SSRG International Journal of Electrical and Electronics Engineering
ISSN: 2348-8379/ https://doi.org/10.14445/23488379/IJEEE-V12I4P115

Volume 12 Issue 4, 210-221, April 2025

Original Article

Lossless Compression of Grayscale Images via Representation of Bit Planes as Minimized Boolean Functions

M. Swathi Pai[1], Shyam P. Joy[2], Jacob Augustine[3]

[1,3]*School of Computer Science and Engineering, Presidency University, Bengaluru, Karnataka, India.*
[1]*Department of Artificial Intelligence and Machine Learning,*
NITTE (Deemed to be University) NMAM Institute of Technology, Karnataka, India.
[2]*Department of Artificial Intelligence and Machine Learning, CMR Institute of Technology, Bengaluru, Karnataka, India.*

[1]*Corresponding Author : swathi.pai@nitte.edu.in*

Received: 11 February 2025 Revised: 13 March 2025 Accepted: 15 April 2025 Published: 29 April 2025

Abstract - *Electronic Design Automation (EDA) research has made great strides owing to the industrial growth in the VLSI domain. Boolean function minimization is an important area in this domain. Many engineering problems can be formulated as logic functions, and the tools available in this domain can be applied to solve them. Image compression is formulated as a logic minimization problem by converting blocks of the bit planes of the grayscale image into Boolean functions and representing them in minimal form. An encoding scheme called logic coding was developed by this approach, which extended the block coding scheme. In this paper, we provide a comprehensive survey of research work that happened in this direction. A Python implementation of the basic logic coding scheme is also presented. Experimental results reveal that there are many blocks in the bit planes that are not compressible by these techniques. This data motivates us to explore options not tried by other researchers, although it is not a very active area of research. Lossless compression of images is an important area for medical images, and there is scope for further research. Medical images are an area where lossless compression is mandated; hence, it is still a relevant problem to research. This exploration is done with the intention to enhance existing logic coding techniques by applying other Boolean function representation schemes and a combination of various techniques to achieve better compression. Also, in areas such as deep learning and Natural Language Processing where binary multi-dimensional space is used, Boolean function representation could open further possibilities.*

Keywords - *Logic minimization, Boolean functions, Lossless image compression, PyEDA, Block coding.*

1. Introduction

Compression of binary [1, 2], grayscale [3-5] and color images [6, 7] images have been the focus of researchers for decades. Large amounts of data are generated in the digital representation of images. For efficient transmission and storage of digital images, it is required to reduce or compress the image data [8]. Several techniques for the compression of image data were proposed, and consequently, many standards were established to achieve the goal of compression with interoperability [9-11].

Image data has considerable redundancy, owing to the fact that adjacent pixels are very likely to be similar. Various compression techniques proposed, attempt to capture this redundancy in some way to achieve reduction of data. While a particular technique does well on a certain class of images, it may not do well in another class of images. Images with smooth variations (pixel values change gradually) are compressed well by most of the techniques, whereas regions with rapid variations in pixel values are not handled easily. There is a need for compression techniques that do not incur loss of information, and those with Loss.

While the lossless techniques are mandatory in certain applications such as medical images, lossy compression techniques can be used in most generic applications. Lossy compression techniques achieve better compression as they allow information loss. Though a lot of research has happened in this area, new techniques have been proposed. Converting image pixels into logic functions and representing them in compact form was one of the new lossless compression approaches proposed [11] and further explored by many researchers. We believe most of the lossless compression algorithms capture mostly one-dimensional redundancy, while the logic minimization approach captures redundancy in two dimensions. As various methods available to represent logic

functions are not explored for lossless compression of images, we believe there is potential for further research. Exploration in this direction may open up avenues for applying switching theoretical techniques in other areas of image processing.

2. Bit Plane Encoding of Grayscale Image

Bit plane coding of grayscale images [6] involves separating the individual bits of pixel values into distinct binary images. Digital images are typically represented as a matrix of pixel values. For an 8-bit grayscale image, each pixel is represented by a number ranging from 0 to 255. Actually, each of these elements may be depicted as being in an individual 'bit plane.' For example, the Most Significant Bit plane (MSB) includes the most significant bit of each pixel, and the Least Significant Bit plane (LSB) includes the least significant bit of each pixel. In the 8-bit per pixel image, there are 8 different bit planes, each one of which forms a binary image. In these planes, a bit is set as 1 if the corresponding bit in the pixel's binary setting is 1 and 0 if this bit is 0.

The higher bit planes (closer to the MSB) contain more significant image information (such as contours and edges), while the lower bit planes (closer to the LSB) contain finer details. It reveals that the bit planes corresponding to the most significant bits contribute most of the information regarding an image. Lena is a widely used standard test image in image processing. It is an 8-bit grayscale image that can be decomposed into 8 distinct bit planes, as shown in Figure 1. The 7^{th} plane will show the most significant information, such as the overall shape and structure of the image, while the 0^{th} plane will have less critical information, mostly fine details. As you move from the MSB (7^{th} plane) to the LSB (0^{th} plane), the planes contain progressively less information relevant to human perception. In bit-plane encoding, the most significant bit planes can be compressed using lossless techniques, ensuring that no important visual information is lost.

The least significant bit planes may be compressed using lossy methods or even omitted to reduce the data size while maintaining acceptable image quality. Bit plane coding effectively leverages the fact that not all bits in an image are equally important for human perception. Separating the image into bit planes and selectively compressing them makes it possible to achieve efficient compression. *Lena*, with its diverse range of details and textures, serves as an excellent example to demonstrate the effectiveness of bit plane coding in preserving critical information while reducing overall data size. In general, these bit planes are compressed using Binary Image Compression Techniques.

3. Block Coding of Images and Related Work

The block coding of binary images was proposed by Kunt and Johnsen. In this approach, a two-level image is divided into blocks of size n x m. A prefix code was used to code the three types of blocks: all black, all-white, and mixed.

