A NEW CASE FOR IMAGE COMPRESSION USING LOGIC FUNCTION MINIMIZATION

Behrouz Zolfaghari[1] and Hamed sheidaeian[2]
[1] Department of Engineering, Islamic Azad University, Garmsar Branch, Iran
zolfaghari@iust.ac.ir
[2] Department of Engineering, Islamic Azad University, Garmsar Branch, Iran
sheidaian@garmsar-iau.ir

ABSTRACT

Sum of minterms is a canonical form for representing logic functions. There are classical methods such as Karnaugh map or Quine–McCluskey tabulation for minimizing a sum of products. This minimization reduces the minterms to smaller products called implicants. If minterms are represented by bit strings, the bit strings shrink through the minimization process. This can be considered as a kind of data compression provided that there is a way for retrieving the original bit strings from the compressed strings. This paper proposes implements and evaluates an image compression method called YALMIC (Yet Another Logic Minimization Based Image Compression) which depends on logic function minimization. This method considers adjacent pixels of the image as disjointed minterms constructing a logic function and compresses the 24-bit color images through minimizing the function. We compare the compression ratio of the proposed method to those of existing methods and show that YALMIC improves the compression ratio by about 25% on average.

KEYWORDS

Logic Function Minimization, Lossless Image Compression

1. INTRODUCTION AND BASIC CONCEPTS

Many research works has focused on developing various data compression (especially image compression) techniques in recent years [1, 3, 4 and 7-10]. Among other approaches, many proposed techniques depend on logic function minimization [11, 12, 19, and 25]. Some of these approaches are lossy [24, 25] and some of them directly depend on lossless schemas like Huffman coding [11, 25]. Such techniques treat bit sequences as logic terms and try to simplify the logic expressions constructed of these terms. Simplification of logic expressions causes some terms and variables to be discarded from the expression and this can compress the input data by discarding the corresponding bits or bit sequences. For example consider the bit stream ***S=1110111111011100***. This stream can be divided into four bit sequences each of which consists of four bits (called ***quartets***). The first quartet (1110) represents a 4-variable minterm such as $xyz\ \overline{w}$. Similarly, the next three quartets represent $xyzw$, $xy\ \overline{z}\,w$ and $xy\ \overline{z}\,\overline{w}$ respectively. Thus, we can model the whole bit stream by a disjunctive logic expression as follows.

$$S = xyz\ \overline{w} + xyzw + xy\ \overline{z}\,w + xy\ \overline{z\,w}$$

Equation (1)

DOI : 10.5121/ijma.2011.3204

The above logic experiment be simplified to $S_s = xy$ or equivalently $S_s = 11$ using Karnaugh map or the Quine–McCluskey method. If it was possible to regenerate S from S_s, we had come to a compressed form for S. But due to the following two reasons, S cannot be fully regenerated from its simplified form S_s:

First: $S_s = 11$ only characterizes a 2-variable minterm. But it contains no information concerning the positions of its constructing variables in the original 4-variable minterms. For example, it can represent xz or yw. If it is assumed to represent xz then it will be considered as the simplified form of $E_1 = x\bar{y}z\bar{w} + x\bar{y}zw + xyz\bar{w} + xyzw$. But if it is considered as yw, it will be the simplified form of $E_2 = \bar{x}y\bar{z}w + \bar{x}yzw + xy\bar{z}w + xyzw$. In fact S_s cannot distinguish E_1 from E_2.

Second: S_s does not convey anything regarding the order of the 4-variable minterms (quartets) in the original image. For example, suppose that we know $S_s = xy$. In this case, the following 4-variable minterms will be generated by S_s: $xy\bar{z}\bar{w}$, $xy\bar{z}w$, $xyz\bar{w}$ and $xyzw$ which are equivalent to the quartets 1100, 1101, 1110 and 1111 respectively. Now since the order of the minterms in the original image is not known, we can regenerate S as ***1100110111101111*** or ***1111110011011110***.

For the two reasons mentioned above, we have to add some extra bits to $S_s = 11$ which make it possible to exactly regenerate S. YALMIC (Yet Another Logic Minimization Based Image Compressor) is a novel lossless image compression algorithm based on logic minimization. In this algorithm, segments of the image are first converted to hypothetical logic expressions and the expressions are simplified using standard methods (Simplifying logic function has been the topic for a lot of research in recent years [2, 6]. There are many methods in the literature which have been proposed for this purpose. Each of these methods can be used here). Then the simplified forms of the expressions are stored in the target compressed file along with some extra bits which allow the primary segment to be fully regenerated from its simplified form. YALMIC has been written in C#. Its screenshot is shown in figure 1. YALMIC computes the size of the compressed image and uses the following equation to calculate the ***compression ratio***.

$$CR = \frac{U - C}{U} = 1 - \frac{C}{U} \qquad \text{Equation (2)}$$

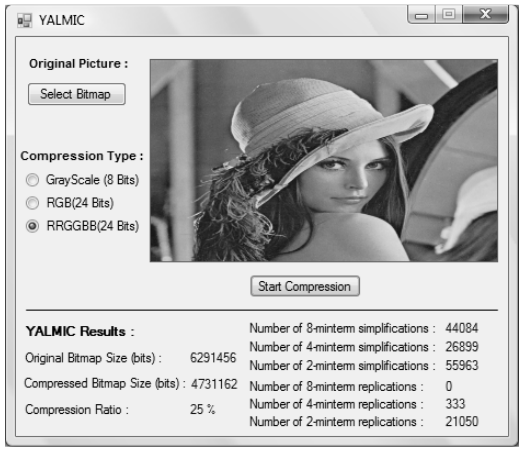

Figure 1. The screenshot of the software implementing YALMIC

In the above equation, U is the size of the original uncompressed file and C is that of the compressed file. The ration $\frac{C}{U}$ is referred to as the ***compression factor*** in this paper.

