International Journal of Innovative Computing, Information and Control
Volume **8**, Number **1(A)**, January **2012**

ICIC International ©2012 ISSN 1349-4198
pp. 329-345

A BINARY IMAGE HIDING-COMPRESSION METHOD USING BFT LINEAR QUADTREE AND LOGIC-SPECTRA

Yu-An Ho[1], Yung-Kuan Chan[2,*], Chwei-Shyong Tsai[2]
Yen-Ping Chu[1] and Hsien-Chu Wu[3]

[1]Department of Computer Science and Engineering
[2]Department of Management Information Systems
National Chung Hsing University
No. 250, Kuokuang Rd., Taichung 402, Taiwan
*Corresponding author: ykchan@nchu.edu.tw; {yaho; ypchu; tsaics}@nchu.edu.tw

[3]Department of Computer Science and Information Engineering
National Taichung Institute of Technology
No. 129, Section 3, San Min Rd., Taichung 404, Taiwan
wuhc@ntit.edu.tw

Received September 2010; revised January 2011

Abstract. *This paper proposes a binary image compression method, called QLS compression method, which integrates BFT linear quadtree and logic-spectra techniques to losslessly compress a binary image. This method employs a breadth first traversal linear quadtree to divide the image into nonoverlapping blocks, and then uses logic functions and spectral techniques to encode the blocks. This paper also presents a QLS hiding-compression method to encode a cover image and embed secret data in the cover image during encoding the cover image. The stego image created by the QLS hiding-compression method is quite similar to the cover image.*

Keywords: Linear quadtree, Image compression, Data hiding, Logic function, Lossless binary image compression method

1. **Introduction.** A digital image generally contains a large amount of data and needs much memory space to hold the data. Due to the limitation of network bandwidth, it indeed takes much time to transmit the data within or between computer networks. As a result, it is undoubtedly necessary to develop a good image compression technique so that the memory space required for storing digital images and the time needed for online transmission can be effectively reduced [5,11,14,25].

Binary image is one of the commonly used image formats, such as FAX and document images [21]. Some logic functions based lossless binary image compression techniques have been proposed [8,12,23,29]. Chaudhary et al. [8] partitioned a binary image into small blocks, and then the mixed blocks are converted to Boolean switching functions and subjected to minimization for leading to a compact representation. Mateu-Villarroya and Prades-Nebot [23] presented a lossless image compression algorithm based on ordered binary-decision diagrams (OBDDs). This algorithm uses Gray code instead of sequential binary code for finding an OBDD which can describe the image exactly and then it encodes the OBDD. The coding algorithm of OBDDs can efficiently reduce redundant data in the image and achieve a good compression result. A lossless image compression technique based on coding schemes and patterns, including minterm, cube and coordinate data coding, Walsh, triangular and ReedMuller weights based patterns, ReedMuller spectra and reference row technique, has also been proposed [12,29]. In this technique, a two-dimensional differencing operation is first applied on the image. The difference image is

segmented and classified into all-black, all-white or mixed blocks. Each all-black or mixed block in the non-overlapping blocks is represented by a variable block-size segmentation and coding scheme. However, the main drawback of the above-mentioned methods is that much memory space is required to hold the coordinates of the blocks.

Many researchers adopted quadtree structure to encode a binary image [16,20,36]. The image region is recursively decomposed into quadrants until each quadrant includes either all-black or all-white pixels. A quadtree can be entirely stored as a tree structure. Alternatively, one can store only the encodings of paths from the root quadrant to individual subquadrants that contain all-black pixels or all-white pixels. The maximal quadtree blocks or quadrants can be obtained through the regular hierarchical decomposition for saving data storage.

This paper proposes a lossless binary image compression method which applies quadtree structure to decomposing a binary image into blocks and then uses the coding techniques [12,29] based on minterm coding, GPMPRM expansion, coordinate data coding, pattern matching, and the reference row technique to encode the blocks. We call the binary image compression method a quadtree and logic-spectra based image compressiom method (QLS compression method). In most cases, the QLS compression method can provide a better compression ratio than the compression methods proposed by Falkowski [12] and Wang et al. [33].

Since the prevalence of the Internet and multimedia has increased exponentially in recent years, people can exchange and transmit a large amount of data via computer networks. However, the security mechanisms of an ordinary computer network, especially a wireless one, are not so sufficient that data transmitted on it may be easily intercepted. Thus, how to prevent data transmitted on a computer network from being hacked by unauthorized users has become an interesting issue to many researchers. Most of the technologies used currently [4,7,10,17,19,25,26,31,32,34,36] embed confidential data in a certain digital medium, and then transmit the data to a receiver via this medium. Even if the data are intercepted halfway, people who intercept the data can merely see the outer appearance of the medium, rather than the initially embedded data.

Image hiding is to embed secret data in an image. In general, the image that carries the secret data is called a cover image. After secret data are embedded, the cover image becomes a stego image. This process of hiding is called embedding. Therefore, the purpose of image hiding is to conceal secret data in a cover image by imposing imperceptible changes on the stego image [6,18].

So far, numerous data hiding methods [4,19,31,35] have been proposed, but most of them can embed the secret data only in a grey-level or a color image. In a binary image, a pixel is generally described by only one bit. If one bit in an all-black or an all-white region is changed, human eyes can easily detect the change. Hence, only the proximity of edge can be used to carry secret data. Since not much space in a binary image can be used to conceal secret data, it is much more difficult to hide secret data in a binary image than in a gray-level or a color image.

Some methods have been proposed to hide secret data in a binary image. Matsui and Tanaka [23] embedded secret data in a dithered image by changing the dithering patterns and in fax images by changing the run-lengths. However, this method cannot be applied to general binary images. Maxemchuk and Low [2,21,24,37] proposed three different methods for embedding secret data in text documents: line shift coding, word-shift coding and feature coding. Although these methods can theoretically be defeated, they require interactive human intervention and are not cost-effective in practice. Wu and Liu [37] hid a moderate amount of data in a general binary image. Their method manipulates "flippable" pixels to enforce specific block based relationship to embed a

significant amount of data without causing noticeable artifacts. The method can hide annotation labels or other side information and verify whether a binary document has been tampered with or not. This technique determines the hidden capacity according to the size of blocks.

This paper presents a QLS hiding-compression method to embed secret data in a cover image during encoding the cover image. This method can not only provide a high hiding capacity but also create a stego image quite similar to the cover image.

2. **The QLS Compression Method.** The QLS compression method contains two phases: data encoding and data decoding phases. The data encoding phase is to compress a binary image, while the data decoding phase is to reconstruct the original binary image from the compressed data. This section will describe both phases in detail.

2.1. **Data encoding phase.** The data encoding phase comprises three stages: Exclusive-OR, quadtree compressing and block coding. The Exclusive-OR stage is to make black pixels appear only around the edge in the image with all the rest being white. In this way, the entropy [1] of the image can be effectively lowered with enhanced data encoding performance. Then, in the quadtree compressing stage, a breadth first traversal linear quadtree (BFT linear quadtree) is used to partition the image, generated in the Exclusive-OR stage, into blocks. The block coding stage further encodes the blocks.