In the case of mixed blocks, the bits in the block are stored as such after the code. Figure 2 shows an example of generalized block coding proposed. Kunt and Johnsen also extended this approach for grayscale images by splitting images into bit-planes [6]

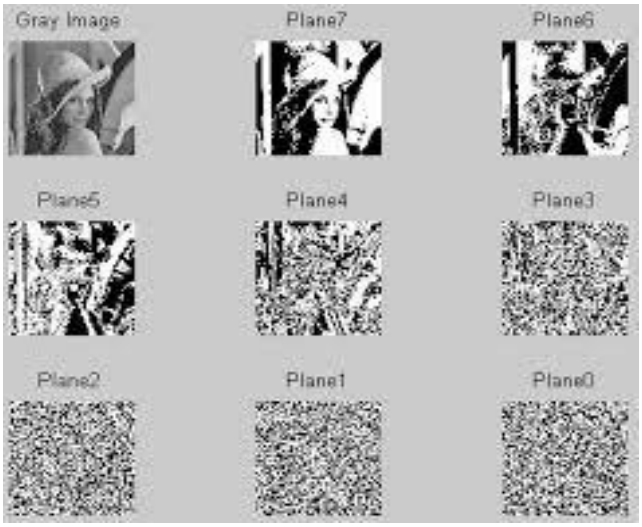


Fig. 1 8-Bit planes of lena

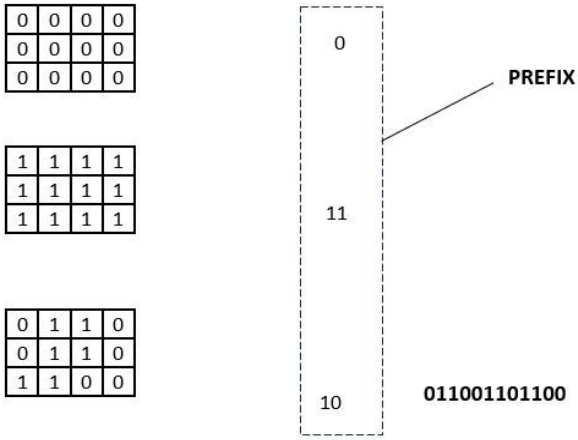


Fig. 2 Illustrating the principle of generalized block coding

Mohamed and Fahmy also suggested a binary image compression technique involving tiling up the image into mathematically efficient nonoverlapping rectangular zones. In this technique, both the x and y coordinates of the diagonally opposite corners of each rectangle are considered. This method's success mostly depends on how well the partitioning algorithm performs.

4. Compact Representation of Boolean (Switching) Functions

Switching functions or Boolean functions can be represented in many ways. Minimization of Boolean functions

is of great importance in VLSI design. Several minimization approaches were developed for the compact representation of Boolean functions [12]. Cube based approach was developed by Brayton et al. [13, 14] and implemented in the ESPRESSO minimizer primarily to handle Boolean function with a large number of variables.

In Figure 3, a simple Boolean function of three variables and its mapping onto vertices of the binary cube is shown where the arrows indicate the cyclic sequence in which the vertices are visited, starting from the origin (000 vertexes), for assigning the bits of the data stream (Boolean function value) to the vertices. The vertices mapped to a 1 are shown by the dark dots. Minimization is achieved if the geometric representation can be compactly described with less than 2^k bits. In this example, the 8-bit sequence can be represented by a single cube, xx0.

X1	X2	X3	
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

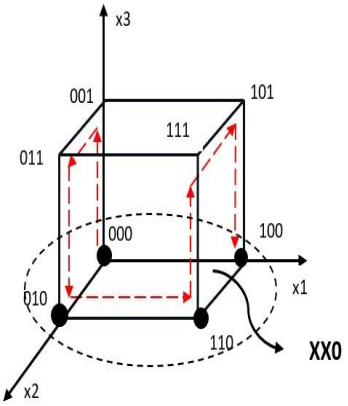


Fig. 3 Mapping a bit stream to a 3-D cube

Another well-known representation for switching functions is Binary Decision Diagrams [17]. BDD is an abbreviation for Binary Decision Diagram, also sometimes called a branching program, and it is a data structure representing Boolean functions. But at a higher level, they represent sets or relations in a compact form, which high-level BDDs signify.

Unlike a lot of other compression techniques, it is possible to perform operations on the compressed data using BDDs without the need for data to be uncompressed. A Boolean function in this format can be represented by a rooted-directed acyclic graph containing decision nodes and two terminal nodes.

These terminal nodes are usually numbered as 0, designating FALSE, and 1, designating TRUE. An important characteristic of a BDD is whether all the paths starting in the root node correspond to the same type of variable order. It is considered reduced if two optimization rules are applied. It is consists of two steps:

- Join all isomorphic subgraphs, and
- Delete the node with all of the children nodes are similar to each other.

When the general public uses the abbreviation BDD, it usually means a Reduced Ordered Binary Decision Diagram (ROBDD), where both ordering and reduction are reflected. An ROBDD is, however, canonical, which means that only one ROBDD exists for each function and order of variables. It is, therefore, most useful when applied to tasks such as FE and FT mapping. OBDD is a data structure similar to quadtree [19]. These data structures are graphs with nodes and leaves, which are used in the compact representation of Boolean functions. BDD has been extended for multiple valued logic functions also.

5. Logic Coding

Jacob et al. proposed logic coding of images by augmenting block coding of binary images with logic minimization. In this approach, the gray level image is divided into bit planes and logic coding is applied to bit planes. In this technique, picture information is represented as switching functions in minimized form. Though multiple possible representations of switching functions have been described in the previous section, here it has been investigated only the possibility of using minimized two-level cubical representation of switching functions for representing picture information. The technique has been named as Logic Coding. Data compression is achieved primarily through a logic minimization operation hence the name logic coding.

The steps of this compression scheme are illustrated in the flowchart presented in Figure 4. In this method, the image is segmented in its bit planes or binary images that are further divisible in smaller dimensions of blocks $n \times m$, where n and m are integer powers of 2. These blocks are categorized into three distinct types:

- All-black: blocks with only black pixels,
- All-white: blocks with only white pixels and
- Mixed: blocks with a combination of black and white pixels

Simple codes can be employed as codes for the first two types of blocks. In the case of mixed blocks, their bits are considered as the output of Boolean switching functions which can be in the form of a truth table. These functions are then optimized using the well-known cube-based two-level logic minimizer ESPRESSO [15]. The minimized cubes, which are, in fact, the implicants (product terms) of the switching functions, are coded with a code set that respects the prefix property in order to give the compressed data. If this method

does not produce compressed pixel data for a particular block, the block's pixels are saved without transformation. The presented technique of partitions and using smaller blocks for bit planes can be compared to the Block Coding method for graphics described in [12] in section 3.

In Block Coding, all-white and all-black blocks are termed 1-bit and 2-bit prefix code blocks. Small blocks of size $n \times$ are of $n \times m$ bits and are stored in direct memory with a 2-bit prefix code. This method uses a statistical sub-optimal coding technique based on a high probability of generating an all-black and an all-white block. Logic coding extends this by compressing mixed blocks even further with Boolean minimization; hence, it is more efficient than Block Coding.