The rest of this paper is organized as follows. Section 2 is discusses previous related works. Section 3 introduces the proposed compression algorithm. Section 4 explains the decompression algorithm. Section 5 evaluates the compression ratio of the algorithm through analytical modelling and experimental results. This section compares the compression ratio of the proposed algorithm to that of previous algorithms. Section 6 concludes the paper and suggests further works.

A brief version of this paper has been published in proceedings of 2010 IEEE international conference on computer engineering and technology [3].

2. RELATED WORKS

There have been a lot of research works in the recent years [1, 3, 4, and 5] which are based on considering information contained by logic functions and using the properties of these functions for compressing different kinds of data. But the most relevant works to our proposed method are discussed below.

Kumar, et al. [4] proposed a logic minimization based approach to lossless compression of gray-scale images which uses a 2-dimentional differencing operation based on logical XOR in order to increase the compression ratio. This operation is performed on adjacent bit planes extracted from rectangular blocks of the image. This approach uses the Quine-McCluskey algorithm for minimizing the logic functions obtained from the image in cubical form. They compared the compression ratio of their approach to that of LOCO-1 and WinZip using five test images: Airplane, Boat, Moon, Couple and Man. They showed that their approach performs better than the two others for three out of the five images. They mentioned that their approach attains higher compression ratios for images with smaller numbers of different intensity levels.

Qawasmeh, et al. [5] proposed a method for data compressing data streams using logic function minimization in the form of sum of products. Their method depends on a frequency

table which is used by Huffman coding. This table keeps frequencies of prime implicants which have been counted during a preprocess phase. Their method converts blocks of the data stream to logic functions represented by sum of products. Prime implicants are extracted from the functions using Quine-McCluskey tabulation method. These prime implicants are searched in the frequency table and compressed using Huffman coding. They showed that the number of prime implicants in a 16-bit block obeys a nearly normal distribution with the mode equal to 4 prime implicants per block. They demonstrated that the proposed method exhibits the best compression ratio for data streams in which about %60 of bits are equal to one. They also showed that the compression ratio of their method is almost not affected by the size of the file to be compressed. Their method shows better compression ratios for smaller block sizes.

Reaz, et al. [14] implemented boolean function classification schemes on Altera FLEXIOK FPGA and used these classifications along with Huffman coding in order to compress input data streams in a lossless way. They gained average compression ratio between 25% and 37.5% for various text strings. They argued that using FPGA for this purpose causes speed up in hardware realization. They used various classical and novel boolean function classification schemes. They used VHDL to verify their algorithm and create the netlist of required digital cells. They showed that their method can exhibit a maximum compression ratio of 87.5% when the input characters are all encoded to all-1 patterns. They performed their algorithm on the FPGA board with a minimum resource usage of 63.5% and a maximum frequency of 27.9MHz.

Falkowski [13, 15, and 20] presented a lossless method based on representations of logic functions for compressing gray-scale images. The first phase of this method performs an intensity coding after file extraction to prepare the image for compression. Then a prediction process is run and the residuals of the predictions are mapped and split into bit-planes. The compression is the last phase which is performed on the bit-planes using compact coding. File extraction involves extracting image pixels and converting each of the image dimensions to the next higher multiple of eight. Within the intensity level coding, the pixel values are rearranged in continuous ascending order. Some minor bit overhead is posed to the image here. The prediction process is performed in order to reduce the correlation which exists between adjacent pixels of the image. Three different prediction methods are used here. The mapping process converts negative and positive prediction residuals to positive numbers. The bit-plane splitting selects corresponding bits of pixels and converts an M*N image consisting of k-bit pixels to k bit-planes each M*N bits long. This technique helps improve compression ratio by taking advantage of large uniform areas existing in bit planes. In the coding phase, the horizontal and vertical transition counts are computed for each of the bit-planes. Then the result of the comparison between the average transition count and some predetermined thresholds is considered as the basis for selecting among a number of different coding schemes. They compared the compression ratio of their method with those of WinZip, BTPC, S+P and LOCO-L.

Agaian, et al. [15,16] explained that the dominating lossless image compression algorithms such as Huffman coding, LZW, Arithmetic coding and run length coding do not use transforms unlike the dominating lossy algorithms such as JPEG. They considered the application of logical transforms to lossless image compression. They designed a series of logical transforms for the purpose of boolean minimization. These transforms does not require to be multiplied in the bit planes of the image unlike the case of lossy compression. They argued that Karnaugh map as a logic simplification tool has some inefficiencies in the domain of lossless compression.

The first is that the time complexity of Karnaugh map rapidly grows with the size of the truth table. The second is that there exists no fast implementation for Karnaugh map and this slows down the compression process. They enumerated several advantages for their method: unlimited size of boolean functions, existence of fast implementations, higher compression ratios, having no need for time-consuming multiplication operations, amenability for hardware implementation and the potential for being applied to other types of data (e.g. text). Their method runs a preprocessing consisting linear prediction. They showed that the compression can attain better compression ratio in contrast with LZW, RLE, arithmetic coding and WinZip for Lena and Girl.

Villarroya, et al. [19] argued that logic function simplification in algebraic forms suffers high time complexity. They presented the logic functions obtained from images in the form of OBDDs (Ordered Binary Decision Diagrams) in order to reduce this complexity. They added arithmetic coding to logic simplification to further reduce the redundancy of the decision trees. They modified the method proposed by Starkey, et al. [22] in order to simplify the OBDDs. This modified method (named SF-OBDD) represents the image in the form of sequential functions. They showed that this method improves compression ratio in addition to reducing compression time. They used CCITT gray scale test images (ptt1,…, ptt8) to evaluate their method and compare it with two previous methods. The first of these methods (CF-AE) presents the image in the form of Combinational Function and simplifies the logic functions in the form of Algebraic Expressions. The second (CF-OBDD) presents the Combinational Functions in the form of OBDDs. These two methods were evaluated and compared in their previous work [26]. They gained an average compression ratio of 18.40% which is better than those of CF-AE (11.89%) and CF-OBDD (14.68).