2.1.1. *Exclusive-OR stage.* According to Shannon [30], there is a fundamental limit on lossless data compression [1,27]. This limit, called entropy, is terminologically referred to as H. The exact value of H depends on the data more specifically, the statistical nature of the data. It is possible to compress the data, in a lossless manner, with a compression rate close to H. Assume that the data are statistically independent. Let k be the number of different items in the data, and P_i be the occurrence probability of the i-th item in the data. The entropy H of the data is defined as:

$$H = \sum_{i=1}^{k} p_i \times \log_2 \left(\frac{1}{p_i} \right)$$

According to Shannon, by entropy, the number of bits required to represent a pixel is at least H bits/pixel, after removing the redundant data from the image. Therefore, one may get higher efficiency in storage space for a digital image with low entropy. A binary image is usually composed of big all-black areas and all-white areas. To enhance the encoding efficiency, in the QLS compression method, the black pixels only exist near the boundaries of objects with all the other pixels being white. In this stage, an image f_{Eh} is created from f_0 by executing an Exclusive-OR logic operation for each two adjacent pixels in f_0. The compression method scans the whole image pixel by pixel. If the color of the currently being visited pixel differs from the color of its predecessor, then the method sets the color of the pixel in f_{Eh}, that is in the same position as the currently visited pixel in f_0, to black; otherwise, it is set to be white. Each row of f_{Eh} can be created by the following formula, where '0' stands for a white pixel and '1' a black pixel:

$$f_{Eh}(i,j) = \begin{cases} f_0(i,j), & \text{if} \quad j = 0, \\ f_0(i,j) \oplus f_0(i,j-1), & \text{otherwise}, \end{cases} \tag{1}$$

where $\oplus$ is the Exclusive-OR logic operation, and $f_0(i,j)$ and $f_{Eh}(i,j)$ are the pixels located at the coordinates (i,j) on f_0 and f_{Eh}.

Similarly, this method also scans each pixel in f_{Eh} pixel by pixel to create an image f_{Ev} by performing the Exclusive-OR logic operation between each two vertical neighboring

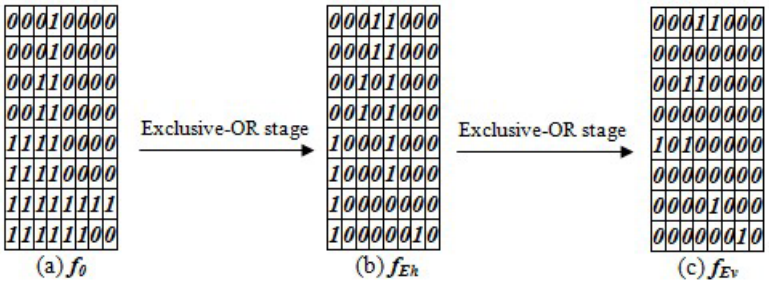

FIGURE 1. A binary image f_0 with 8×8 pixels and its f_{Eh} and f_{Ev}

pixels in f_{Eh}. f_{Ev} can be constructed by the following formula:

$$f_{Ev}(i,j) = \begin{cases} f_{Eh}(i,j), & \text{if} \quad i = 0, \\ f_{Eh}(i,j) \oplus f_{Eh}(i-1,j), & \text{otherwise.} \end{cases} \tag{2}$$

For example, Figure 1(a) shows an 8×8 binary image f_0. After the Exclusive-OR stage, f_0 is changed into f_{Eh} and then into f_{Ev}, as respectively shown in Figures 1(b) and 1(c). In this example, the entropy of the image in Figure 1(a) is 0.99, while the entropy of the image in Figure 1(c) is 0.54. As the lower entropy indicates, the image in Figure 1(c) is capable of offering higher compression efficiency.

TABLE 1. The numbers of black and white pixels

Image	f_0			f_{Ev}		
	Black pixel	White pixel	Entropy	Black pixel	White pixel	Entropy
CCITT 1	155,591	3,950,137	0.2326	51,476	4,054,252	0.0972
CCITT 2	184,240	3,921,488	0.2642	28,552	4,077,176	0.0598
CCITT 3	337,052	3,768,676	0.4095	78,392	4,027,336	0.1363
CCITT 4	509,635	3,596,093	0.5411	204,154	3,901,574	0.2852
CCITT 5	317,707	3,788,021	0.3929	90,380	4,015,348	0.1526
CCITT 6	207,110	3,898,618	0.2883	46,010	4,059,718	0.0887
CCITT 7	356,850	3,748,878	0.4261	186,544	3,919,184	0.2667
CCITT 8	1,766,467	2,339,261	0.9859	52,294	4,053,434	0.0984

Table 1 demonstrates the numbers of white and black pixels on f_0 and f_{Ev}, where CCITT 1 to CCITT 8 are the test images [38] shown in Figure 9. Each of the test images consists of 2376×1728 pixels. Table 1 also displays the entropies of each f_0 and f_{Ev}; that is, the number of white pixels in f_{Ev} is much greater than that in f_0. It also tells that the entropies of f_0 are much larger than that of f_{Ev}.

2.1.2. *Quadtree compressing stage.* A quadtree is a well-known data structure that describes the spatial information of an image. It has been amply depicted in a number of publications [16,20,36]. A quadtree is obtained by recursively dividing a binary image into NW, NE, SW and SE quadrants [16,20,36]. A leaf node in the tree stands for a quadrant of the same color. Whether to assign the black or white color to the node depends on whether the quadrant consists entirely of black or white pixels. Otherwise, the quadrant is depicted by an internal node and further subdivided into four equally-sized subquadrants until each subquadrant is completely filled with black or white pixels. For example, Figure 3 is the corresponding quadtree of the image in Figure 2 which is the f_{Ev} in Figure 1.

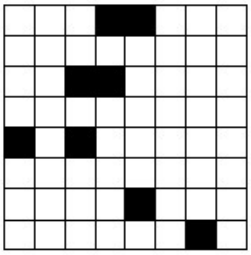

FIGURE 2. A binary image

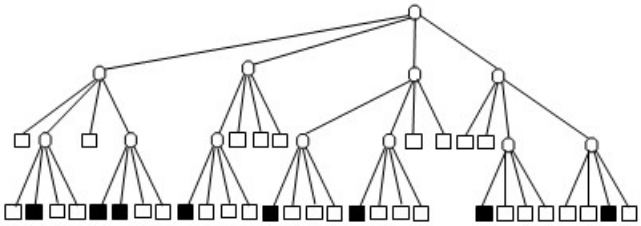

FIGURE 3. The related quadtree of the binary image in Figure 2

LQ: (1 1111 0101 1000 1100 0011)

FIGURE 4. The corresponding BFT linear quadtree of the quadtree in Figure 3

The QLS compression method uses a quadtree to partition f_{Ev} into blocks; in the block coding stage, it then further compresses the blocks. A higher quadtree requires much more memory space to hold the tree. To build a shorter quadtree, the QLS compression method therefore repeatedly partitions f_{Ev} into quadrants so that each quadrant consists of only white pixels or contains exact 8×8 pixels. Each of the 8×8 quadrants is called a basic block. The QLS compression method first divides f_{Ev} into non-overlapping 8×8 blocks and each pixel in image f_B stands for a basic block (one 8×8 pixels) in f_{Ev}. In f_B, pixel '0' corresponds to one all-white block, and pixel '1' maps to one block with at least one black pixel in f_{Ev}.