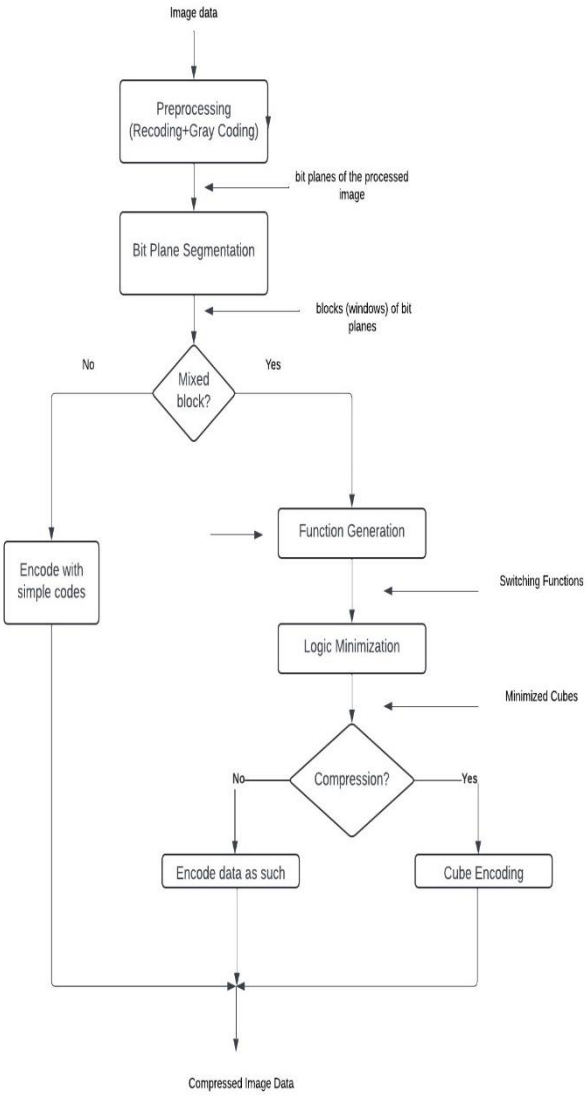


Fig. 4 Compression scheme

A possible interpretation of this technique may be given below. Generation of a switching function from a block of pixels of a bit plane can be viewed as the process of mapping the sequence of 2^k pixels of the block to the vertices of a hypercube of dimension k. This is the geometric representation of a switching function where the set of vertices mapped to 1 (black pixels) constitutes the *ON-set*, and the other set of vertices mapped to 0 (white pixels) constitute the *OFF-set* of the corresponding switching function.

5.1. Preprocessing the Image

The idea of Gray coding helps to reduce the transitions on the bit planes for the advantage of the lossless coder. Gray coding increases the probability of all-black and all-white blocks in the bit planes [12]. This is effective in compression by the experiment. In this experiment, monochrome images with 256 possible intensity values, each represented by 8 bits, are considered. However, it is important to know that not all 256 intensity values may not be in an image. To optimize compression, an additional preprocessing step can be introduced, essentially recoding the intensity values. In this step, the intensity values that show in the image are given successive integer values beginning from 0 so that the values are adjacent. This recoding process costs 256 additional bits to indicate which intensity values are active and which of them are inactive. When converted using dual port RAMs, these intensity values may be subjected to Gray coding for better compaction after recoding.

An example can be used to explain how this recording is done. Example: Consider an image represented by 3 bits/pixel whose pixels can have 8 possible gray values. Suppose only 4 gray levels appear on a particular image, say 0, 3, 4, 5. If we can map these gray levels to 0, 1, 2, and 3 and replace the image's pixel values with the new values, the recoded image is obtained. Only pixel values ranging from 0 to 3 are present on the recorded image, and hence, 2 bits/pixel is sufficient. In other words, the process of recoding has helped to reduce the number of bits/pixels from 3 to 2. The information regarding the gray levels present and absent can be indicated by 8 bits i.e., 10011100. The presence of a gray level is indicated by 1 and absence by 0.

5.2. Function Generation and Logic Coding of Mixed Type Blocks

In the switching-theoretic approach, there is a block of mixed type of size $n \times m$ is converted into a Boolean switching function with $\log_2(\mathrm{nm})$ variables. The bitstream of n x m pixels is considered as the output of the Boolean function. The dimensions n and m are selected as integer powers of 2 to be sure $\log_2(\mathrm{nm})$ it is an integer. The computation of the truth table for the switching function is done by placing the pixels in the block and using gray codes to represent the minterms of the function. This is done to ensure that physically neighboring pixels are associated with neighbouring minterms, helping in the function minimization

process. Clusters of 2^{α} logically adjacent minterms form α-cube therefore, reduces the complexity. The scanning is done in row-wise order, with the orientation of scanning being opposite for successive rows (as shown in Figure 5(a)) so that the pixel at the edge of each row is mapped onto the successive minterm in the following logic sequence. In Figure 5 there is an illustration showing how the process of converting a 4 × 8 block of 32 pixel arrangement with 32 binary values into a switching function of five variables. An example of this scheme is given in Figure 5(a) for a mixed-type block from a bit plane.

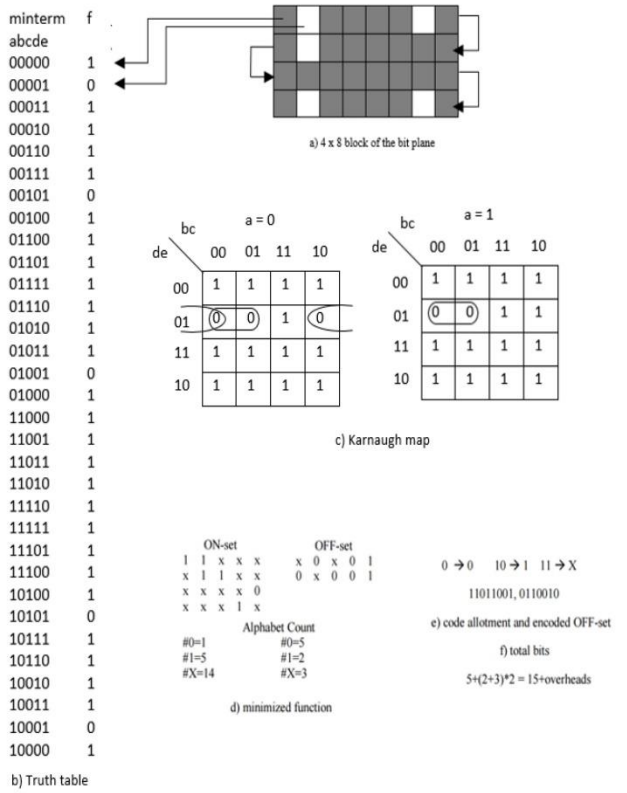


Fig. 5 Function generation and logic coding

As seen in the Figure 5(b), Gray code is applied to allocate minterms to the pixels to convert to Boolean switching function. Figure 5(c) shows the Karnaugh map representing the offset minterms, and the function is made simple by grouping these minterms into cubes. As shown in Figure 5(d), ON-set and OFF-set cubes minimized using ESPRESSO logic minimizer are presented along with the alphabet count. For a specific function corresponding to a pixel block, the count of cubes in the minimized ON-set and OFF-set is different.