Pramanik, et al. [23] presented an adaptive lossless image compression method based on logic coding and auto-adaptive block coding as well as Huffman coding with a compression ratio comparable to that of JPEG. The main advantage of this method is that automatically tunes itself to reach the highest possible compression ratio. Adaptive block coding is a modified variant of block coding [30] that decomposes bit-planes into fixed-length 2*2 pixels blocks of three different types: *all-white*, *all-black* and *mixed*. They argued that avoiding variable-length blocks prevents the Huffman coding tree from getting to large. They also presented an approach for including a large proportion of variable-length mixed blocks (8*8, 8*4, 4*4, and 2*2) without significant increase in the size of the Huffman tree. They applied the minterm coding to the variable-length blocks but excluded the 2*2 blocks from the minterm coding. They classified the blocks (other than 2*2 blocks) into four separate classes: completely black, completely white, compressible with minterm coding and incompressible. They proposed metrics for selecting between two different coding schemes for different kinds of blocks. They applied their method to Lena, boats, girl and baboon. The average compression ratio was 29.25%. They compared their method with Lossless mode of PVRG-JPEG which gives an average compression ratio of 21.25% to 28.5 for the four mentioned images. They also compared compression and decompression times of their method to those of PVRG-JPEG.

Damodare, et al. [24] proposed a pair of methods for compressing monochrome images. One of these methods is lossy and the other is lossless. Their methods rely on simplifying two-level logic functions in cubical form using ESPRESSO [25]. Their methods partition the image into bit planes which are divided into variable-size blocks in turn. These blocks are minimized as cubical logic functions. The lossless method runs a preprocess on the image which performs a

linear prediction. They showed that the lossless compression method can outperform JPEG from compression ratio point of view. They also demonstrated that the lossy method exhibits a better compression ratio than that of JPG for identical mean square errors. This compression ratio is achieved without distinguishable changes in visual properties of the original image. Their experiments demonstrate that the compression phase of their methods performs a little slower than JPEG while the decompression time is comparable to that of JPEG.

Yang, et al. [12] proposed a lossless data compression method based on multilevel logic function minimization. They argued that multilevel minimization is a better choice for circuit design as well as lossless data compression. Their method uses two tools: binary to Boolean converter and boolean minimizer. The former tool converts to blocks representing multilevel logic functions and the latter minimizes each function using ESPRESSO [29]. They have considered a preprocessing phase in their method which stores differences between consecutive data symbols instead of the symbols themselves. The converter tool partitions the data stream into octets of bits and assigns each bit of the octet to the value of a line in a truth table representing a 3-variable function. This allows multilevel minimization of the hypothetic function using ESPRESSO. The minimizer uses ESPRESSO in ON mode for two level minimization and Berkeley SIS in BLIF [27] mode for multilevel minimization.

Falkowski, et al. [18, 21] considered using Kleene Algebra, Multiple-valued functions and spectral analysis for lossless compression of gray scale images [19]. Their method consists of preprocessing, prediction, splitting of bit planes and variable block-size segmentation and coding .They applied similar ideas to gray scale and color biomedical images [22]. They used Walsh and Reed-Muller transforms for the spectral analysis.

The method proposed in this paper differs in the following points from the methods introduced above. This method is lossless. It does not depend on Huffman coding, arithmetic coding, prediction, differencing, logic transforms or OBDDs. It has not been specifically designed for text or gray scale images and it does not treat blocks of data as truth tables.

3. THE COMPRESSION ALGORITHM

The compression algorithm consists of two phases; the first phase is called the ***minimization phase*** and the second is referred to as the ***block construction*** phase.

In the minimization phase, the original image file is first divided into 8-bit sequences called ***octets***. Each octet is then considered as a minterm of 8 variables which are called b_0 , b_1 , b_2 , b_3 , b_4 , b_5 , b_6 and b_7 with respect to their positions in the minterm. For example, the octet 10110010 is considered as $b_7\overline{b_6}b_5b_4\overline{b_3}\overline{b_2}b_1\overline{b_0}$. In the next step, a hypothetical '+' operator is inserted between each pair of consecutive minterms. The '+' operator represents the logic OR operation. In this step, the image is converted to a disjunctive logic expression represented as a sum of minterms. This sum is then scanned for sets of successive identical minterms or successive minterms which can construct implicants. Each of these sets is called a ***segment***. Each segment should consist of 1, 2, 4 or 8 successive minterms. An 8-minterm segment is preferable to two 4-minterm segments and so on. The block construction phase stores an individual block in the compressed file for each segment detected in this phase. Each block contains a simplified form of the corresponding segment along with some extra bits. Segments containing identical minterms are called ***repetitive segments***. The simplified form of

such a segment is simply one of the identical minterms. For example, the sequence ***P=1011011010110110*** is converted to the following expression.

$$P = b_7\overline{b_6}b_5b_4\overline{b_3}b_2b_1\overline{b_0} + b_7\overline{b_6}b_5b_4\overline{b_3}b_2b_1\overline{b_0}$$
$$= b_7\overline{b_6}b_5b_4\overline{b_3}b_2b_1\overline{b_0} \qquad \text{Equation (3)}$$