About 90% of memory space required to store a quadtree is wasted on holding the pointers in this quadtree [3,36]. Thus, the concept of linear quadtree is proposed to get rid of the pointers. A linear quadtree can be considered to be a linear array of a certain type of structure elements so that the tree structure can be implicitly preserved.

This paper proposes a breadth first traversal (BFT) linear quadtree LQ to segment f_B into small blocks. LQ can be obtained by traversing its corresponding quadtree in the breadth first traversal order. In LQ, '0' denotes a leaf node and '1' an internal node. Figure 4 shows the corresponding LQ of the quadtree in Figure 3. One can directly

convert a binary image f_B into LQ by executing the following $AlgorithmBFT\_LQ$.

Algorithm $BFT\_LQ()$
  for $(k = l \ to \ 1)$
    scan the pixels on f_B
    for each four-pixels $f_B[2i][2j], f_B[2i][2j+1], f_B[2i+1][2j], f_B[2i+1][2j+1]$
      if $f_B[2i][2j] = f_B[2i][2j+1] = f_B[2i+1][2j] = f_B[2i+1][2j+1] =$ '0' then
        $f'[i,j] =$ '0'
      else $f'[i,j] =$ '1'
        if $k \neq l$ then
          $L' = L'||f_B[2i][2j]||f_B[2i][2j+1]||f_B[2i+1][2j]||f_B[2i+1][2j+1]$
  $f_B = f'$
  $LQ = L'||LQ$
  $LQ =$ '1'$||LQ$

$AlgorithmBFT\_LQ$ constructs the LQ of a $2^l \times 2^l$ input image f_B from bottom to top. The algorithm scans each pixel on f_B. It first views each pixel as a quadrant, then combines each four quadrants $f_B[2i][2j]$, $f_B[2i][2j+1]$, $f_B[2i+1][2j]$ and $f_B[2i+1][2j+1]$ into one at the k-th level, and generates a $2^{k-1} \times 2^{k-1}$ image f'. If the colors of all the four quadrants are '0', then $f'[i][j] =$ '0'; otherwise, $f'[i][j] =$ '1' and $L' = L'||f_B[2i][2j]||f_B[2i][2j+1]||f_B[2i+1][2j]||f_B[2i+1][2j+1]$. The operation $A||B$ returns the concatenation of A and B. The algorithm repeatedly extracts the data of the quadtree at the $(k-1)$-th level. LQ only records the information from level 0 to level $l-1$ and the internal nodes (representing by '1') in level $l-1$ will link to four leaf nodes in level l each corresponding to one basic block. Figure 5 illustrates the steps of transforming f_B into its corresponding LQ. Figure 5(f) demonstrates the LQ of the image in Figure 5(a).

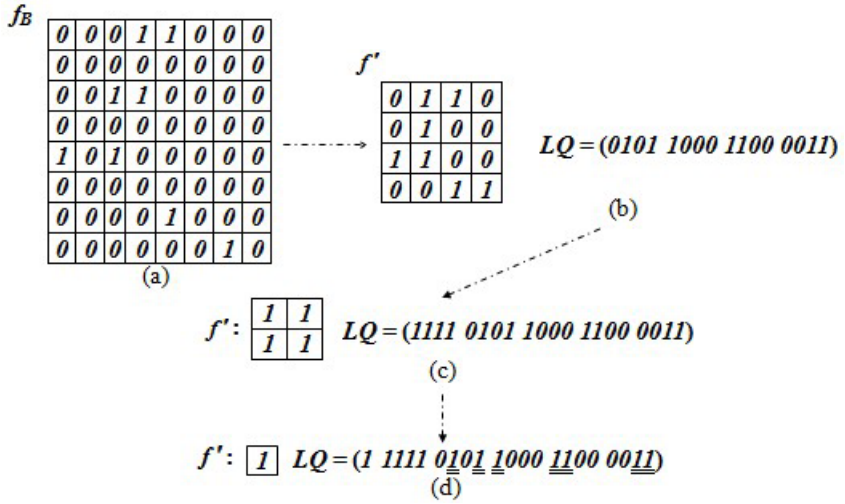

FIGURE 5. The steps to transform the image f_B into a BFT linear quadtree

Let n_1 be the number of '1'-bits and n_0 the number of '0'-bits in a quadtree. Since each internal node in a quadtree connects to four children, $n_0 = 3n_1 + 1$, each of the last $\frac{3n_1 + 1 - n_0'}{4}$ '1'-bits in LQ links to 4 leaf nodes (each of which respectively corresponds to one basic block), where n_0' is the number of '0'-bits in LQ and there are n_1 '1'-bits in LQ. We call the last $\frac{3n_1 + 1 - n_0'}{4}$ '1'-bits in LQ are dangled bits since their children are

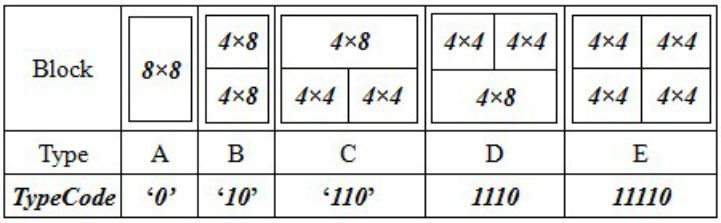

FIGURE 6. The five types of basic blocks and their type codes

not written down in LQ. The '1'-bits in LQ marked by underlines in Figure 5(d) are the dangled bits.

The front part of LQ (or called the pre-substring) is usually filled with '1' bits, so we call these '1' bits the $1-presubstring$. The remaining part (the post-substring) is called $postsubstring$ that has intertwined '0' bits and '1' bits. LQ can be translated to $LQ = N_1||postsubstring$. N_1 records the number of bits in $1-presubstring$. Hence, LQ in Figure 5(d) can be described by $5||$'0101100011000011'.

2.1.3. *Block compressing stage.* Each dangled bit in LQ links to four basic blocks, each of which consists of 8×8 pixels. The block compressing stage is to compress the basic blocks. This approach first categorizes each basic block into one of five types from A to E. Figure 6 shows the five types of basic blocks and their type codes $TypeCode$. A type A block is the whole basic block, called an 8×8-block. For type B, the basic block is divided into two 4×8-blocks; we call the block a 4×8-block. The block in types C and D is split into two 4×4-blocks and one 4×8-block; the block of type E is separated into four 4×4-blocks. The flow chart in Figure 7 explains how to categorize each basic block B. In this flow chart, $NBP(B)$ is the number of black pixels in B.