The set with a smaller number of cubes is chosen to encode because ON-set and OFF-set represent the same Boolean function. Bits are included in the block header to specify which set; ON-set or OFF-set has been used for the compressed stream. This choice makes the representation of data more efficient. In this example, the OFF-set has a lesser number of cubes and, hence, is chosen for encoding. Figure 5(e) shows the particular allotment of prefix codes 0, 10, and 11 to the cube alphabets 0, 1, and X based on the frequency of occurrence of the symbols in cubes of function corresponding to the block. The number of bits required to represent the 32-pixel block is reduced from 32 bits to 15 bits plus overheads, using logic coding. The total bits required to represent the blocks are shown in Figure 5.

5.3. Format of the Compressed Image

The compressed image format has a global header that indicates the size of the image and, if there are any, a 256-bit overhead, which indicates how intensity values have been recoded during preprocessing. This overhead is missing if recoding is not done. Many blocks in a bit plane are often compressible through logic minimization or because all the bits are black or white; there are times, however, that a block cannot be compressed. This, actually, can cause a net increase for the entire bit plane when encoded. Such cases are more frequent in the planes corresponding to the two or three LSBs.

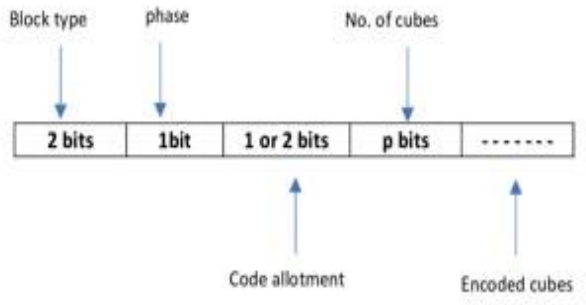


Fig. 6 Format of encoded block

In such cases, the entire bit plane is stored as it is (without compression) in the compressed file. A two-bit global header associated with each bit plane indicates whether it is in the original or compressed form. These two bits indicate whether a bit plane is all-black, all-white, logically compressible, or incompressible. For each logically compressible bit plane, the blocks are encoded in the specific format shown in Figure 6. The block header bits are interpreted as follows. The first two bits indicate the block type and are interpreted as follows:

- 00: all-black block
- 01: all-white block
- 10:compressible block (minimization yields compression)
- 11: incompressible block (minimization fails to compress the block)

For all-black and all-white blocks, only the first two bits are present. In the last case(type 11) of an incompressible

block, the *n* x *m* bits of the block are stored directly after the 2-bit code for the block type. If the block is compressible by logic minimization (type 10), then the next one-bit field, phase, indicates whether the minimized ON-set or OFF-set is selected, as indicated below:

- 0: ON-set
- 1: OFF-set

The code allotment field, 1 or 2 bits in length, fixes prefix codes to the cube alphabets 0, 1, and X. This allocation is dynamic, meaning the prefix code set is used for each block. {0,10,11}, where the single-bit code 0 is linked to the character most frequently used in string s. This approach improves the compression because it prioritises the most frequently occurring alphabet. These codes could be assigned in three ways depending on the degree of distinction and type of analysis, as described in Table 1. A special prefix code is used to show which of the three applicable arrangements relates to the block in question. Additionally, the p-bit field is caused to convey the number of cubes, and it defines the amount of minimized cubes in the ON-set or OFF-set of the block according to the phase bit field.

Table 1. Prefix code allotment to symbols

Allotment Indicator	Code Allotment		
	0	1	X
0	0	10	11
10	11	0	10
11	10	11	0

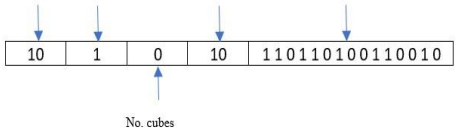
The p-value varies between 1 and 6 in the experiments, depending on the block size. The encoded cubes are placed after this. For example, for the block shown in Figure 5, the bits in the encoded block are as follows. Since the block is mixed type and is compressible, the first two-bit field will be 10. As the OFF-set is smaller, that would be chosen for encoding making the phase bit 1. Since the cube alphabet 0 has the maximum frequency in the OFF-set cubes, we will choose the first of the 3 possible code allotments for the cube alphabet. In the case of a 4 x 8 block, more than 3 cubes generally fail to produce any compression. Hence, p = 2 bits will be sufficient to indicate the number of cubes. Finally, the two OFF-set cubes, X0X01, and 0X001, are encoded. The compressed format, therefore, is 10,1,0,10,11011010, 0110010, and consists of 21 bits, whereas the original block has 32 bits.

5.4. Decompression Scheme

The bit planes are recovered from their compressed form by decoding the coded blocks of each bit plane. Recovery of the pixels is obvious in the case of all-black and all-white blocks and incompressible mixed-type blocks since they are stored without applying logic minimization.

For decoding logic-coded blocks of a bit plane, the values of the corresponding minimized function for all possible minterms of the block are reconstructed. This is done using the cube subsuming operation whereby the computed values are placed in respective pixels on the block. If a minterm is contained in any cube in the minimized ON-set (OFF-set) of the block, the corresponding pixel is given the value of 1 (0). Otherwise, the pixel is set to a value of 0 (1). It is also possible to view this decoding process as a reverse of the truth table to minimize the switching function.

Figure 7 illustrates the above recovery process that explains how, from a minimized cubical form, through the process of cube subsuming, an image block can be reconstructed. An example of a 4×8 block is given in Figure 7(a). The block type and phase bits indicate that the block is minimized and the OFF-set cubes have been encoded. From the knowledge of the code allotment to the cube alphabets and the number of cubes, the encoded OFF-set cubes can be easily recovered, as shown in Figure 7. To expand the function to its truth table form, the minterms of the function using a Gray counter are generated and check whether each minterm subsumes the minimized cubes of the OFF-set. If a minterm subsumes any of the cubes in the OFF-set the function takes the value 0 for the minterm and this value is assigned to the corresponding pixel in the 4x8 block. All minterms that do not subsume the OFF-set (and hence the corresponding pixels) evaluate to 1.



a) Bits of an example 4 X 8 block after compression

MINTERMS	SUBSUMES THE OFF – SET ?	PIXEL VALUE
00000	No	1
00001	Yes	0
00011	No	1
00010	No	1
-	-	-
-	-	-
-	-	-
10001	Yes	0
10000	No	1

Minimized OFF-set cubes = X0X01, 0X001

b) Recovering the block from compressed data

Fig. 7 Decoding of a logic-coded block

5.5. Bits Required to Code Logically Compressed Blocks

The bits required to represent the cubes in the case of an *n* x *m* pixel block can be computed as follows. Let the number of minimized cubes of the N variable function to be encoded be C ($0 \leq C \leq 2^{N-1}$) where N= $\log_2(nm)$. The symbols 0, 1,

and X of the cubes will be coded using the prefix codes 0, 10, and 11. The bits required to represent the cubes will vary depending on the frequency of occurrence of the cube alphabets. The symbol with maximum frequency is allotted the one-bit code and the other two-bit codes. Let n_1 be the count of the alphabet with maximum frequency and n_2 and n_3 the counts of the other two alphabets for the C cubes to be encoded. Now the number of bits BITCOUNT required to code the cubes can be written as,

$$\text{BITCOUNT} = n_1 + 2 \times (n_2 + n_3) \quad (1)$$

Based on this calculation, the upper and lower bounds of the bit requirement, excluding overheads, can be easily computed to represent an n x m block logically minimized to C cubes. It is obvious that $n_1 + n_2 + n_3 = \text{CN}$

The Upper Bound (UB) corresponds to the situation when all the symbols are equally likely or the frequency of occurrence of the symbols is the same i.e., CN/3. So the upper bound is,

$$UB = \frac{CN}{3} + \frac{2CN}{3} + \frac{2CN}{3} = 1.67\ CN\ bits$$

The possible Lower Bound (LB) occurs when only two of the 3 possible symbols occur in the C cubes, and each symbol can be coded with only 1 bit/symbol, i.e., $LB = \text{CN}$ bits.

