find the most economical window size for a particular image.

Currently we are trying to improve this algorithm and to extend the approach to gray-level images.

Acknowledgment– Authors are grateful to Anamitra Makur, Dept. of Electrical Comm. Eng., Indian Institute of Science for the fruitful discussions. Jacob Augustine is with the Dept. of Electrical Eng., Regional Engineering College, Calicut, Kerala, India and currently a research student at the Dept. of Electrical Comm. Eng., Indian Institute of Science under the QIP scheme.

References

[1] N. N. Biswas, *Logic Design Theory,* Prentice Hall Inc., NJ, 1993.

[2] 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.

[3] M. A. Breuer, *Design Automation of Digital Systems Vol-1: Theory and Techniques,* Prentice Hall Inc., Englewood Cliffs, NJ, 1972.

[4] A. Gersho and R. M. Gray, *Vector Quantization and Signal Compression,* Kluwer Academic Publishers, Boston, 1992.

[5] N. S. Jayant and P. Noll, *Digital Coding of Waveforms: Principles and Applications to Speech and Video,* Prentice-Hall Inc., NJ, 1984.