To encode B into a block code BC, the type code of B is first assigned to $BC = TypeCode$. The detailed steps to encode an 8×8-, 4×8- and 4×4-block are described in Table 2. Some functions used in this table will be introduced. The minterm of a '1'-bit located at the coordinates (x, y) in a block B is $x||y$. Here x (y resp.) is described by 3 bits. The following $Function\ Minterm(B)$ describes the minterm of the 1-bits in B.

$Function\ Minterm(B)$
  If $NBP(B) = 1$ then return ('0'$||m_1$)
    else return ('1'$||m_1||m_2$)
  $/* m_1$ is the minterm of the first 1-bit in $B */$
  $/* m_2$ is the minterm of the second 1-bit in $B */$

The following function $Coordinate(B)$ is applied to put down the coordinates of the '1'-bits in B.

$Function\ Coordinate(B)$
  for each row in B
    for each '1'-bit in this row
      $BC' = BC'||$'1'$||m$
    if the last pixel of this row is '0'-bit then $BC' = BC'||$'0'
  return (BC')
  $/* m$ described by $\log_2\lceil m\rceil$ bits is the difference between the coordinates of the current '1'-bit and last '1'-bit in this row. $*/$

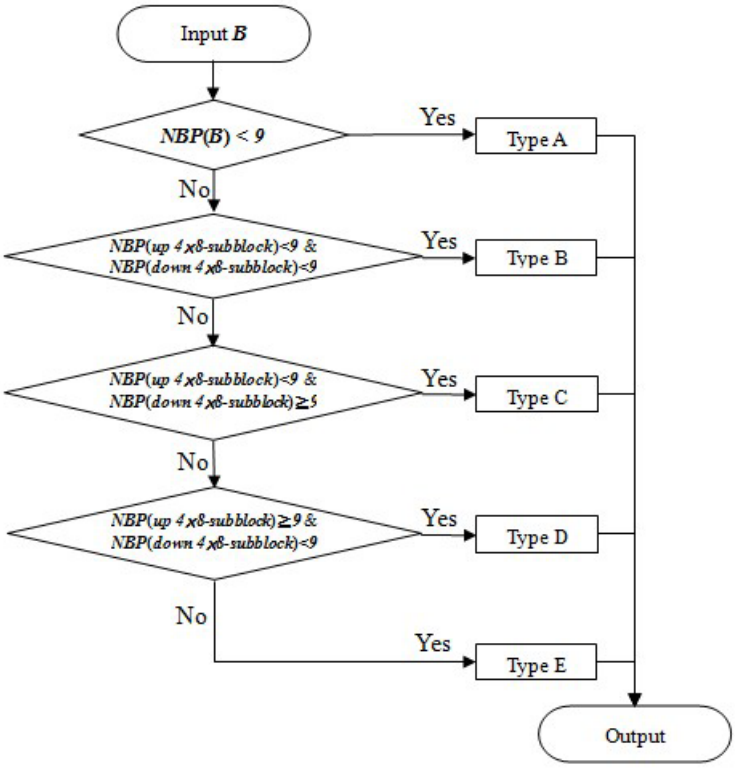

FIGURE 7. The flow chart to categorize each basic block

An n-variable Boolean switching function can be expressed as a canonical ReedMuller expansion [8,11,13] of 2^n product terms as follows:

$$f(x_1, x_2, \cdots, x_{n-1}, x_n) = \bigotimes_{j=0}^{2^n-1} a_j \prod_{i=1}^{n} \widehat{x}_i^{j_i},$$

where $\bigotimes$ denotes the modulo-2 addition, $a_j \in \{0, 1\}$ is called a ReedMuller coefficient, and $j_i \in \{0, 1\}$ is called the power of x_i so that $\langle j_1 j_2 \ldots j_n \rangle$ is equal to the binary representation of j. If $j_i = 0$, the i-th literal is absent in the product term $\prod_{i=1}^{n} \widehat{x}_i^{j_i}$; otherwise, it is presented in the product term. When each literal ($\widehat{x}_i$, for $i = 1, \ldots, n-1, n$) throughout above formula assumes either complemented or non-complemented but not both forms simultaneously, it is known as the fixed polarity ReedMuller (FPRM) expansion [9,13,15].

For each cube $a_j \prod_{i=1}^{n} \widehat{x}_i^{j_i}$, if $j_i = 0$, literal $\widehat{x}_i$ is absent and then $\widehat{x}_i$ is replaced by a "dont care" literal x. Hence, a literal $\widehat{x}_i$ can be x_i, $\overline{x}_i$, or x but can be only one of them. Table 3 shows the codes where each $\widehat{x}_i$ will be replaced in the cube. In Table 3, if x_i occurs most frequently in the cube, the codes of the first row are used; if $\overline{x}_i$ occurs most frequently, the codes of the second row are employed; otherwise, those of the third row are applied. The code $Prefix$ describes which of the three cases is in the cube. $Function\ GPMPRM()$ is used to depict a cube.

TABLE 2. The detailed steps to code an 8×8-, 4×8- and 4×4-block

Block	Condition	Header	Algorithm
	NBP(B)=0	***'0'***	***BC=BC‖'0'***.
8×8	***0<NBP(B)<3***	***'110'***	The index of each ***1***-bit in ***B*** is depicted by a minterm: ***BC=BC‖'110'‖Minterm(B)***.
	2<NBP(B)<9	***'10'***	***BC=BC‖'10'‖Coordinate(B)***.
	The GPMPRM expansion contains less than three cubes.	***'1111'***	The GPMPRM expansion is computed. If the GPMPRM expansion contains less than three cubes, then ***BC=BC‖'1111'‖GPMPRM( ).***
	Not compressed by above approaches	***'1110'***	If ***B*** is divided into ***4×8***- or ***4×4***- blocks and the memory space required to record these ***4×8***- and ***4×4***- blocks is more than that to hold ***B***, then ***B*** is directly written down. Hence ***BC=BC‖ '1110'‖B.***
4×8	***NBP(B)=0***	***'110'***	***BC=BC‖'110'***.
	0<NBP(B)<7	***'0'***	***BC=BC‖'0'‖Coordinate(B)***.
	The GPMPRM expansion contains at least one and less than three cubes.	***'111'***	The GPMPRM expansion is computed. If the GPMPRM expansion contains less than three cubes, then ***BC=BC‖'111'‖GPMPRM( ).***
	Not compressed by above approaches	***'10'***	If ***B*** is divided into two ***4×4***-blocks and the memory space required to record the ***4×4***- blocks is more than that to hold ***B***, then ***B*** is directly written down. Hence ***BC= '10' ‖ B***
4×4	***NBP(B)=0***	***'1110'***	***BC=BC‖'1110'***.
	0<NBP(B)<3	***'10'***	***BC=BC‖'10'‖Coordinate(B)***.
	B matches one of possible pattern combinations in Figure 11.	***'110'***	***BC=BC‖'110'‖ Pattern_matching( ).***
	The GPMPRM expansion contains at least one and less than three cubes.	***'1111'***	The GPMPRM expansion is computed. If the GPMPRM expansion contains less than three cubes, then ***BC=BC‖'1111'‖GPMPRM( ).***
	The total number of different bits between the first and the second rows, as well as the second and the fourth rows is less than or equal to ***1*** bit	***'01'***	***BC=BC‖'01'‖Reference_Row(B)***.
	Otherwise	***'00'***	***B*** is directly written down; ***BC= BC‖'00'‖B***.

