The application of logical transforms to lossless image compression using Boolean minimization

Sos S. Agaian
University of Texas at San Antonio
Electrical Engineering
6900 Noah Loop
San Antonio, TX 78249-0665
(210) 458-5939
sagaian@utsa.edu

Thomas A. Baran
Tufts University Electrical and Computer Engineering
161 College Avenue
Medford, MA 02155
(617) 627-5976
tbaran@eecs.tufts.edu

Karen A. Panetta
Tufts University Electrical and Computer Engineering
161 College Avenue
Medford, MA 02155
(617) 627-5976
karen@eecs.tufts.edu

ABSTRACT

A rapidly advancing field, image compression has seen many recent developments. A number of well-known compression algorithms dominate the world of lossy compression, but in the realm of lossless compression, fewer techniques have been explored. The most successful lossy compression methods are based on transform methodologies, commonly involving the KLT, cosine, and wavelet transforms. However, well-known lossless compression techniques such as Huffman coding, arithmetic coding, the Lempel-Ziv algorithm (LZW), and run-length coding do not utilize transforms. *In this paper, we investigate the application of transform-based algorithms to the domain of lossless compression.*

As a first step in this research, we have applied *logical transforms* to lossless compression. Logical transforms are matrix-based transforms that operate on vectors whose elements represent either a binary '0' or '1'. The central idea of the logical transform method is, indeed, similar to that of the lossy case: the image at hand is divided into blocks, and each block is transformed. However, logical transforms are applied to blocks of binary data in *bit planes* of the grayscale image to be compressed. Using a sequence of logical transforms designed to perform the process of *Boolean minimization*, we were able to successfully compress images and achieve ratios comparable to that of alternative lossless techniques.

Existing techniques for image compression using the minimization of Boolean "switching" functions rely chiefly on time-inefficient algorithms for minimization using Karnaugh-maps (K-maps) or other approaches. Using K-maps presents several drawbacks. First, the complexity of minimizing a given Boolean function increases rapidly as the size of its output truth table increases. Minimizing a 6-variable (length 64) Boolean function, for example, is a relatively complex challenge compared to the problem of minimizing a 4- or 5-variable function (lengths 16 and 32, respectively) when K-maps are used. Second, there does not exist a fast implementation for the K-map approach, so the compression process is rather slow.

In this paper, we propose a new technique for the lossless compression of images using logical transforms to achieve Boolean minimization. This new technique provides several important advantages over other Boolean minimization-based compression methods: a) here, a Boolean function of an arbitrary number of input variables can be minimized, b) fast implementations exist, c) attaining compression ratios greater than those of existing lossless methods is possible, d) no multiplications are needed, e) hardware implementations are practical, f) the technique can be extended to compress data sets other than images.

The compression methods presented here clearly demonstrate the feasibility of lossless coding schemes based on logical transforms. We expect to find further optimizations and soon arrive at a definitive standard for a compression technique using logical transforms.

Keywords

Boolean minimization, lossless compression, image, transform, prediction, fast transform

1. INTRODUCTION

1.1 Existing Lossless Techniques

Many existing methods for lossless compression do not, as explained above, make use of transform-based approaches. Because our work is an extension of an existing Boolean compression method [1, 2], the general idea of Boolean image compression will be outlined.

In general, Boolean compression consists of taking a binary sequence of data of length $N=2^n$, assigning it to the output of a truth table of some Boolean function of *n* variables, and finding the minterms of this function. Often, representing a binary sequence in terms of its minterms requires fewer bits than simply transmitting the sequence. In particular, functions with certain repeating patterns can often be represented by one or two minterms. As we will demonstrate below, images often contain such data, so Boolean minimization is a beneficial technique.

1.2 Our Transform-Based Boolean Minimization Technique

Many lossy compression techniques are based upon 3 steps:

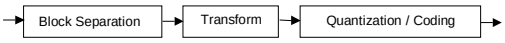

Figure 1. Typical lossy compression scheme.

Our lossless compression technique is loosely inspired by these lossy techniques and operates in 3 similar steps:

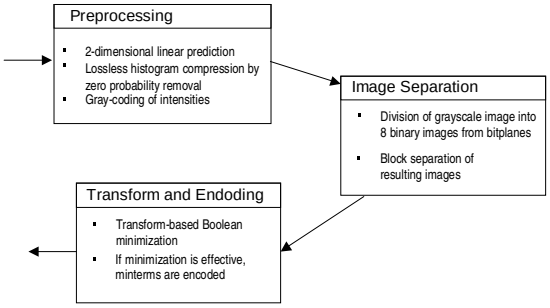

Figure 2. Our lossless scheme.

