JOURNAL OF VISUAL COMMUNICATION AND IMAGE REPRESENTATION
Vol. 8, No. 3, September, pp. 291–298, 1997
ARTICLE NO. VC970360

# A New Binary Image Representation: Logicodes

Jung-Gen Wu

*Department of Information and Computer Education, National Taiwan Normal University, No. 162, Section 1, Hoping E. Road, Taipei, Taiwan 10610, Republic of China*

and

Kuo-Liang Chung*

*Department of Information Management and Graduate Program of Information Engineering, National Taiwan University of Science Technology, No. 43, Section 4, Keelung Road, Taipei, Taiwan 10672, Republic of China*

Received August 21, 1996; accepted August 14, 1997

**Using bincodes to represent binary images is shown to be very simple and storage-saving. Given a set of bincodes, this paper presents two improved codes, namely, the logicodes and the restricted logicodes to represent binary images. We first transform the given bincodes into a set of logical expressions. Then a minimization technique is employed to reduce the storage space required for these logical expressions, thus obtaining the logicodes, on which set operations can be applied directly. Further, we put some restrictions into these logicodes to make each resulting logicode, called the restricted logicode, representing a connected black block. Given 20 different-type real images, experimental results show that our logicodes (restricted logicodes) present a saving of 29% to 44% (12% to 34%) with respect to bincodes. When compared to Sarkar's method, except spending a little more space, our proposed codes do have three advantages: (1) it is easier to extract the related geometrical coordinates; (2) the bincodes can be used as direct input; i.e., they do compress the bincodes further; and (3) each restricted logicode represents a connected block block.** 

*Key Words:* **spatial data structures; binary image compression; bincodes; logical expressions; minimization; logicodes; set operations; Sarkar's codes.**

## 1. INTRODUCTION

Designing efficient image representations [17, 18] is an important issue in pattern recognition, image processing, computer graphics, computer vision, etc. Efficient image representations may save space and facilitate the manipulation of the acquired images. Researchers use strings [6, 23, 12, 1, 2], trees [5], and sets of codes [7, 15, 21, 19, 20] to represent digital binary images, and many efficient algorithms on these spatial data structures have been developed.

Bincode is a new method to compress binary images [15]. Each bincode represents a black rectangular subimage of the input image; the code is formed by interleaving the binary representations of the $x$- and $y$-coordinates of the subimage with its level in the corresponding bintree. It has 0 to 25% space utilization improvement over the linear quadtree coding method in empirical comparisons [22]. Some fact image algorithms on bincodes [9, 10, 3] have also been developed. In [19], the input is a rectangular binary $2^m \times 2^n$ image array and each pixel is labelled with an $m$- and an $n$-bit Gray code. Then, each pixel position of the binary image is represented by a logical expression of $m + n$ variables. Sarkar applies the Quine–McCluskey method to minimize the Boolean function resulting from the encoding of the original binary image. Experimental results show that Sarkar's encoding method is better than both the bincodes representation and the linear quadtree representation. Based on Sarkar's codes, some efficient operations, such as intersection, union, complement, and area calculation, have been developed [20]. In centroid computation, it needs some effort to extract the geometrical coordinates.

In one of our previous works [24], a bincode is interpreted as a logical expression based on the $x$- and $y$- coordinates of the corresponding subimage; this interpretation allows operations on bincodes to be performed using logical operators [24, 4]. This paper presents two improved codes, namely, logicodes and restricted logicodes for the handling of binary images. We first transform input bin-

* Corresponding author. E-mail: klchung@cs.ntust.edu.tw.