6. Review of Related Literature

Agaian et al. [20] presented a lossless compression technique by extending the logic coding approach, drawing ideas from the lossy compression technique, where transform-based techniques are used. In their approach, the steps involved are pre-processing, image segmentation into blocks and bit planes, followed by transform and encoding. Preprocessing consists of linear prediction, histogram compression by zero probability removal and gray coding of intensities. After these steps, a transform-based technique developed by authors [3] is applied.

Yang et al. [16, 21] proposed a lossless image compression approach using multilevel logic synthesis instead of two-level minimization. Pre-processing and other entropy coders are used. Experimental results are provided on several test images. The authors concluded multilevel minimization do not offer compression improvements over two-level minimizers.

Starkey and Bryant [22] explored an approach of using OBDD to represent images with the intention of achieving compression. The number of nodes and bits required to store the ROBDD representation of the image is explained. A comparison of compression results achieved for test images using bintree and ROBDD is presented, and it is concluded that ROBDD gives better compression. Authors also present how a sequence of frames similar to the video can be represented efficiently by using ROBDD, indicating the possibility of achieving video compression.

Villarroya and Nebot [23] presented a lossless compression technique using an OBDD representation of the Boolean function generated from binary images. Their approach is based on the original idea of Starkey et al. [22], further enhanced with a more efficient encoding scheme for OBDD. Also, it is argued that the minimization of the Boolean function in OBDD representation is less compute-intensive compared to other methods of logic minimization presented earlier in this paper.

The authors also presented a method to use sequential logic instead of combinational logic to represent Boolean functions generated from binary images. OBDD representation of sequential functions is used. Experimental results using standard CCITT [24] fax images are also presented. P. Mateu et al. [25] proposed a method wherein the image is represented as a sequential logic function and minimized to obtain its compact representation. Boolean functions are represented using OBDD to reduce the cost of computation. OBDD of sequential function is obtained, reduced and then coded efficiently to reduce redundancy.

Luca et al. [28] presented data compression via logic synthesis. This approach synthesises the logic core to produce the given data string. Authors draw the idea from Kolgomorov complexity wherein the shortest program to generate the given bit string is found. This approach finds the logic function that generates the given binary data. Further, logic synthesis techniques are used to minimize this logic function and eliminate redundancy. Results are presented with experiments done on different benchmark data such as linear, linear plus noise, quadratic, and Random (XOR intensive).

Falkowski [26] presented an approach for binary image compression extending logic coding combined with Reed-Muller spectra. In his approach, the mixed blocks are converted to a Boolean function and a generalized approach that combines logic coding with other techniques of Reed-Muller weights-based patterns. The author combines logic coding (cubes), minterm coding, walsh, triangular, Reed-Muller (GPMPRM) transform and reference row technique in this approach. In this approach, the image blocks are divided into sizes varying from 8x8 to 4x4 based on criteria developed and different coding techniques are applied. The author implemented the technique in C language and tested it on standard CCITT images. Results are compared with techniques using OBDD1 [22] and OBDD2 [17]. Falkowsky's approach uses only switching theoretic techniques, whereas the OBDD2 approach uses arithmetic coding for the last phase.

Further, Falkowski presented a lossless compression technique for grayscale images using a compact representation of logic functions [27]. This is an extension of the author's technique for binary images [26]. In this approach, the grayscale image is subjected to prepossessing steps of file extraction, recoding of intensitiesand predictive coding.

The GAP predictor, used in the CALIC [28], is used for interior pixel prediction in an image. This predictor is adaptively based on the intensity gradients around the target pixels, thereby improving its predictive capability. After that, the grayscale image is segmented into bit planes [6] that reflect the intensity values of the image within the ranges of 0 and 255.

The author presents a set of coding schemes to encode the bit planes. As a first step, the transition count of pixels along the horizontal and vertical directions is measured. Depending on this count, the bit-plane is encoded as such, using minterm coding or dividing it into blocks of size 8x8 pixels and code. If segmented into blocks, a coding scheme is devised to have 5 types.

These are 8x8 blocks or further subdivided in the combination of 4x8 and 4x4 based on transition count and an empirical threshold. Variable length headers are also designed to represent the type. A block or sub-block may be coded using any of the following schemes.

They are minterm coding (if the number of 1s is very low), all white blocks indicated by headers, and coordinate coding. At a block size of 4x8, if appropriate based on the count of 1s, Multiple-Valued Logic (MVL) product term encoding. If the block is incompressible, it is further divided into 4x4, and any of the following approaches are used depending on the count of 1s. They are uniform block, minterm coding, pattern coding, MVL product term, reference row technique and incompressible.

In pattern matching, four possible patterns are indicated: Walsh, Reed-Muller, Triangular, and Special. The 4x4 block code can have direct match, inverse match, Direct match with one correction, and inverse match with one correction. The decompression technique is straightforward. Experimental results are presented for standard images and compared against well-known techniques such as Winzip, BPTC, LOCO, and S+P. Some of the images the proposed technique outperforms established techniques.

We feel this technique achieves good results, using too many combinations of methods and does not explore many of the available Boolean function manipulation techniques. We would like to explore purely switching theoretic techniques to replace pattern-matching techniques. Behrouz Z et al. [29] implemented image compression using logic minimization and compared results. Their system is called YALMIC. The picture data is changed with logical operations like AND, OR, and XOR. These transformations are used on pixel values or picture areas to encode data in a more compact and efficient fashion.

The converted picture data is expressed as Boolean expressions, which are built around logical actions between pixels or image components. Each phrase represents a certain feature of the picture data. The produced Boolean expressions are then analyzed using Boolean minimization techniques. These approaches seek to simplify expressions by minimizing duplication and complexity, while preserving the critical information necessary for accurate picture reconstruction.

The minimized Boolean statements encode the picture data. The technique compresses images while keeping fidelity by lowering expression size and complexity. The encoded representation is decoded during picture reconstruction by reversing the logical changes and rebuilding the original image data using the reduced Boolean expressions.