1.3 The Transform

Performing the Boolean compression step using a transform-based method allows for a completely generalized algorithm for minimizing Boolean functions of an arbitrary number *n* of input variables, unlike Karnaugh-map techniques which work well only for Boolean functions of a small number of input variables.

The following transform-based algorithm, presented in [3], is used to perform the Boolean minimization step in the compression algorithm.

Let us define the transpose of the interval splicing matrix as

$$\mathbf{A}_n^T = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}^{\otimes n}, \qquad (1)$$

the absorbing matrix as

$$\mathbf{B}_n = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}^{\otimes n}, \qquad (2)$$

and the Kronecker $\boldsymbol{\delta}_0$ as a function which takes a vector, turns all of its 0 elements to 1's and all others to 0's, then returns this vector [3].

The first step in the minimization of a Boolean function *f* whose truth table output vector is **f** is calculating

$$\mathbf{y} = \mathbf{B}_n\{\boldsymbol{\delta}_0[\mathbf{A}_n^T \boldsymbol{\delta}_n(\mathbf{f})]\} \text{ [3].} \qquad (3)$$

The elements of **y** that equal 1 represent terms in a Boolean algebraic expression of the original function. The particular minterm represented by a given 1-valued element of **y** is obtained by taking the *base-3* expansion of the index of this element. (Note that our output vector **y** is longer than the input vector **f** because of the use of the asymmetric matrix $\mathbf{A}_n^T$.) For a digit d_i of the base-3 index of an element of **y** whose value equals 1,

$d_i = 0$ $\quad \bar{x}_i$

if $d_i = 1$, then x_i is included in this term.

$d_i = 2$ $\quad 1$

As an example let us consider

$$\mathbf{f} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 1 \\ 1 \\ 0 \\ 1 \end{bmatrix}, \text{ with } \boldsymbol{\delta}_0(\mathbf{f}) = \boldsymbol{\delta}_0\left(\begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 1 \\ 1 \\ 0 \\ 1 \end{bmatrix}\right) = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}.$$

Therefore,

$$\mathbf{A}_n^T\boldsymbol{\delta}_0(\mathbf{f}) = \begin{bmatrix} 1&0&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0&0 \\ 1&1&0&0&0&0&0&0 \\ 0&0&1&0&0&0&0&0 \\ 0&0&0&1&0&0&0&0 \\ 0&0&1&1&0&0&0&0 \\ 1&0&1&0&0&0&0&0 \\ 0&1&0&1&0&0&0&0 \\ 1&1&1&1&0&0&0&0 \\ 0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&1&0&0 \\ 0&0&0&0&1&1&0&0 \\ 0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&0&1 \\ 0&0&0&0&0&0&1&1 \\ 0&0&0&0&1&0&1&0 \\ 0&0&0&0&0&1&0&1 \\ 0&0&0&0&1&1&1&1 \\ 1&0&0&0&1&0&0&0 \\ 0&1&0&0&0&1&0&0 \\ 1&1&0&0&1&1&0&0 \\ 0&0&1&0&0&0&1&0 \\ 0&0&0&1&0&0&0&1 \\ 0&0&1&1&0&0&1&1 \\ 1&0&1&0&1&0&1&0 \\ 0&1&0&1&0&1&0&1 \\ 1&1&1&1&1&1&1&1 \end{bmatrix}\begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 2 \\ 0 \\ 2 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 2 \\ 0 \\ 2 \\ 3 \\ 0 \\ 3 \end{bmatrix}$$

and

$$\boldsymbol{\delta}_0[\mathbf{A}_n^T\boldsymbol{\delta}_0(\mathbf{f})] = \boldsymbol{\delta}_0\left(\begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 2 \\ 0 \\ 2 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 2 \\ 0 \\ 2 \\ 3 \\ 0 \\ 3 \end{bmatrix}\right) = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}.$$