The simplified form of the above expression in the corresponding block will be 10110110. If a segment can be simplified to an implicant, the implicant will be the simplified form of the segment. Such a segment is called an ***implicant segment***. As an example, ***Q=10110110101101111011010110110100*** is converted to the following experiment.

$$Q = b_7\overline{b_6}b_5b_4\overline{b_3}b_2b_1\overline{b_0} + b_7\overline{b_6}b_5b_4\overline{b_3}b_2b_1b_0$$
$$+ b_7\overline{b_6}b_5b_4\overline{b_3}b_2\overline{b_1}b_0 + b_7\overline{b_6}b_5b_4\overline{b_3}b_2\overline{b_1}\overline{b_0}$$
$$= b_7\overline{b_6}b_5b_4\overline{b_3}b_2 \qquad \text{Equation (4)}$$

As equation (2) shows, two variables b_0 and b_1 have been discarded and the implicant $b_7\overline{b_6}b_5b_4\overline{b_3}b_2$ has been formed. The simplified form of the above experiment will be ***101101*** which will be stored in the corresponding block.

In the block construction phase, simplified forms of sequences are stored in blocks. We refer to these blocks as ***compressed blocks***. The structure of a compressed block is shown in figure 2.

The ***F0*** field is a one-bit flag which distinguishes implicant segments from repetitive segments. This flag will be 0 for blocks corresponding to implicant segments (***implicant blocks***) and 1 for blocks corresponding to repetitive segments (***repetitive blocks***).

The ***F1*** field shows the number of discarded variables for implicant segments. In other words, this field is equal to $\log_2 n$ for an implicant segment of n minterms. n can be equal to 1, 2, 4 or 8 ($n = 1$ represents a block consisting of a single minterm which cannot be part of any repetitive or implicant segment. Such blocks are called ***single blocks***). Thus, the possible values for this field are 0, 1, 2 and 3. This field is 2 bits long. ***F1*** is also equal to the logarithm of the number of identical minterms to the base 2 for repetitive segments.

F2 does not exist in repetitive blocks or single blocks in which ***F 1= 00***. This field shows the combination of discarded variables in implicant blocks. If ***F1*** is equal to ***01***, ***F2*** should indicate one of 8 variables ($\{b_i \mid i \in [0,7]\}$). Thus, the length of this field will be equal to $\lceil \log_2 8 \rceil = 3$ bits.

If ***F1*** is ***10***, ***F2*** should determine one of $\binom{8}{2} = 28$ combinations which may have been discarded.

F0	F1	F2	F3	F4
repetitive or implicant	number of excluded variables	excluded variables	order of minterms	simplified form

Figure 2. The structure of a compressed block.

In this case, the length of ***F2*** will be equal to $\lceil \log_2 28 \rceil = 5$. If ***F 1= 11***, ***F2*** will be equal to $\left\lceil \log_2 \binom{8}{3} \right\rceil = 6$ bits.

The field ***F3*** indicates the order of the simplified minterms in the original file. This field does not exist in repetitive blocks or single blocks. If ***F1=01***, F3 should determine one of $2! = 2$ possible orders. In this case, the length of this field will be equal to $\lceil \log_2 2 \rceil = 1$ bit. If ***F1=01***, ***F3*** should indicate one of $4! = 24$ possible orders for 4 minterms. In this case, the length of ***F3*** will be equal to $\lceil \log_2 24 \rceil = 5$ bits. If ***F1*** is 11, the length of ***F3*** will be equal to $\lceil \log_2 (8!) \rceil = 16$ bits.

The field ***F4*** in each block contains the simplified form of the corresponding segment. The length of this field will be equal to 8 bits for every repetitive block or every single block. This field will have 7 bits in 2-implicant blocks, 6 bits in 4-implicant blocks and 5bits in 8-implicant blocks.

According to the above discussions the compression algorithm will be as follows:

I=1
While Not EOF (***OriginalFile***) {
Allocate a new block
If minterm I is identical to minterms i+1 to i+7 then
{
Set ***F0=1***, ***F1=11***
Consider no ***F2*** and no ***F3***
Put the minterm I in ***F4*** in 8 bits
Store the constructed block in ***CompressedFile***
Set i=i+8
}
Else If minterm I is identical to minterms i+1 to i+3 then
{
Set ***F0=1***, ***F1=10***
Consider no ***F2*** and no ***F3***
Put the minterm I in ***F4***
Store the constructed block in ***CompressedFile***
Set i=i+4
}

Else If minterm I is equal to minterm i+1
{
Set ***F0=1***, ***F1=01***
Consider no ***F2*** and no ***F3***
Put the minterm I in ***F4*** in 8 bits
Store the constructed block in ***CompressedFile***
Set i=i+2
}
Else If minterm I can be disjointed to minterms i+1 to i+7
{
Set ***F0=0***, ***F1=11***
Put the combination of the discarded variables in the array ***C***