This approach guarantees that the rebuilt image closely resembles the original image while minimizing data loss. Overall, the paper’s compression technique uses digital logic concepts, namely logical transformations and Boolean reduction, to produce effective picture compression while maintaining image quality. The specifics of the algorithm may differ depending on the implementation details and optimizations used by the authors.

N Kumar and S Gupta [30] presented a lossless compression technique for Grayscale images using logic coding and further enhancements. The image is subjected to pre-processing of prediction, bit-plane splitting and partitioning into rectangular blocks. The blocks of pixels are converted to boolean functions and minimized using the Quine-McClusky technique.

The predictor employed in preprocessing is MED, which is used in LOCO-1 (Low Complexity Lossless Coder [31]). Results are presented using standard images and compared against UNIX compress (LZW) and JPEGLS [32] based on LOCO.

El Qawasmeh [33] describes creating and analysing a unique compression approach based on Boolean reduction. The suggested compression approach starts by expressing data as Boolean expressions, with each bit or set of bits acting as a variable in the equation. Boolean minimization methods, such as Quine-McCluskey or Espresso, are then used to simplify these expressions while retaining the original data semantics.

Experiments with diverse datasets, such as text, photos, and sensor data, are used to assess the compression technique’s performance. Compression ratios, compression speeds, and reconstruction accuracy are tested and compared to baseline

procedures. The experimental findings show that the suggested compression approach based on Boolean minimizations provides considerable compression ratios for many data types.

Furthermore, the compression process is competitively fast, making it suited for real-time operations. Reconstruction accuracy has also been demonstrated to be good, showing low data loss during compression and decompression. Compared to standard compression methods, the suggested methodology outperforms them regarding compression efficiency and computing complexity.

7. Experimental Results

Both mechanisms of the Logic Coding compression and decompression was done in Python 3.12 using LINUX operating system. The methods were tested with standard test images of Lena, Barbara, Bridge and Cameraman. ESPRESSO 2.1 was used, which is available in a Python package PyEDA (https://pyeda.readthedocs.io/en/latest/).

Table 2 below also presents the results of the compression experiments done with fixed block sizes. The table also compares Logic Coding and JPEG's lossless mode in terms of performance and advantages of the proposed method. Compression Ratio is calculated using the formula:

$$\text{Compression Ratio} = \frac{\text{Size of an original image}}{\text{Size of compressed image}}$$

For comparison, the PVRG-JPEG Codec 1.1, developed by the Portable Video Research Group at Stanford University, was utilized (available at havefun.stanford.edu:/pub/jpeg/JPEGv1.2.tar.Z). JPEG was run in its lossless mode, with seven possible predictor types being explored. Figures 3 and 4 below depict the minimum and maximum attainable vertical compression ratios during the above tests in Table 2.

The results obtained by Logic Coding are comparable to JPEG lossless; however, it may be pointed out that JPEG always employs a decorrelation scheme based on DPCM on the original image (one of the 7-predictors is used), whereas this scheme does not use any decorrelation. Many techniques in the switching theoretic approach (BDD, handling inclined edges, etc.) are yet to be tried.

The motivation for this experiment is to lay the foundation for developing a compression technique using only switching theoretic techniques and build further from here by deploying new approaches for handling incompressible blocks. Also, increasing the block size by joining blocks after compression is another approach we are exploring now.

Table 2. Summary of compression results

Image	Lena	Barbara	Bridge	Cameraman
Logic Coding				
Compression Ratio	17.4	13.2	19.4	19
Compression Time (s)	139.6	544.7	537.5	132.9
Decompression Time (s)	0.1	0.36	0.53	0.10
Block Coding				
Compression Ratio	12.6	1.8	13.8	17.5
Compression Time (s)	4.6	5.0	4.5	4.7
Decompression Time (s)	2.6	2.7	2.6	2.5
PVRG-JPEG				
Compression Ratio (%)	22.7-29.4	6.7-13.7	24.9-32.9	30.7-38.1
Compression Time (s)	0.4	0.4	0.4	0.4
Decompression Time (s)	0.3	0.3	0.3	0.3
Gray Levels	230	216	208	120

The time, which is reported as the compression time, contains the preprocessing phase as well. Regarding the influence of the proposed logic coding method on CPU time consumption for compression, it should be stated that the corresponding indices are comparatively high presently. This is more so because ESPRESSO in PyEDA is used on a standalone basis, thereby entailing a lot of overhead in the form of communication and file management.

An order-of-magnitude improvement in speed can be expected by integrating ESPRESSO with our code. It is also possible to create a logic minimizer optimized for this purpose. For the time being, our focus is to create a compression scheme based solely on switching theoretical techniques. Decompression is faster than compression because the cube-subsuming operation that needs to be performed is not complex.

This variation in processing time is evident from the data shown in Table 2. However, it must be pointed out that speed was not our primary concern, and it is possible to accelerate the compression and decompression steps significantly with a more efficient and fine-tuned implementation.

Table 3. Compression ratios and statistics for bit planes

Row	Bit Plane	s7	s6	s5	s4	s3	s2	s1	s0
1	Block size (n x m)	8 x 4	8 x 4	8 x 4	8 x 4	8 x 4	8 x 4	8x4	8x4
2	All-black blocks (%)	53.1	23.1	14.5	6.6	0.5	0.0	0.0	0.0
3	All-white blocks (%)	17.3	21.6	8.1	4.0	0.4	0.0	0.0	0.0
4	Logically compressible (%)	26.6	43.1	49.5	43.8	27.1	5.1	0.8	0.7
5	Logically incompressible (%)	2.8	12.0	27.7	45.4	71.8	94.8	99.1	99.2
6	Compressed to 1 cube (%)	8.4	10.0	10.4	6.64	2.3	0.04	0.0	0.0
7	Compressed to 2 cubes (%)	8.3	11.8	13.2	9.2	4.4	0.2	0.0	0.0
8	Compressed to 3 cubes (%)	6.0	11.1	13.0	12.5	7.5	1.0	0.0	0.1
9	ON-set encoded (%)	16.4	24.2	30.7	20.7	14.4	2.6	0.3	0.3
10	OFF-set encoded (%)	10.2	19.0	18.8	23.1	12.7	2.5	0.5	0.4
11	1 isolated minterm (%)	2.9	2.0	3.2	2.0	0.6	0.0	0.0	0.0
12	2 isolated minterms (%)	2.0	2.3	2.5	0.7	0.6	0.0	0.0	0.0
13	3 isolated minterms (%)	1.3	1.9	3.3	1.3	0.8	0.0	0.0	0.0
14	Total compression ratio for the bit plane (%)	69.4	45.3	20.1	2.3	-5.8	-6.3	-	-
15	Compression ratio for block coding with same block size (%)	58.9	32.5	9.6	-3.3	-6.3	-6.3	-	-
16	Bit plane compressible by logic coding?	YES	YES	NO	NO	NO	NO	NO	NO
Overall compression ratio for logic coding (%) (assuming expanding bit plans are stored as such) 17.1									

Table 3 gives various statistics on the test images. The statistics for the no pre-processing cases (binary coding) for the image Lena are given. No data is provided for the two or more least significant bit planes of each image, which are neither compressible by logic coding nor block coding. The second row gives the percentage of blocks (out of a total 65536 / (n X m)) belonging to the all-black and all-white category in each bit plane.