TABLE 3. Cube encoding

$Prefix$	x_i	$\overline{x}_i$	x
00	0	10	11
01	11	0	10
10	10	11	0

Function $GPMPRM()$

```
if No. of cubes = 1 then
   Return('0' ||Prefix||the code of cube 1)
else
   Return('1'||Prefix||the code of cube 1||the code of cube 2)
```

A 4×4-block B is compared with a set of four frequently occurring patterns shown in Figure 8. The possible combinations are that B.

(1) Exactly matches with one of the patterns.

(2) Exactly matches with the transpose of one pattern.

(3) Matches with one of the patterns or the transpose of one pattern except only one pixel.

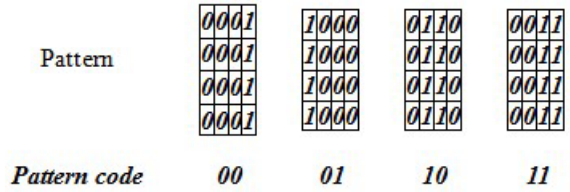

FIGURE 8. Four 4×4-block patterns and their patterns codes

Function Pattern_matching(B) gives the code for these blocks mentioned above.

Function Pattern_matching(B)
  if B is exact one of the 4×4-block patterns then
    return('1'||Pattern code||'0')
  else if B is a 4×4-block pattern except one bit
    return('1'||Pattern code||'1'||k)
  else if the transpose of B is exact one of the 4×4-block patterns then
    return('0'||Pattern code||'0')
  else if the transpose of B is one of the 4×4-block patterns except one bit then
    return('0'||Pattern code||'1'||k)
  /* k is the coordinates of the difference bit */

The second and the fourth rows are compared with the first and the third rows, respectively. The possible situations are processed by *Function Reference_Row*(B):

Function Reference_Row(B)
  if $FR = SR$ and $TR = OR$ then
    return('00'||FR||TR)
  if there is only one different bit between FR and SR, and $TR = OR$ in B then
    return('01'||FR||TR||k)
  if there is only one different bit between FR and SR, and $FR = SR$ in B then
    return('10'||FR||TR||k)
  if $FR = SR$ and $TR = OR$ but there is one different bit between SR and OR
    return('11'||FR||TR||k)
/* FR is first row, SR the second row, TR the third row, OR the fourth row of B *
/* k means that the k-th bit is the different bit */

After the block compressing stage, the compressed image f_0 is transformed into one LQ and a set of BCs, each BC corresponding to a basic block.

2.2. **Decoding phase.** To reconstruct f_0, the left most integer N_1 of LQ is removed and N_1 '1'-bits are appended to the front of the remaining of LQ. The following algorithm Decode_LQ() can be used to transform LQ into image f'. In recovering f_B, each '0'-bit in f' corresponds to 2×2 white pixels in f_B and each of the 2×2 white pixels maps to one 8×8 all-white basic block in f_{Ev}. Each '1'-bit (dangled bits) in f' will be succeeded

by 2×2 pixels in f_B, each of the 2×2 pixels corresponding to one basic block which will be decoded by one block code BC. The $TypeCode$ and $header$ of BC can be used to identify the block type and coding method of a basic block B; then the '1'-bits in B can be decided by the remaining code of BC.

Algorithm Decode _LQ(LQ)
  remove the left most bit b from LQ
  $f'[0,0] = b$
  for $k = 2^0$ to 2^{l-2}
    for $i = 2^0$ to 2^{l-2}
      for $j = 0$ to $l-1$
        if $f'[i,j] = 1$ then
          remove the left most 4 bits $b_1b_2b_3b_4$ from LQ
          $f_b[2i,2j] = b_1$; $f_b[2i,2j+1] = b_2$; $f_b[2i+1,2j] = b_3$; $f_b[2i+1,2j+1] = b_4$;
        else
          $f_b[2i,2j] = f_b[2i,2j+1] = f_b[2i+1,2j] = f_b[2i+1,2j+1] = 0$;
    $f_b = f'$

After that, f_{Ev} can be constructed by using f_b and BCs. To generate f_{Eh} from f_{Ev}, f_{Eh} is scanned, pixel by pixel. For each visited pixel $f_{Ev}(i,j)$, $f_{Eh}(i,j)$ is computed by the following formula:

$$f_{Eh}(i,j) = \begin{cases} f_{Ev}(i,j), & \text{for} \quad i = 0, \\ f_{Ev}(i,j) \oplus f_{Ev}(i-1,j), & \text{for} \quad i \neq 0. \end{cases} \tag{3}$$

Then, each pixel of $f_0(i,j)$ can be obtained by the following formula:

$$f_0(i,j) = \begin{cases} f_{Eh}(i,j), & \text{for} \quad j = 0, \\ f_{Eh}(i,j) \oplus f_{Eh}(i-1,j), & \text{for} \quad j \neq 0. \end{cases} \tag{4}$$

3. **The QLS Hiding-Compression Method.** Due to the constraint on bandwidth, a user often hopes to compress the stego-image first for reducing the data size before transmitting the compressed data over the Internet. The QLS hiding-compression method compresses the cover image f_0 by the QLS compression method and embeds the secret data in the intermediate goods f_{Eh} when encoding the cover image. The major difference between the QLS hiding-compression method and the QLS compression method is that the QLS hiding-compression method embeds the secret data in f_{Eh}. This section therefore only describes the approach.

When a pixel in an all-white area is changed into a black one or a pixel in an all-black area is changed into a white one, the human eyes can easily detect the change. In data hiding, embedding secret data in a cover image usually distorts the cover image. The binary image hiding techniques generally hide secret data near the boundaries of objects in f_0. Hence, the QLS hiding-compression method also attempts to conceal secret data in the boundaries of objects in the cover image. In the Exclusive-OR stage, f_0 is transformed into f_{Eh} and then into f_{Ev}. If the colors of the $(i-1)$-th and the i-th pixels in f_0 are the same, the i-th pixel in f_{Eh} is given a '0'-bit; otherwise, the i-th pixel in f_{Eh} is a '1'-bit. The '1'-bits in f_{Eh} are located near the boundaries of objects in f_0. Therefore, the QLS hiding-compression method embeds secret data in the pixels nearby the '1'-bits in f_{Eh}.