Find the ID of the combination using the ***CombinationToID*** algorithm and assign the ID to ***F2***
Put the minterms I to i+7 in the array ***nums*** with respect to their positions in ***OriginalFile***
Find the ID of the permutation using the ***permutation*** algorithm and assign the ID to ***F3*** in 16 bits
Assign the simplified form of the 8 minterms I to i+7 to ***F4*** in 5 bits
Store the constructed block in ***CompressedFile***
Set i=i+8
}
Else If minterm I can be disjointed to minterms i+1 to i+3
{
Set ***F0=0***, ***F1=10***
Put the combination of the discarded variables in the array ***C***
Find the ID of the combination using the ***CombinationToID*** algorithm and assign the ID to ***F2*** in 5 bits
Put the minterms I to i+3 in the array ***nums*** with respect to their positions in ***OriginalFile***
Find the ID of the permutation using the ***permutation*** algorithm and assign the ID to ***F3*** in 5 bits
Assign the simplified form of the 4 minterms I to i+3 to ***F4*** in 6 bits
Store the constructed block in ***CompressedFile***
Set i=i+4
}
Else If minterm I can be disjointed to minterm i+1
{
Set ***F0=0***, ***F1=01***
Put the j in 3 bits in ***F2*** in 3 bits if b_j has been discarded
Set ***F3=0*** if the smaller minterm proceeds the larger minterm in **OriginalFile**, Set **F3=1** otherwise
Assign the simplified form of the 8 minterms I to i+3 to ***F4*** in 7 bits
Store the constructed block in ***CompressedFile***
Set i=i+2
}
Else
{
Optionally Set ***F0=0***, ***F0=1***
Set ***F1=00***
Consider no ***F2*** and no ***F3***
Assign the minterm I to ***F4*** in 8 bits
Store the constructed block in ***CompressedFile***
Set i=i+1
}
}

The algorithm ***CombinationToID*** is a simple recursive algorithm that takes a combination of variables as the input and gives a numerical ID that uniquely represents the input combination. ***PermutationToID*** is also another simple algorithm that takes a permutation of numbers and gives a unique numerical ID for that permutation.

4. THE DECOMPRESSION ALGORITHM

The decompression algorithm makes use of ***F0***, ***F1***, ***F2*** and ***F3*** fields of each block in the compressed file in order to convert the ***F4*** field to the corresponding segment and store the segment in the decompressed file. This algorithm works as follows.

i=1
While not EOF (***CompressedFile***) {
Read bit i as ***F0*** and bits i+1 and i+2 as ***F2***
i=i+3
If ***F1=00*** then
{
Read the next 8 bits as **F4** and store it in ***DecompressedFile.***
i=i+8
}
If ***F0=1*** and ***F1=01*** then
{
Read the next 8 bits as ***F4*** and store it twice in ***DecompressedFile.***
i=i+8
}
If ***F0=1*** and ***F1=10*** then
{
Read the next 8 bits as ***F4*** and store it four times in ***DecompressedFile.***
i=i+8
}
If ***F0=1*** and ***F1=11*** then
{
Read the next 8 bits as ***F4*** and store it eight times in ***DecompressedFile***.
i=i+8
}
If F0=0 and F1=01 then
{
Read the next 3 bits as ***F2***, ***i=i+3***
Read the next bit as ***F3***, ***i=i+1***
Read the next 7 bits as ***F4***, ***i=i+7***
Put once 0 and once 1 in ***F4*** in the position shown by ***F2*** and construct two octets.
If ***F3=0*** first store the octet containing 0 in ***DecompressedFile*** and next store the octet containing 1.
Otherwise, first store the octet containing 1 in ***DecompressedFile*** and next store the octet containing 0.

If F0=0 and F1=10 then
{
Read the next 5 bits as ***F2***, ***i=i+5***
Read the next 5 bits as ***F3***, ***i=i+5***
Read the next 6 bits as ***F4***, ***i=i+6***
Pass ***F2*** as an integer number to the algorithm ***IDToCombination*** and determine the two discarded variables.
Put once 0 and once 1 for each of the two discarded variables in ***F4*** and construct four octets.
Sort the four octets in ascending order in the array ***C*** and pass ***C*** with ***F3*** (as an integer number) to the ***IDToPermutation*** algorithm to determine the order of the octets.
Store the octets in ***DecompressedFile*** in the determined order.
}
If F0=0 and F1=11 then
{
Read the next 6 bits as ***F2, i=i+6***
Read the next 16 bits as ***F3, i=i+16***
Read the next 5 bits as ***F4***, ***i=i+5***
Pass ***F2*** as an integer number to the algorithm ***IDToCombination*** and determine the three discarded variables.
Put once 0 and once 1 for each of the three discarded variables in ***F4*** and construct eight octets.
Sort the eight octets in ascending order in the array ***C*** and pass ***C*** with ***F3*** (as an integer number) to the ***IDToPermutation*** algorithm to determine the order of the octets.
Store the octets in ***DecompressedFile*** in the determined order.
}

The algorithm ***IDToCombination*** is the reverse of ***CombinationToID***. It is a simple recursive algorithm that takes the ID of a combination of variables and constructs the combination. ***IDToPermutaion*** is also the reverse of ***PermutatioToID***. It takes the ID of a permutation and constructs the permutation.

5. PERFORMANCE EVALUATIONS

The probability that 8 consecutive minterms in the original file are the same can be obtained from the following equation.

$$P_{8-rep} = \left(\frac{1}{256}\right)^7 \qquad \text{Equation(5)}$$

In this case, the block length in the compressed file is equal to 11. Thus, the compression factor in this case can be calculated as follows.

$$C_{8-rep} = \frac{11}{8 * 8} = \frac{11}{64} \qquad \text{Equation(6)}$$

P_{4-eq} , C_{4-eq} , P_{2-eq} and C_{2-eq} can be obtained from the following equations.

$$P_{4-eq} = \left(\frac{1}{256}\right)^3 \qquad \text{Equation(7)}$$

$$C_{4-eq} = \frac{11}{4 * 8} = \frac{11}{32} \qquad \text{Equation(8)}$$

$$P_{2-eq} = \frac{1}{256} \qquad \text{Equation(9)}$$

$$C_{2-eq} = \frac{11}{2 * 8} = \frac{11}{16} \qquad \text{Equation(10)}$$

If each minterm has n variables, the number of implicants constructed from 2^m minterms (considering the order of minterms) can be calculated from the following equation.