Kunt's block coding scheme exploits this component. The percentage of all-black and all-white blocks compressible and incompressible by logic coding are given in rows 2 and 3, respectively. The compression ratio computed for logically compressible blocks (excluding all black and white blocks) for each bit plane is given in row 4 to indicate the contribution of logic coding alone.

This is computed using the total number of bits for such types of blocks in uncompressed form and compressed form. The existence of a large percentage of logically incompressible blocks, especially on the lower bit planes, suggests the possibility of compressing them using other switching theoretic techniques. Statistics of the blocks compressed to 1, 2, and 3 cubes are given in rows 7, 9, and 11, respectively. Though in the case of block sizes of 8 x 4 pixels, more than 3 cubes in minimized form will not lead to compression, the percentages of such types of blocks are not given in the tables.

Logic coding is more successful if the proportion of blocks compressed to one cube constitutes a larger portion due to the increased bit saving in such cases. There are cases where the blocks contain only a few logically isolated pixels (black or white). Logic minimization has no effect in the case of such blocks as the isolated pixels correspond to minterms, which cannot be combined into larger cubes. The percentage of such types of blocks (which are minimized to 1, 2, or 3 isolated minterms) are also shown in the tables (rows 13, 14, and l5). Simpler cube encoding of these cases is possible as the minimized cubes consists only of minterms which have only two symbols, 0 and 1. A modification in the coding scheme to take advantage of this factor is considered. When the percentage of such types of blocks is significant, special attention is required to avoid wasting bits.

The average number of cubes per logically compressible block indicates the nature of compressible blocks. This count increases as the compressibility of the bit planes through logic coding decreases. The exact measure of the contribution of logic coding using a two-level minimization approach is indicated by the portion of the blocks that are logically compressible, excluding the blocks with isolated minterms and the corresponding compression ratio obtained. As logic minimization in such cases does not contribute beyond what is possible by coding the coordinates of the pixels, this differential measure helps to clearly establish the actual contribution of logic minimization.

It may be pointed out that in computing the percentage of blocks compressed to 1,2 or 3 cubes, the blocks that reduce to 1,2 or 3 isolated minterms were included already. Hence, the sum of the percentage values of rows 7,8 and 9 will correspond to the percentage of mixed logically compressible blocks given in row 4 of Table 3. For example, in Table 3 for bit plane s5, the percentage of logically compressible blocks, excluding the case of blocks with isolated minterms, is 49.56. Row 16 in the table indicates whether a given bit plane is compressible by logic coding or not. The incompressible bit planes are stored as they are since logic coding of these result in expansions. Two bits in the global header are used to indicate whether an entire bit plane is all-white, all-black, logically incompressible or logically compressible. Among the logically compressible blocks the percentage of blocks where ON-set/OFF-set is chosen for coding is also indicated in the Tables (rows 9 and 10). The fact that these two cases are more or less equally likely justifies the allotment of 1 bit in the header for distinguishing the phase.

For example, in Table 3, for the bit plane s4, the ON-set is chosen for 20.70 blocks, whereas for the OFF-set, 23.14 of blocks are encoded. Net compression achieved on each bit plane by logic coding and Kunt's block coding scheme are shown at the bottom of the tables (rows 14 and 15). It can be noted that logic coding always achieves a better compression ratio than block coding, giving an experimental justification for the intuitively obvious assumption made earlier in this section. In the last row of the tables, the net compression achieved on the entire image for each case is reported.

8. Conclusion

In this work, we investigated the research trail related to lossless compression of images using switching function representation and manipulation. Lossless compression of images is a fundamental problem in information theory, and scope for new techniques exists. Though it is not a very active research area, some researchers have always been fascinated by the difficult task of achieving better compression.

Research track related to representing images as switching functions and obtaining their compact representation to achieve data compression is explored in this paper. The evolution of this track of research started from block coding through logic coding to the usage of various logic synthesis techniques were surveyed. Experiments done by various researchers over a decade were summarized to find gaps in this research direction for further exploration.

We have implemented the basic logic coding technique in Python using the PyEDA library which supports ESPRESSO logic minimizer. Our investigation indicates that several options are still unexplored. For example, there could be a possibility to use a combination of techniques like cubes and BDD to handle incompressible blocks. Also, regarding the LSB bit-planes, it may be possible to use XOR function representation. Medical images require lossless compression, and the need to improve compression to be achievable is still an open problem that has not been fully solved. In this paper, we laid the foundation for further enhancement of *logic coding* to achieve better compression for medical images. There are also many image classes like barcodes and QR codes where the image by nature is bi-level, and Boolean function techniques could be handy. Deep Learning and NLP are areas where multi-dimensional space is used in embedding. Wherever data is binary in nature, the Boolean function techniques may be applicable for pursuing another branch of research direction.

References

[1] Eugene Ageenko, and Pasi Fränti, "Lossless Compression of Large Binary Images in Digital Spatial Libraries," *Computers and Graphics*, vol. 24, no. 1, pp. 91-98, 2000. [CrossRef] [Google Scholar] [Publisher Link]

[2] Pasi Fränti, and Olli Nevalainen, "Compression of Binary Images by Composite Methods Based on Block Coding," *Journal of Visual Communication and Image Representation*, vol. 6, no. 4, pp. 366-377, 1995. [CrossRef] [Google Scholar] [Publisher Link]

[3] David A. Clunie, "Lossless Compression of Grayscale Medical Images: Effectiveness of Traditional and State-of-the-Art Approaches," *Medical Imaging 2000: PACS Design and Evaluation: Engineering and Clinical Issues*, vol. 3980, pp. 74-84, 2000. [CrossRef] [Google Scholar] [Publisher Link]

[4] Khalid Sayood, *Introduction to Data Compression*, 5th ed., Morgan Kaufmann, 2017. [Google Scholar] [Publisher Link]

[5] M.J. Weinberger, G. Seroussi, and G. Sapiro, "From Logoi to the JPEG-LS Standard," *Proceedings International Conference on Image Processing (Cat. 99CH36348)*, Kobe, Japan, 1999. [CrossRef] [Google Scholar] [Publisher Link]

[6] Rafael C. Gonzalez, *Digital Image Processing*, Pearson Education India, 2nd ed., 2009. [Google Scholar]

[7] B.J. Falkowski, and Lip-San Lim, "Gray Scale Image Compression Based on Multiple-Valued Input Binary Functions, Walsh and Reed-Muller Spectra," *Proceedings 30th IEEE International Symposium on Multiple-Valued Logic*, Portland, USA, pp. 279-284, 2000. [CrossRef] [Google Scholar] [Publisher Link]