Our output vector **y** is therefore $\mathbf{y} = \mathbf{B}_n\left\{\boldsymbol{\delta}_0[\mathbf{A}_n^T\boldsymbol{\delta}_0(\mathbf{f})]\right\}$

$$= \begin{bmatrix} 1&0&1&0&0&0&1&0&1&0&0&0&0&0&0&0&0&0&1&0&1&0&0&0&1&0&1 \\ 0&1&1&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&1&1 \\ 0&0&1&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&1 \\ 0&0&0&1&0&1&1&0&1&0&0&0&0&0&0&0&0&0&0&0&0&1&0&1&1&0&1 \\ 0&0&0&0&1&1&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1 \\ 0&0&0&0&0&1&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&1 \\ 0&0&0&0&0&0&1&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&1 \\ 0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1 \\ 0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1 \\ 0&0&0&0&0&0&0&0&0&1&0&1&1&0&0&0&1&1&0&1&0&1&0&0&0&0&1 \\ 0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&1&1&0&1&1&0&0&0&0&0&1&1 \\ 0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&1&0&0&0&0&1&0&1&0&0&0&1 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&1&1&0&0&0&0&1&0&1&0&0&1 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&1&0&0&0&0&0&1&0&1&0&1 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&1&0&0&0&1&0&1&0&0&1 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&1&0&0&0&1&0&1&0&1 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&1&1&0&0&1&1&0&1 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&1&1&0&1&1&0&1 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&1&1&0&1&1&1 \end{bmatrix}\begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 & 000 \\ 4 & 001 \\ 0 & 002 \\ 0 & 010 \\ 4 & 011 \\ 0 & 012 \\ 0 & 020 \\ 2 & 021 \\ 0 & 022 \\ 2 & 100 \\ 5 & 101 \\ 1 & 102, term\ x_1\bar{x}_2 \\ 0 & 110 \\ 4 & 111 \\ 0 & 112 \\ 0 & 120 \\ 2 & 121 \\ 0 & 122 \\ 0 & 200 \\ 2 & 201 \\ 0 & 202 \\ 0 & 210 \\ 2 & 211 \\ 0 & 212 \\ 0 & 220 \\ 1 & 221, term\ x_3 \\ 0 & 222 \end{bmatrix}.$$

A Boolean expression representing *f*, then, would be $f(x_1x_2x_3) = x_1\bar{x}_2 + x_3$. This can be verified by calculating

the truth table for the above expression; it will, indeed, match our input vector **f**.

2. COMPRESSION METHOD

The following steps outline the method used to compress the image. Note that there are two general stages: the linear prediction preprocessing stage (applicable for images with sufficient spatial correlation [4]) and the Boolean minimization (encoding) stage [5]. Let us define $p_{x,y}$ as the intensity value of the pixel in column *x* and row *y* of the *X*-pixels wide by *Y*-pixels high image at hand. ($p_{0,0}$ represents the intensity value of the pixel in the upper left hand corner of the image.)

1. The preprocessing stage
   a. Apply 2-D linear prediction to the image
   b. In the topmost row of the image, perform horizontal one-dimensional linear prediction from left to right:

$$\forall x \in \{1,2,3,\ldots,X-1\}: p_{x,0} = \left(p_{x,0} - p_{x-1,0} + 128\right) \bmod 256 \cdot \quad (4)$$

   c. In the leftmost column of the image, perform vertical one-dimensional linear prediction from top to bottom:

$$\forall y \in \{1,2,3,\ldots,Y-1\}: p_{0,y} = \left(p_{0,y} - p_{0,y-1} + 128\right) \bmod 256 \cdot \quad (5)$$

   d. For all other pixels, perform 2-dimensional linear prediction:

$$\forall x \in \{1,2,3,\ldots,X-1\}, y \in \{1,2,3,\ldots,Y-1\}:$$

$$p_{x,y} = \left[p_{x,y} - \frac{1}{2} \cdot \left(p_{x,y-1} + p_{x-1,y}\right) + 128 \right] \bmod 256 \cdot \quad (6)$$

   e. Compute histogram of resultant image.
   f. Find and remove all intensity levels that occur with zero probability, reordering all intensity levels by counting from zero upward. Store this table of available intensity levels to the output file.
   g. Re-code the intensity levels for each pixel in the image to its Gray-coded value.
2. The Boolean minimization stage
   a. Break input image into *n* by *m* blocks. Output size of input image and size of blocks to file.
   b. Divide each block into its bit planes.
   c. For each block in each bit plane, scan the block according in a top to bottom, alternating left-right ordering, as shown below. Load these binary values into a vector.