Converting the color of the i-th pixel in f_{Eh} will make the colors of all the pixels be altered after the i-th pixel in the decompressed f_0; while the colors of both the i-th and the j-th pixels in f_{Eh} are changed, only the colors of all the pixels between the i-th pixel and the j-th pixel in f_0 decompressed from f_{Eh} are changed. Based on this property,

the QLS hiding-compression method considers each row in f_{Eh} to be a big binary string and partitions the big binary string into 4-bit units. Each unit containing at least one '1'-bit will be used to carry one secret data bit since the QLS hiding-compression method engages in embedding the secret data near the boundary of objects in the cover image. We call the unit with at least one '1'-bit an embedding unit.

TABLE 4. The replaced 4-bit units in '0'-group and '1'-group

U	0-group	1-group
1111	0011	1111
0011	0011	0101
1100	1010	1100
1001	1001	0101
0101	0011	0101
1010	1010	1100
0110	1010	0110
0111	0111	0001
1101	1101	0001
1110	1000	1110
1011	1000	1011
0001	0010	0001
0010	0010	0001
1000	1000	0100
0100	1000	0100

The QLS hiding-compression method categorizes all the possible 4-bit embedding units into two groups: '0'-group and '1'-group. Each secret bit b corresponds to one embedding unit U. If $b =$ '0' (resp. $b =$ '1'), one embedding unit U' in 0-group (resp. '1'-group) is used to replace U. If the two different bits between U and U' are located more closely to each other, the distortion of the stego image compared to the cover image can be reduced more. Table 4 shows the replaced 4-bit embedding units, where there are exactly 0 or 2 different bits between U and U', and the different bits are more closely located. After that, the QLS compression method continues to encode the f_{Eh} which carries the secret data.

It is easy to transform digital data into binary codes. In this paper, we assume that the secret data SD is a big binary string and $|SD|$ is the size of SD. Before hiding, the QLS compression method first concatenates $|SD|$ and SD into a binary string SD'. To prevent the secret data from unauthorized access, this method uses a private key PK (over 512 bits) as the seed of a random number generator G to generate a big binary string K, where $|K| = |SD'|$. The method then computes $SD'' = K \oplus SD'$ and hides SD'' in the embedding units bit by bit.

To extract the secret data from the stego image, the QLS hiding-compression method transforms the stego image f_{st} into f_{Eh} by Formula (1). The QLS hiding-compression method considers each row in f_{Eh} to be a big binary string and partitions the big binary string into 4-bit units in order. Next, the method takes out each embedding unit U'; one '0'-bit is appended to the rear of SD'' if U' is in '0'-group, and else one '1'-bit is appended to the rear of SD''. The proposed method then uses the same private key PK as the seed of the random number generator G to generate a big binary string K, where $|K| = |SD''|$. Besides, the proposed method computes $SD' = K \oplus SD''$. It removes the leftmost integer of SD'; the removed integer is $|SD|$. The leftmost $|SD|$ bits of SD are the embedded

secret data. The secret data extracted by the QLS hiding-compression method is lossless. Since the secret data are embedded only near the boundaries of objects in the cover image, the stego image is quite similar to the cover image.

4. **Experiments.** This section is to investigate the performances of the QLS compression method and the QLS hiding-compression method by experiments. Eight document images in Figure 9 are used as the test images each with 1728×2376 pixels. There are many big all-black areas or big all-white areas in images CCTTT2 and CCTTT8, while images CCTTT4 and CCTTT7 have fewer big areas consisting of all-black pixels or all-white pixels. In this paper, compression rate Cr is employed to measure the performance of an image compression method, where Cr is defined as:

$$C_r = \frac{\text{The number of bytes needed to store original binary image}}{\text{The number of bytes needed to store compressed binary image}}.$$

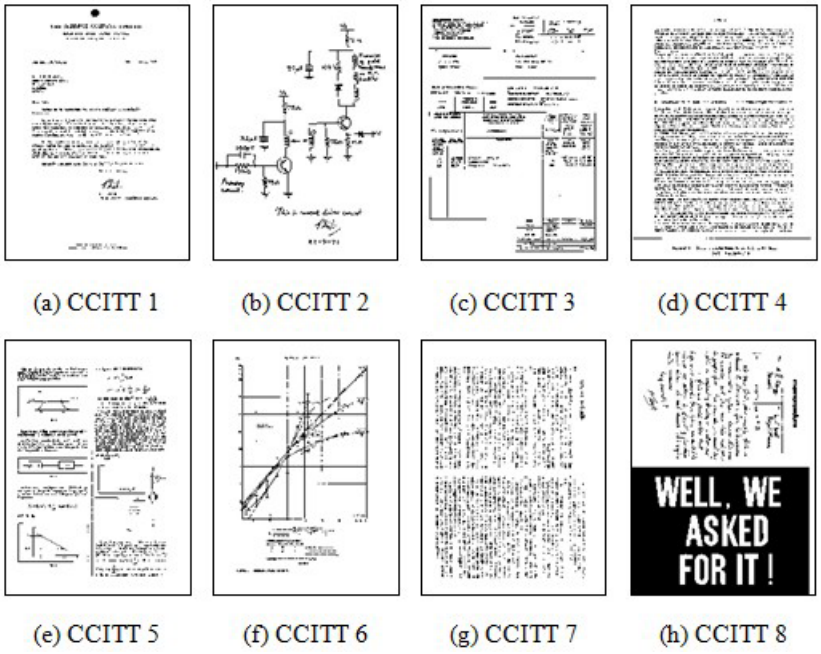

FIGURE 9. CCITT standard images

The purpose of the first experiment is to scrutinize the performance of the QLS compression method and compare it to the compression methods proposed by Wang et al. [35] and Falkowski [12]. Table 5 shows the results of this experiment, where LFS and QSBBIC respectively stand for the methods proposed by Wang et al. [12] and Falkowski [35]. The experimental results tell that the QLS compression method can give a better Cr than the LFS compression method. The QLS compression method is mostly superior to the QSBBIC compression method in Cr, too, except for the images CCTTT2 and CCTTT8; the QSBBIC compression method can provide a better compression rate than the QLS and LFS compression methods only if the compressed images are mainly made of fewer big areas consisting of all-black pixels or all-white pixels.