[8] Majid Rabbani, and Paul W. Jones, *Digital Image Compression Techniques*, SPIE Press, vol. 7, 1991. [CrossRef] [Google Scholar] [Publisher Link]

[9] Shahriar Akramullah, *Digital Video Concepts, Methods, and Metrics: Quality, Compression, Performance, and Power Trade-off Analysis*, 1st ed., Apress Berkeley, CA, 2014. [CrossRef] [Google Scholar] [Publisher Link]

[10] J. Mitchell, "Digital Compression and Coding of Continuous Tone Still Images: Requirements and Guidelines," *ITU-T Recommendation*, 1992. [Google Scholar] [Publisher Link]

[11] Abd Abrahim Mosslah, Reyadh Hazim Mahdi, and Hind Khalil Abrahem, "Efficient JPEG-LS for Lossless Compression of Rib Cage with Visual Quality Preservation," *Proceedings of the World Congress on Engineering, IAENG*, London, U.K, 2021. [Google Scholar] [Publisher Link]

[12] Murat Kunt, and O. Johnsen, "Block Coding of Graphics: A Tutorial Review," *Proceedings of the IEEE*, vol. 68, no. 7, pp. 770-786, 1980. [CrossRef] [Google Scholar] [Publisher Link]

[13] S.A. Mohamed, and M.M. Fahmy, "Binary Image Compression using Efficient Partitioning into Rectangular Regions," *IEEE Transactions on Communications*, vol. 43, no. 5, pp. 1888- 1893, 1995. [CrossRef] [Google Scholar] [Publisher Link]

[14] Nripendra N. Biswas, *Logic Design Theory*, Prentice-Hall, Inc, 1993. [Google Scholar] [Publisher Link]

[15] [17] Robert K. Brayton et al., *Logic Minimization Algorithms for VLSI Synthesis*, 1st ed., Springer, New York, 1984. [CrossRef] [Google Scholar] [Publisher Link]

[16] Robert K. Brayton et al., "MIS: A Multiple-Level Logic Optimization System," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 6, no. 6, pp. 1062-1081, 1987. [CrossRef] [Google Scholar] [Publisher Link]

[17] Randal E. Bryant, "Symbolic Boolean Manipulation with Ordered Binary-Decision Diagrams," *ACM Computing Surveys*, vol. 24, no. 3, pp. 293-318, 1992. [CrossRef] [Google Scholar] [Publisher Link]

[18] Heh-Tyan Liaw, and Chen-Shang Lin, "On the OBDD-Representation of General Boolean Functions," *IEEE Transactions on Computers*, vol. 41, no. 6, pp. 661-664, 1992. [CrossRef] [Google Scholar] [Publisher Link]

[19] Hanan Samet, "Data Structures for Quadtree Approximation and Compression," *Communications of the ACM*, vol. 28, no. 9, pp. 973-993, 1985. [CrossRef] [Google Scholar] [Publisher Link]

[20] S. Agaian, T. Baran, and K. Panetta, "The Application of Logical Transforms to Lossless Image Compression using Boolean Minimization," *Proceedings, GSPx and International Signal Processing Conference*, pp. 13-31, 2003. [Google Scholar] [Publisher Link]

[21] Jeehong Yang, Serap A. Savari, and Oskar Mencercv, "Lossless Compression using Two-Level and Multilevel Boolean Minimization," *IEEE Workshop on Signal Processing Systems Design and Implementation*, Banff, AB, Canada, pp. 148-152, 2006. [CrossRef] [Google Scholar] [Publisher Link]

[22] Mike Starkey, Randy Bryant, and Y. Bryant, "*Using Ordered Binary Decision Diagrams for Compressing Images and Image Sequences*," Carnegie-Mellon University, Department of Computer Science, 1995. [Google Scholar] [Publisher Link]

[23] P. Mateu-Villarroya, and J. Prades-Nebot, "Lossless Image Compression using Ordered Binary-Decision Diagrams," *Electronics Letters*, vol. 37, no. 3, 2001. [CrossRef] [Google Scholar] [Publisher Link]

[24] P. Mateu-Villarroya, and J. Prades-Nebot, "Sequential Logic Compression of Images," *Proceedings International Conference on Image Processing (Cat. No. 01CH37205)*, Thessaloniki, Greece, 2001. [CrossRef] [Google Scholar] [Publisher Link]

[25] Luca Amarú et al., "Data Compression via Logic Synthesis," *19th Asia and South Pacific Design Automation Conference*, Singapore, vol. 207, pp. 628-633, 2014. [CrossRef] [Google Scholar] [Publisher Link]

[26] Bogdan J. Falkowski, "Lossless Binary Image Compression using Logic Functions and Spectra," *Computers and Electrical Engineering*, vol. 30, no. 1, pp. 17-43, 2004. [CrossRef] [Google Scholar] [Publisher Link]

[27] Bogdan J. Falkowski, "Compact Representations of Logic Functions for Lossless Compression of Grey-Scale Images," *IEEE Proceedings Computers and Digital Techniques*, vol. 151, no. 3, pp. 221- 230, 2004. [CrossRef] [Google Scholar] [Publisher Link]

[28] X. Wu, and Nasir Memon, "Context-Based, Adaptive, Lossless Image Coding," *IEEE Transactions on Communications*, vol. 45, no. 4, pp. 437-444, 1997. [CrossRef] [Google Scholar] [Publisher Link]

[29] Behrouz Zolfaghari, Hamed Sheidaeian, and Saadat Pour Mozaffari, "Yalmic: Yet another Logic Minimization Based Image Compressor," *2nd International Conference on Computer Engineering and Technology*, Chengdu, China, vol. 44, pp. 90-95, 2010. [CrossRef] [Google Scholar] [Publisher Link]

[30] Narendra Kumar, and Sachin Gupta, "Use of Local Minimization for Lossless Gray Image Compression," *International Journal of Computer Science and Information Technologies*, vol. 1, no. 4, pp. 203-207, 2010. [Google Scholar] [Publisher Link]

[31] Nasir Memon, and X. Wu, "Recent Developments in Context-Based Predictive Techniques for Lossless Image Compression," *The Computer Journal*, vol. 40, no. 2 and 3, pp. 127-136, 1997. [CrossRef] [Google Scholar] [Publisher Link]

[32] JPE Group, Jpeg Standard for Image Compression, 2024. [Online]. Available: https://jpeg.org/jpeg/

[33] Eyas El Qawasmeh, Pit Pichappan, and Arif Alfitiani, "Development and Investigation of a New Compression Technique using Boolean Minimizations," *2nd International Conference on the Applications of Digital Information and Web Technologies*, London, UK, pp. 505-511, 2009. [CrossRef] [Google Scholar] [Publisher Link]