1047-3203/97 $25.00

codes into a set of logical expressions. Then a minimization technique reduces the storage required for these logical expressions, thus obtaining the logicodes, on which set operations can be applied directly. Further, we introduce some restrictions into these logicodes to make each resulting logicode, called the restricted logicode, represent a connected black block. The restricted logicodes are more suitable for some operations, such as neighbor finding [17, 18], which is the kernel of connected component labelling, than logicodes. This issue is beyond the scope of the paper. Since a DF- (depth first-) expression can be transformed into bincodes [9] in linear time, a DF-expression can also be transformed into our logicodes in linear time. Given 20 different-type real images, experimental results show that our logicodes (restricted logicodes) present a 29% to 44% (12% to 34%) space improvement over bincodes. When compared to Sarkar's codes, except for spending a little more space, our proposed codes have the three advantages: (1) it is easier to extract the related geometrical coordinates when compared to the Gray code-based approach [19]; (2) the bincodes can be used as direct input; and (3) each restricted logicode represents a connected black block.

The rest of this paper is organized as follows. Section 2 introduces the logical interpretation for bincodes. Section 3 illustrates our new representations. Section 4 illustrates some experimental results on 20 real images. Section 5 gives the conclusions and outlines our future works.

## 2. LOGICAL INTERPRETATION FOR BINCODES

Assume that the given image is represented to the positive quadrant of a Cartesian coordinate system. The position of each pixel is represented by its $x$- and $y$-coordinates, i.e. a pair of positive numbers. The bincodes of the image can be derived from its bintree representation. To represent a binary image by using a bintree, we recursively divide the image, if it is not either totally black or totally white, into two equal subimages in the $x$-direction and the $y$-direction alternatively. Initially, the original image is represented by the root node. Two son-nodes are added to represent the two equal-size subimages divided in the $x$-direction. If any subimage is either black or white, the dividing process terminates on that part. This process is performed recursively until all subimages are either totally black or totally white. As a result, a bintree is obtained. The nodes at odd-numbered levels of the bintree represent the subimages subdivided in the $x$-direction; the nodes at even-numbered levels of the bintree represent the subimages subdivided in the $x$-direction; the nodes at even-numbered levels of the bintree represent the subimages subdivided in the $y$-direction. Figure 1a shows a $2^2 \times 2^2$ binary image. Its bintree representation is shown in Fig. 1b.

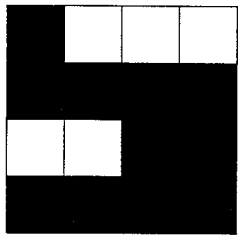
The bincode is based on the bintree structure and represents the bintree as an ordered collection of black nodes. Given a $2^m \times 2^m$ binary image, each black node at level $l$ in the bintree with left-bottom corner $(x, y)$ is encoded as $\sum_{k=0}^{m-1} (x_k \times 2^{4k+3}) + \sum_{k=0}^{m-1} (y_k \times 2^{4k+1}) + \sum_{k=0}^{2m-1} (s_k \times 2^{2k})$, where $x_{m-1}x_{m-2} \ldots . x_0$ and $y_{m-1}y_{m-2} \ldots . y_0$ denote the binary representations of $x$ and $y$, respectively; $s = 2^{2m} - 2^{2m-l} = (s_{2^m-1}s_{2m-2} \ldots . s_0)_2$. In fact, the encoded bincode with $4m$ bits for that black node is $(x_{m-1}s_{2m-1}y_{m-1}s_{2m-2}$ $x_{m-2}s_{2m-3} \ldots x_0s_1y_0s_0)_2$.

The black nodes of Fig. 1b are labeled by their corresponding bincodes as shown in Fig. 1c, where each bincode is represented by a 3-tuple ($x$-coordinate, $y$-coordinate, $s$), a binary number, a decimal number, and a string of logical symbols described below. Here, the listed bincodes of Fig. 1c are not dependent of any specific traversal order. In [9, 10, 3] they are listed by preorder.