$$N_m^n = \binom{n}{m}.2^{n-m}.(2^m)! \qquad \text{Equation(11)}$$

In the above equation, $\binom{n}{m}$ shows that m variables should be discarded. 2^{n-m} shows that the rest of variables can each be 0 or 1. $(2^m)!$ demonstrates all possible orders of 2^m minterms. According to the above equation, the probability that a minterm in the original file can be disjointed to its 7 next minterms and construct an 8-implicant can be calculated as follows.

$$P_{8-imp} = \frac{\binom{8}{3}.2^5.(2^3)!}{(256)^8} \qquad \text{Equation(12)}$$

The block length in the compressed file is equal to 30 bits in this case. Thus the compression factor will be calculated as follows.

$$C_{8-imp} = \frac{30}{8 * 8} = \frac{30}{64} \qquad \text{Equation(13)}$$

P_{4-imp} , C_{4-imp} , P_{2-imp} and C_{2-imp} are obtained from the following equations.

$$P_{4-imp} = \frac{\binom{8}{2}.2^6.(2^2)!}{(256)^8} \qquad \text{Equation(14)}$$

$$C_{4-imp} = \frac{19}{4 * 8} = \frac{19}{32} \qquad \text{Equation(15)}$$

$$P_{2-imp} = \frac{\binom{8}{1}.2^7.(2^1)!}{(256)^8} = \qquad \text{Equation(16)}$$

$$C_{2-imp} = \frac{14}{2 * 8} = \frac{14}{16} \qquad \text{Equation(17)}$$

Now let us calculate the number of ways to divide a set of 8 consecutive 8-variable minterms into segments of 1, 2, 4 and 8 minterms. To do this, we should first solve the following equation.

$$m + 2n + 4p + 8q = 8$$

$$m, n, p, q \in [0,8] \qquad \text{Equation(18)}$$

In the above equation, m, n, p and q represent the numbers of 1-segments, 2-segments, 4-segments and 8-segments respectively. The above equation has 10 sets of answers. There are a number of permutations for each set of answers. Table 1 shows all each of the 10 sets of

answers with the number of permutations corresponding to each of the sets. The table also shows the probability that the 8 consecutive minterms is divided in each way.

Table 1.Sets of Answers and their Probabilities

Set of answers				Number of permutations	Probability
m	**n**	**p**	**q**		
0	**0**	**2**	**0**	$\frac{2!}{2!} = 1$	$\frac{1}{66}$
0	**0**	**0**	**1**	$\frac{1!}{1!} = 1$	$\frac{1}{66}$
0	**2**	**1**	**0**	$\frac{3!}{2!*1!} = 3$	$\frac{3}{66}$
0	**4**	**0**	**0**	$\frac{4!}{4!} = 1$	$\frac{1}{66}$
2	**3**	**0**	**0**	$\frac{5!}{3!*2!} = 10$	$\frac{10}{66}$
2	**1**	**1**	**0**	$\frac{4!}{2!*1!*1!} = 12$	$\frac{12}{66}$
4	**0**	**1**	**0**	$\frac{5!}{4!*1!} = 5$	$\frac{5}{66}$
4	**2**	**0**	**0**	$\frac{6!}{4!*2!} = 15$	$\frac{15}{66}$
6	**1**	**0**	**0**	$\frac{7!}{6!*1!} = 7$	$\frac{7}{66}$
8	**0**	**0**	**0**	$\frac{8!}{8!} = 1$	$\frac{1}{66}$

There are $2^8 = 256$ possible 1, 2, 4 or 8-repetitive segments. The total number of possible 2-repetitive segments is equal to $\binom{8}{1}.2^7.(2^1)! = 2048$. A 2-repetitive segment has compression factor of $\frac{11}{66}$. Every 2-implicant segment has a compression factor of $\frac{14}{66}$. Thus, the average compression factor of a 2-segment is calculated as follows.

$$C_2 = \frac{256 * \frac{11}{16} + 2048 * \frac{14}{16}}{2048 + 256} \approx 0.85 \qquad \text{Equation(19)}$$

We can calculate the compression factors of 4 and also 8-segments in a similar way.

$$C_4 = \frac{256 * \frac{11}{32} + \binom{8}{2}.2^6.(2^2)! * \frac{19}{32}}{256 + \binom{8}{3}.2^6.(2^2)!} \approx 0.60 \qquad \text{Equation(20)}$$

$$C_8 = \frac{256 * \frac{11}{64} + \binom{8}{3}.2^5.(2^3)! * \frac{30}{64}}{256 + \binom{8}{3}.2^5.(2^3)!} \approx 0.47 \qquad \text{Equation(21)}$$

The compression factor for a 1-segment is equal to $\frac{19}{66} \approx 1.19$ which is larger than unity. Now we can calculate the average compression factor for each of the 10 states in table 1. These factors will be equal to 0.6, 0.47, 0.66, 0.85, 0.93, 0.81, 0.89, 1.0, 1.10 and 1.19 respectively.

Table 2.Experimental Results

Color Image	**Original Size**	**Compressed Size**	**Compression Ratio**
Lena	6291456	4731162	25%
Baboon	5760000	5347825	7.2%
Pepper	6291456	4654743	26%
Gold hill	9953280	7355764	26%
Dreamnight	6291456	1157792	82%
Sailboat	6291456	5450349	13%
Splash	6291456	4222848	33%
Greyscale Image	**Original Size**	**Compressed Size**	**Compression Ratio**
Lena	2097152	1533440	27%
Baboon	1920000	1821017	5.2%
Pepper	2097152	1514595	28%
Gold hill	9953280	2381777	28%
Dreamnight	6291456	1100866	48%
Sailboat	2097152	1632913	22%
Splash	2097152	1336786	36%
		Average	**29%**

We can calculate the average compression factor of YALMIC algorithm by multiplying the compression factor of each state by its probability and adding the products to each other.
The sum of the mentioned products is $C_t \approx 0.8$. Thus, the compression factor of YALMIC is equal to 0.8 and its compression ratio will be equal to $CR_t \approx 1 - 0.8 = 0.2 = 20\%$.