The next experiment is to explore the performance of the QLS hiding-compression method and compare it to the performance of the pair-wise logical computation (PWLC) hiding method [22]. Table 6 demonstrates the results of this experiment. In this experiment, a random number generator G is applied to generate $MDHC$ bits of secret data and then the QLS hiding-compression method is used to hide the generated $MDHC$-bits

TABLE 5. The Cr of the LFS, QSBBIC and QLS compression methods

Image	LFS	QSBBIC	QLS
CCITT 1	16.3	15.77	**16.56**
CCITT 2	24.40	**28.53**	25.25
CCITT 3	10.90	9.60	**11.09**
CCITT 4	4.30	4.19	**4.34**
CCITT 5	9.60	8.49	**9.69**
CCITT 6	18.60	15.69	**19.11**
CCITT 7	5.00	4.80	**5.05**
CCITT 8	14.60	**15.73**	15.11

TABLE 6. $MDHC$ and SD obtained by the PWLC hiding method and the QLS hiding-compression method

	PWLC		QLS-Hiding	
Images	$MDHC$	SD (%)	$MDHC$	SD (%)
CCITT 1	29431	98.92	42598	99.49
CCITT 2	23916	99.12	25079	99.70
CCITT 3	51322	98.12	71188	99.13
CCITT 4	95560	96.50	165323	97.98
CCITT 5	53141	98.05	81074	99.01
CCITT 6	34544	98.73	47635	99.42
CCITT 7	77279	97.17	133074	98.37
CCITT 8	45733	98.32	49041	99.41

secret data in a $W \times H$ cover data, where $MDHC$ is the maximal number of bits which can be embedded in the cover image by the QLS hiding-compression method. Here, the distortion degree SD is adopted to measure the distortion between the cover image and the stego image. SD can be defined as:

$$SD = \left(\frac{100}{W \times H}\right) \sum_{i=1}^{W} \sum_{j=1}^{H} (1 - (b_{ij} - b'_{ij})^2)$$

where b_{ij} is the color of the pixel located at (i, j) in the cover image, and b'_{ij} the color of the pixel located at (i, j) in the stego image.

Table 6 demonstrates that the QLS hiding-compression method can obtain better $MDHC$ and SD than the PWLC hiding method. Figure 10(a) (resp. Figure 10(b)) is one region on CCITT 1 (resp. CCITT 2), Figure 10(c) (resp. Figure 10(d)) and Figure 10(e) (resp. Figure 10(f)) are its corresponding regions on the stego images respectively generated by the PWLC hiding method and the QLS hiding-compression method. Figure 10 shows that the PWLC hiding method gives a much worse quality of stego image than the QLS hiding-compression method, especially near the boundaries of objects.

5. **Conclusions.** In this paper, the QLS compression method is proposed to encode a binary image. It first uses the BFT linear quadtree to decompose the image into blocks and employs logic functions and spectral technique to encode the blocks. The experimental results show that the QLS compression method is more effective in compression rate than the LFS compression method. In most cases, it gives a better performance than the

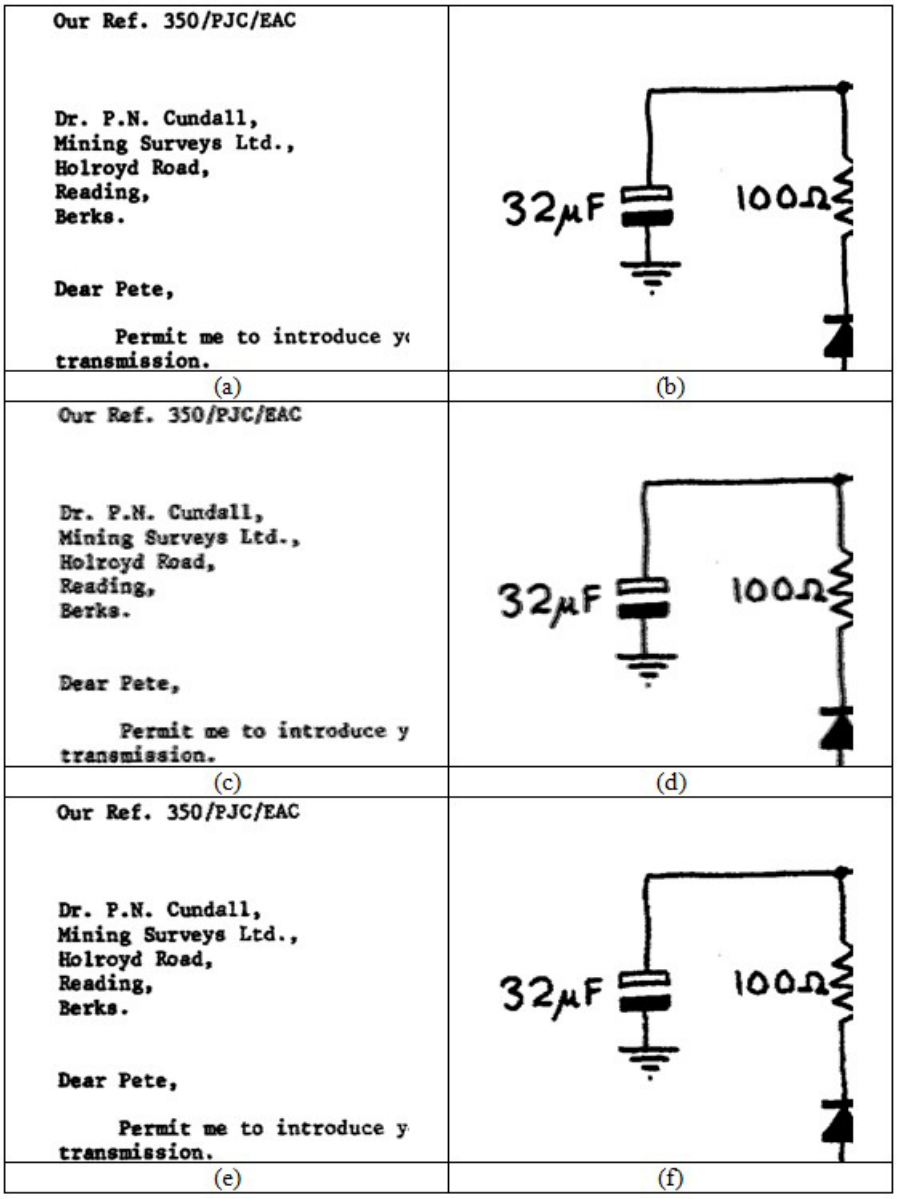

FIGURE 10. Two regions respectively on CCITT 1 as well as CCITT 2 and their corresponding regions on the stego images obtained by the PWLC hiding method and the QLS hiding-compression method

QSBBIC compression method too, especially when the compressed image is mainly made of some all-black and all-white areas.

This paper still provides a QLS hiding-compression method by which secret data are embedded in the cover image while the cover image is being encoded by the QLS compression method. The experimental results demonstrate that the QLS hiding-compression method can give much better $MDHC$ and SD than the PWLC hiding method.

REFERENCES

[1] T. Batu, S. Dasgupta, R. Kumar and R. Rubinfeld, The complexity of approximating entropy, *Proc. of the 17th Annual IEEE Conference on Computational Complexity*, Montreal, Canada, pp.678-687, 2002.