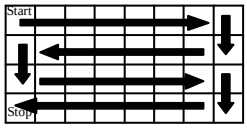

Figure 3. Pixel scanning order.

   d. If the vector is all 1's or all 0's, record an appropriate escape sequence to the output file noting this.
   e. Re-order the elements in the vector using Gray code ordering.
   f. If all elements in the vector are not the same value, perform Boolean minimization of the vector as described in the transform-based approach above. Try the above approach both before and after performing Boolean inversion of every element in the vector, making note of which vector (on-set vs. off-set) is minimized to fewer minterms. Store the more efficiently minimized set of minterms to the file [1, 2].
   g. Determine the probabilities for each element ($\bar{x}_i$, x_i , or 1) being in minterms for this block, and construct and output an appropriate 1- or 2-bit code for these elements.
   h. Output the number of terms to a temporary bit stream, followed by the entropy-encoded value of each element of every term.
   i. If the length of this bit stream is less than that which would be required to transmit the raw pixel data from the block, deem the block efficiently minimizable and write the temporary bit stream to the file. If not, simply write an escape sequence noting that image data is to follow, followed by the Boolean pixel data for the block and bit plane at hand.

3. DECOMPRESSION

In general, the above steps are completed in reverse order for decompression.

1. Reading of header information
   a. Read image size and block size from file.
   b. Read table of intensity levels from file.
2. Decoding of blocks – for each block stored:
   a. If the block was noted as being all 1's or 0's, store all 1's or 0's (as applicable) to the appropriate block and bit plane.
   b. If the block was not noted as being all 1's or 0's and was not noted as being "effectively minimizable," read in raw image data that follows. Inverse-Gray code the order of this data and store in the appropriate block and bit plane according to the same top-to-bottom, alternating left-right ordering outlined in the compression algorithm.
   c. If the block was not noted as being all 1's or 0's and was noted as being "effectively minimizable," perform the following steps for Boolean expansion:
      i. Read in the code representation for the elements of the minterms to follow.
      ii. Read in the number of minterms stored.
      iii. According to the code scheme from step **i.**, read in each minterm for the current block.
      iv. Run through each combination of input variables to the minterms, and evaluate the Boolean expression for each. Store the series of outputs in a vector.
      v. If the minterms were noted to represent an "off-set," take the Boolean inverse of every element of this vector [1, 2].
      vi. Re-order this vector using inverse Gray-code ordering.

vii. Store the data in the vector to the appropriate block and bit plane according to the same top-to-bottom, alternating left-right ordering outlined in the compression algorithm.

3. Preprocessing reversal
   a. For each pixel, inverse Gray-code the intensity values.
   b. Restore the original histogram by inserting the appropriate zero-probability intensities into the histogram according to the table retrieved in *1b*.
   c. Perform inverse prediction on the image
      i. In the topmost row of the image, perform horizontal one-dimensional inverse linear prediction from left to right:

$$\forall x \in \{1,2,3,\ldots,X-1\}: p_{x,0} = \left(p_{x,0} + p_{x-1,0} + 128\right) \bmod 256 . \quad (7)$$

      ii. In the leftmost column of the image, perform vertical one-dimensional inverse linear prediction from top to bottom:

$$\forall y \in \{1,2,3,\ldots,Y-1\}: p_{0,y} = \left(p_{0,y} + p_{0,y-1} + 128\right) \bmod 256 . \quad (8)$$

      iii. For all other pixels, perform 2-dimensional inverse linear prediction:

$$\forall x \in \{1,2,3,\ldots,X-1\}, y \in \{1,2,3,\ldots,Y-1\}:$$

$$p_{x,y} = \left[p_{x,y} + \frac{1}{2} \cdot \left(p_{x,y-1} + p_{x-1,y}\right) + 128 \right] \bmod 256 . \quad (9)$$

It is important to note that when decompressing a bit stream represented as a minimized Boolean function, it is not necessary to use computationally intensive matrix multiplications. Conversely, the original bit stream is attained simply by progressing through every combination of input values, evaluating the expressions for the minterms in each combination, and storing the Boolean output of these evaluations in a list.

Table 1. Compression ratios[1] of different algorithms

Image	Our Approach (8x8 blocks)	Arithmetic Coding	LZW	RLE	WINZIP
Girl	1.73	1.08	1.39	1.00	1.37
Lena	1.57	1.10	1.23	0.99	1.19

4. FAST IMPLEMENTATIONS

Fast implementations for $\mathbf{A}_n^T$ and $\mathbf{B}_n$ are inspired by their Kronecker expansions. The computational advantage of these fast implementations is explained by the fact that the fast implementations prescribe a method for combining smaller matrix multiplications to form larger, more complex ones. The fast implementation of $\mathbf{A}_n^T$ is given recursively by:

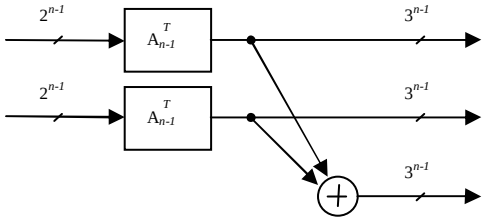

Figure 4. Generalized fast implementation of $\mathbf{A}_n^T$.