An important point to consider here is that the average compression ratio can be greater than 20% for normal images. The reason is that we have made no extra assumptions regarding dependencies among the pixels or bytes of the image while in real images, there is often a lot of dependency between the pixels and bytes. The experimental results verify this point. These results are shown in table 2.
Table 2 consists of two parts. The upper part shows the results obtained from color images and the lower part shows the same results for gray scale images. Each row in this table contains the name, the original size, the compressed size and the compression ratio for one of the images which have been compressed using YALMIC. All the images are standard except for one of them which is named ***Dreamnight***. Dreamnight is a painted image drawn with the MS paint. This image is shown in figure 3.

Figure 3. The Dreamnight Test Image

TABLE 3. RESULTS REPORTED BY AUGUSTINE, ET AL. IN [28]

Image	Method	Compression Ratio
Girl	Logic coding	39.9
	PVRG-JPEG	30.7-38.1
Mandrill	Logic coding	11.6
	PVRG-JPEG	6.7-13.7
Boats	Logic coding	27.9
	PVRG-JPEG	24.9-33.1

TABLE 4. RESULTS REPORTED BY VILLARROYA, ET AL.IN [19]

Method	Average Compression Ratio
CF-AE	11.89
CF-OBDD	14.68
SF-OBDD	18.4

As shown in table 2, the compression ratio is larger than 25% for all images except for ***Baboon***. The average compression ratio is about 25% which is quite comparable to the results reported in previous works such as those reported by Augustine, et al. in [28], or those obtained by Villarroya, et al. in [19].

Table 3 shows the results obtained by Augustine, et al. [28]. As shown in this table, the average compression ratio of their method (called ***Logic Coding***) is almost 26.5. Table 4 shows the results reported in [7]. The latter results have been obtained through applying three different methods to CCITT fax images. These methods are called CF-AE, CF-OBDD and SF-OBDD. As demonstrated by table 4, the compression ratios of the used methods are 11.89, 14.68 and 18.4 respectively.

6. CONCLUSIONS AND FURTHER WORKS

In this paper an image compression method called YALMIC was introduced. This algorithm divides the image into segments of 1, 2, 4 or 8 octets of bits. The segments are then considered as logic expressions in the form of sum of products. YALMIC distinguishes two types of segments. The first type consists of duplicated octets and the second consists of octets which can construct prime implicants using the Quine–McCluskey minimization technique. The simplified forms of logic expressions are stored in the compressed file along with some extra bits which are required to fully regenerate the original segments. Analytical modelling shows that the average compression ratio of the algorithm will be greater than 20%. Experimental results show that the average compression ratio is almost 25%. The method proposed in this paper is lossless. It does not depend on Huffman coding, arithmetic coding, prediction, differencing, logic transforms or OBDDs. It has not been specifically designed for text or gray scale images and it does not treat blocks of data as truth tables. This work can be continued by enlarging the size of the segments, dividing the image into rectangular windows instead of linear segments or applying the algorithm to text, audio and other kinds of data.

REFERENCES

[1] Behrouz Zolfaghari, Hamed Sheidaeian, Saadat Pour Mozaffari, "M-YALMIC: An Approach to Further Compressing Images Using Logic function Minimization", In Proceedings of IEEE International Conference on Intelligent Network and Computing (ICINC 2010), Kuala Lumpur, Malaysia, November 2010.

[2] Hamed Sheidaeian, Behrouz Zolfaghari, Saadat Pour Mozaffari, "SPIDERS: Swift Prime Implicant Detection Using Exhaustive Rotation and Shift", In Proceedings of IEEE International Conference on Networking and Information Technology (ICNIT 2010), Manila, Philipine.

[3] Behrouz Zolfaghari, Hamed Sheidaeian, Saadat Pour Mozaffari, "YALMIC: Yet Another Logic Minimization Based Image Compressor", In Proceedings of IEEE International Conference on Computer Engineering and Technology, Chengdu, China, 16-18 April 2010.

[4] Narendra Kumar, Dr. Sachin Gupta, "Use of Local Minimization for Lossless Gray Image Compression", International Journal of Computer Science and Information Technologies, VOLUME 1, ISSUE 4, Pages 203-207, September 2010.

[5] E. El Qawasmeh E., P. Pichappan,A. Alfitiani, "Development and investigation of a new compression technique using Boolean minimizations", In Proceedings of Second International Conference on Applications of Digital Information and Web Technologies (ICADIWT '09), 2009.

[6] Behrouz Zolfaghari, Saadat Pour Mozafari, Haleh Karkhaneh, "QTOP: A Topological Approach to Minimizing Single-Output Logic Functions", In Proceedings of IEEE SCOReD2009 Conference, Malaysia 2009.

[7] Shoa A., Shirani S.,Optimized, "Atom Position and Coefficient Coding for Matching Pursuit-Based Image Compression", IEEE Transactions on Image Processing, Volume 18,Issue 12, pp. 2686-2694,Dec 2009.

[8] Dong-U Lee, Hyungjin Kim, Rahimi M., Estrin D., Villasenor J.D.,"Energy-Efficient Image Compression for Resource-Constrained Platforms", IEEE Transactions on Image Processing, Volume 18, Issue 9, pp. 2100-2113,Sept. 2009.

[9] Sanchez V., Nasiopoulos P., Abugharbieh R., "Novel Lossless fMRI Image Compression Based on Motion Compensation and Customized Entropy Coding", IEEE Transactions on Information Technology in Biomedicine,Volume 13, Issue 4,pp.645 - 655,July 2009.