[2] J. Brassil, S. Low, N. Maxemchuk and L. O'Gorman, Electronic marking and identification techniques to discourage document copying, *IEEE Journal on Selected Areas in Communications*, vol.13, no.8, pp.1495-1504, 1995.
[3] Y.-K. Chan and C.-C. Chang, An efficient data structure for storing similar binary images, in *Information Organization and Databases: Foundation of Data Organization*, K. Tanaka and S. Ghandeharizadeh (eds.), Massachusetts, Kluwer Aeademic, 2001.
[4] C. K. Chan and L. M. Cheng, Hiding data in images by simple LSB substitution, *Pattern Recognition*, vol.37, no.3, pp.469-474, 2004.
[5] Y.-K. Chan, P.-Y. Pai, R.-C. Chen and C.-C. Chang, A VQ compression method based on the variations of the image block groups, *International Journal of Innovative Computing, Information and Control*, vol.6, no.10, pp.4527-4537, 2010.
[6] C.-C. Chang, T. S. Chen and L. Z. Chung, A steganographic method based upon JPEG and quantization table modification, *Information Sciences*, vol.141, no.1, pp.123-138, 2002.
[7] C.-C. Chang, K.-N. Chen and Z.-H. Wang, Hiding secret information in modified locally adaptive data compression code, *ICIC Express Letters*, vol.4, no.5(B), pp.1887-1892, 2010.
[8] A. K. Chaudhary, J. Augustine and J. Jacob, Lossless compression of image using logic minimization, *Proc. of IEEE International Conference on Image Processing*, Lausanne, Switzerland, pp.77-80, 1996.
[9] M. Davio, J. P. Deschamps and A. Thayse, *Discrete and Switching Functions*, McGraw-Hill, New York, USA, 1978.
[10] I. Echizen, Y. Suzuki and X. Niu, Special issue on information hiding and multimedia signal processing, *International Journal of Innovative Computing, Information and Control*, vol.6, no.3(B), pp.1207-1208, 2010.
[11] B. J. Falkowski, Lossless compression of binary images using logic methods, *Proc. of the South Eastern Europe Workshop on Computational Intelligence and Information Technologies*, Nis, Yugoslavia, pp.111-116, 2001.
[12] B. J. Falkowski, Lossless binary image compression using logic functions and spectra, *Computers and Electrical Engineering*, vol.30, no.1, pp.17-43, 2004.
[13] B. J. Falkowski and C. H. Chang, Hadamard-Walsh spectral characterization of Reed-Muller expansions, *Computers and Electrical Engineering*, vol.25, no.2, pp.111-134, 1999.
[14] B. J. Falkowski and L. S. Lim, Gray scale image compression based on multiple-valued input binary functions, Walsh and Reed-Muller spectra, *Proc. of the 30th International Symposium on Multiple-Valued Logic*, Portland, Oregon, pp.279-284, 2000.
[15] D. H. Green, *Modern Logic Design*, Addison-Wesley, MA, USA, 1986.
[16] S. Hanan, Data structures for quadtree approximation and compression, *Communications of the ACM*, vol.28, no.9, pp.973-993, 1985.
[17] D.-C. Huang, Y.-K. Chan and J.-H. Wu, An agent-based LSB substitution image hiding method, *International Journal of Innovative Computing, Information and Control*, vol.6, no.3(A), pp.1023-1038, 2010.
[18] N. F. Johnson and S. Jajodia, Steganography: Seeing the unseen, *IEEE Computer*, pp.26-34, 1998.
[19] S. L. Li, K. C. Leung, L. M. Cheng and C. K. Chan, A novel image-hiding scheme based on block difference, *Pattern Recognition*, vol.39, no.6, pp.1168-1176, 2006.
[20] T. W. Lin, Compressed quadtree representations for storing similar images, *Image and Vision Computing*, vol.15, no.11, pp.833-843, 1997.
[21] S. H. Low, N. F. Maxemchuk, J. T. Brassil and L. O'Gorman, Document marking and identification using both line and word shifting, *Proc. of Infocom, Boston*, MA, USA, pp.853-860, 1995.
[22] K. Matsui and K. Tanaka, Video-steganography: How to secretly embed a signature in a picture, *Proc. of IMA Intellectual Property Project*, vol.1, no.1, pp.187-206, 1994.
[23] P. Mateu-Villarroya and J. Prades-Nebot, Lossless image compression using ordered binary-decision diagrams, *Electronics Letters*, vol.37, no.3, pp.162-163, 2001.
[24] N. F. Maxemchuk and S. Low, Marking text documents, *Proc. of the 1997 International Conference on Image Processing*, vol.3, 1997.
[25] M. Nelson and J. L. Gailly, *The Data Compression Book*, 2nd Edition, M&T Books, New York, USA, 1996.
[26] A. Nishimura, Audio data hiding that is robust with respect to aerial transmission and speech codecs, *International Journal of Innovative Computing, Information and Control*, vol.6, no.3(B), pp.1389-1400, 2010.

[27] G. Pandurangan and E. Upfal, Can entropy characterize performance of online algorithms, *Proc. of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms*, Washington D.C., USA, pp.727-734, 2001.

[28] P.-Y. Pai, C.-C. Chang, Y.-K. Chan and C.-C. Liao, Meaningful shadow based multiple gray level visual cryptography without size expansion, *International Journal of Innovative Computing, Information and Control*, vol.7, no.3, pp.1457-1465, 2011.

[29] G. R. Robertson, M. F. Aburdene and R. J. Kozick, Differential block coding of bi-level images, *IEEE Transactions on Image Processing*, vol.5, no.9, pp.1368-1370, 1996.

[30] C. E. Shannon, A mathematical theory of communication, *The Bell System Technical Journal*, vol.27, no.3, pp.379-423, 1948.

[31] C. C. Thien and J. C. Lin, A simple and high-hiding capacity method for hiding digit-by-digit data in images based on modulus function, *Pattern Recognition*, vol.36, no.12, pp.2875-2881, 2003.

[32] P. Tsai, Data hiding for index-based images using overlapping codeword clustering and run length concept, *International Journal of Innovative Computing, Information and Control*, vol.6, no.6, pp.2701-2713, 2010.

[33] C. L. Tsai, K. C. Fan, C. D. Chung and T. C. Chuang, Reversible data hiding and lossless reconstruction of binary images using pair-wise logical computation mechanism, *Pattern Recognition*, vol.38, no.11, pp.1993-2006, 2005.

[34] C.-C. Wang, Y.-T. Hwang, C.-C. Chang and J.-K. Jan, Hiding information binary images with complete reversibility and high embedding capacity, *International Journal of Innovative Computing, Information and Control*, vol.6, no.11, pp.5143-5161, 2010.

[35] J. Wang and L. Ji, A region and data hiding based error concealment scheme for images, *IEEE Transformations on Consumer Electronics*, vol.47, no.2, pp.257-262, 2001.

[36] C. L. Wang, S. C. Wu, Y. K. Chan and R. F. Chang, Quadtree and statistical model-based lossless binary image compression method, *Imaging Science Journal*, vol.53, no.2, pp.95-103, 2005.

[37] M. Wu and B. Liu, Data hiding in binary image for authentication and annotation, *IEEE Transactions on Multimedia*, vol.6, no.4, pp.528-538, 2004.

[38] *CCITT Standard Fax Images at ftp://nic.funet.fi/pub/graphics/misc/test-images/.*