The fast implementation of $\mathbf{B}_n$ is given recursively by:

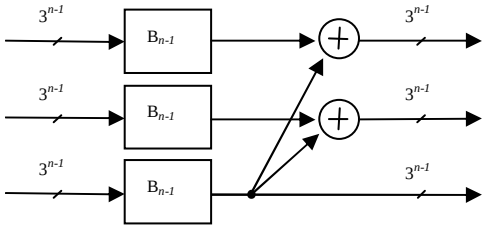

Figure 5. Generalized fast implementation of $\mathbf{B}_n$.

As an example, let us demonstrate a flow diagram for the fast implementation of the $\mathbf{A}_2^T$ transform.

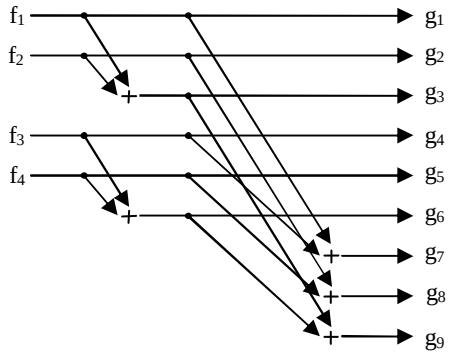

Figure 6. Fast implementation of $\mathbf{A}_2^T$.

[1] Calculated by: (original file size)/(compressed file size)

Likewise, the flow diagram for the fast implementation of the $\mathbf{B}_2$ transform is:

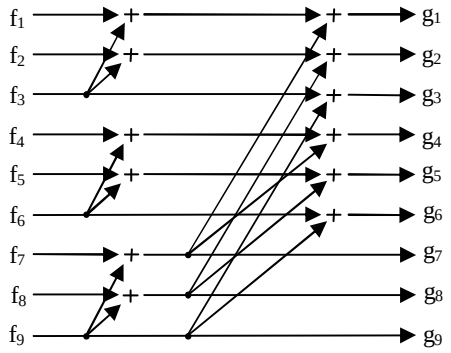

Figure 7. Fast implementation of $\mathbf{B}_2$ [3].

5. CONCLUSIONS

The utilization of transform-based methods for Boolean compression results in a low-complexity compression algorithm whose performance is comparable to that of algorithms of equal or greater complexity. Because the matrices used ($\mathbf{A}_n^T$ and $\mathbf{B}_n$) consist of 1- and 0-valued elements, no multiplies (only additions) are needed to perform vector-matrix multiplication. Also, fast implementations of these matrices exist.

Furthermore, these matrices are not required in the decompression process; Boolean expansion (inverse-minimization) is achieved simply by progressing through every combination of input values, evaluating the expressions for the minterms in each combination, and storing the Boolean output of these evaluations in a list. The decompression process is, then, much less complex than the compression process.

Further research will be conducted to investigate prediction techniques optimized for Boolean minimization, extending the algorithm to achieve lossy compression using our paradigm, and hardware implementations of various parts of the compression scheme.

6. ACKNOWLEDGMENTS

The authors wish to thank Dr. Robert Gonsalves of Tufts University, whose ideas were helpful in the formulation of the linear prediction stage of the algorithm.

7. REFERENCES

[1] Augustine, J., W. Feng, A. Makur, and J. Jacob. "Switching Theoretic Approach to Image Compression," *Signal Processing*, no. 44, March 1995.

[2] Chaudhary, A., J. Augustine, and J. Jacob. "Lossless Compression of Images Using Logic Minimization," *Proceedings, International Conference on Image Processing*, 1996.

[3] Agaian, S., J. Astola, and K. Egiazarian. *Binary Polynomial Transforms and Nonlinear Digital Filters*, Marcel Dekker, Inc., New York, 1995.

[4] Jayant, N. and P. Noll. *Digital Coding of Waveforms: Principles and Applications to Speech and* Video, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1984.

[5] Mandyam, G., N. Ahmed, and S. Stearns. "A Two-Stage Scheme for Lossless Compression of Images," *Proceedings, IEEE International Symposium on Circuits and Systems*, vol. 2, 1995.