[10] Ates H.F., Orchard M.T., "Spherical Coding Algorithm for Wavelet Image Compression", IEEE Transactions on Image Processing,Volume 18,Issue 5,pp. 1015 - 1024,May 2009.
[11] El Qawasmeh E.,Pichappan P.,Alfitiani A., "Development and investigation of a new compression technique using Boolean minimizations", In Proceedings of the Second International Conference on the Applications of Digital Information and Web Technologies (ICADIWT '09), pp. 505-511,4-6 Aug 2009.
[12] J. Yang, S. A. Savari, and O. Mencer. "Lossless Compression Using Two-Level and Multilevel Boolean Minimization", In Proc. of the 2006 IEEE Workshop on Signal Processing Systems, Banff, Canada, October 2006.
[13] B.J. Falkowski, "Compact representations of logic functions for lossless compression of grey-scale images", Journal of International Conference on Computers and Digital Techniques,pp. 221-230, 2004
[14] M.B.I. Reaz, F. Mohd-Yasin, M.S. Sulairnan, K.T. Tho, K.H. Yeow, "Hardware prototyping of boolean function classification schemes for lossless data compression", In Proceedings of Second IEEE International Conference on Computational Cybernetics (ICCC), pp. 47-51, 2004.
[15] B.J. Falkowski, "Compact representations of logic functions for lossless compression of grey-scale images", Journal of International Conference on Computers and Digital Techniques,pp. 221-230, 2004.
[16] Sos S. Agaian, Thomas A. Baran and Karen A. Panetta, "The Application of Logical Transforms to Lossless Image Compression Using Boolean Minimization", Proceedings, GSPx and International Signal Processing Conference, 13-31 March 2003.
[17] S.S. Agaian, T.A Baran, K.A. Panetta, "Transform-based image compression by noise reduction and spatial modification using Boolean minimization", In Proceedings of 2003 IEEE Workshop on Statistical Signal Processing,pp. 226-229, 2003.
[18] B.J., Falkowski, B.T. Olejnicka, "Multiple-valued and spectral approach to lossless compression of binary, grey scale and color biomedical images", In Proceedings of .32nd IEEE International Symposium on Multiple-Valued Logic, pp. 136–142, Boston, MA, May 2002.
[19] P. Mateu Villarroya, J. Prades-Nebot, "Sequential logic compression of images", In Proceedings of 2001 International Conference on Image Processing,pp 479-482, 2001.
[20] B.J. Falkowski, "Lossless compression of binary images using logic methods", In Proceedings of Europe Workshop on Computational Intelligence and Information Technologies, pp. 111–116, Nis, Yugoslavia, June 2001.
[21] B. J. Falkowski, L.S. Lim, "Gray scale image compression, based on multiple-valued input binary functions", Walsh and Reed- Muller spectra,In Proceedings of 30th IEEE International Symposium on Multiple-Valued Logic,pp. 279-284,Portland, Oregon, May 2000.
[22] P. Mateu-Villarroya. "Compressi d'imatges sense pkrdua d'informaci mitjancant tkcniques de minimitzaci6 Ibgica", Treball Fi de Camera, September 2000.
[23] D. Pramanik, J. Jacob, J Augustine, "Lossless compression of images using minterm coding, In Proceedings of International Conference on Information", Communications and Signal Processing,1997.
[24] R.P. Damadore, J. Agustine and J. Jacob, "Lossless and Lossy Image Compression Using Boolean Function Minimization", Sadhana Journal, pp. 55-64,Vol. 20, Part 1, Feb. 1996.
[25] Chaudhary A.K., Augustine J., Jacob J., "Lossless compression of images using logic minimization", In Proceedings of International Conference on Image Processing, pp. 77-80, Lausanne, Switzerland, Sep. 1996.
[26] J. Augustine,Wen Feng, J. Jacob, "Logic minimization based approach for compressing image data", In Proceedings of the 8th International Conference on VLSI Design, pp. 225-228, 1995.
[27] M. Starkey and R. E. Bryant, "Using Ordered Binary-Decision Diagrams for Compressing Images and Image sequences", Technical Report, CMU-CS-95- 105, Carnegie Mellon University, January 1995.
[28] Augustine, Jacob and Feng, Wen and Makur, Anamitra and Jacob, James, "Switching theoretic approach to image compression", journal of Signal Processing, Vol. 44, No 2., pp. 243-246, 1995.

[29] E.M. Sentovich, K.J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P.R. Stephan, R.K. Brayton, and A.L. Sangiovanni-Vincentelli, "SIS: A System for Sequential Circuit Synthesis", Memorandum No. UCB/ERL M92/41, Department of EECS. UC Berkeley, 1992.
[30] R. K. Brayton, G. D. Hachtel, C. T. McMullen, A. L. Sangiovanni-Vincentelli, "Logic Minimization Algorithms for VLSI Synthesis", Kluwer Academic Publishers, Boston, 1984.
[31] M. Kunt and 0. Johnsen, "Block Coding of Graphics: A Tutorial Review", In Proceedings of the IEEE, vol. 68, NO. 7, pp. 770-786, July 1980.

Authors

Behrouz Zolfaghari is a Ph.D. student in computer engineering at Amirkabir University of Technology (AUT), Tehran, Iran. He is also a faculty member of engineering department in Islamic Azad University, Garmsar Branch. He has published six journal papers and 16 conference papers by far. His research areas include image processing, computer architecture and computer networks.

Hamed Sheidaeian has M.S. degree in computer engineering from Sharif University of Technology and is a faculty member of engineering department in Islamic Azad University, Garmsar Branch. His research areas include image and video compression, 3D computer graphic visualization, data communication and computer networks.