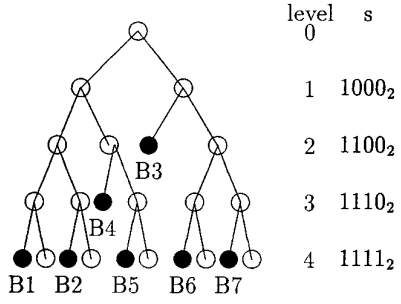
Observing the process in building the bintree, we first divide the original image into two equal-sized subimages in the $x$-direction. All the pixels in the left (right) subimage have their $x_{m-1}$ to be 0 (1); the order coordinate-variables $x_{m-2}, \ldots, x_0, y_{m-1}, y_{m-2}, \ldots$, and $y_0$ are viewed as "don't-care" symbols. Logically, we denote the left (right) part as $\overline{X}_{m-1}$ ($X_{m-1}$). In the next step, we divide the two subimages in the $y$-direction. Four generated subimages, namely, the left-lower part, left-upper part, right-lower part, and right-upper part, are denoted by $\overline{X}_{m-1}\overline{Y}_{m-1}$, $\overline{X}_{m-1}Y_{m-1}$, $X_{m-1}\overline{Y}_{m-1}$, and $X_{m-1}Y_{m-1}$, respectively. As shown in [24], each pair $x_is_{2i+1}$ ($y_is_{2i}$), $0 \le i \le m-1$, can be represented by a boolean function of a binary variable $X_i$ ($Y_i$). That is, a bincode is represented by the logical expression:

$$f_{m-1}(X_{m-1})g_{m-1}(Y_{m-1})f_{m-2}(X_{m-2})g_{m-2}(Y_{m-2}) \ldots$$
$$f_1(X_1)g_1(Y_1)f_0(X_0)g_0(Y_0),$$

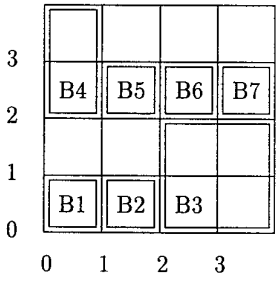
where each function $f_i(\,)$ ($g_i(\,)$) has three possible outputs, namely, $X_i$ ($Y_i$), $\overline{X}_i$ ($\overline{Y}_i$), and $\delta$ = "don't care." Therefore, each of the functions $f_i(X_i)$ and $g_i(Y_i)$, $0 \le i \le m-1$, can be represented by a 2-bit value. To make a direct correspondence with the encoding previously introduced, we choose 01 to represent $\overline{X}_i$ and $\overline{Y}_i$; 11 to represent $X_i$ and $Y_i$; 00 to represent $\delta$. The 2-bit number corresponding to $f_i(X_i)$ is located at positions $4i + 3$ and $4i + 2$ in the encoded bincode, and that corresponding to $g_i(Y_i)$ is located at positions $4i + 1$ and $4i$. For example, in Fig. 1c, the encoded bincode for block $B1$ is represented by the binary string 01010101 and its logical product expression is $\overline{X}_1\overline{Y}_1\overline{X}_0\overline{Y}_0$. The leftmost 2 bits, 01, denote the $\overline{X}_1$; the next 2 bits, 01, at positions 5 and 4 denote the $\overline{Y}_1$; the 2 bits, 01, at positions 3 and 2 denote the $\overline{X}_0$; and the 2 bits, 01, at positions 1 and 0 denote the $\overline{Y}_0$. By the same argument, the encoded bincode for block $B3$, represented by 11010000, is $X_1\overline{Y}_1 = X_1\overline{Y}_1\delta\delta$. In this case, the memory required is reduced from 8 to 4 bits since the "don't-care" symbols are discarded.



(a) A binary image.




(b) Bintree.




$B1=\{00_2, 00_2, 1111_2\} = 01010101_2 = 85 = \bar{X}_1\bar{Y}_1\bar{X}_0\bar{Y}_0$
$B2=\{01_2, 00_2, 1111_2\} = 01011101_2 = 93 = \bar{X}_1\bar{Y}_1X_0\bar{Y}_0$
$B3=\{10_2, 00_2, 1100_2\} = 11010000_2 = 208 = X_1\bar{Y}_1$
$B4=\{00_2, 10_2, 1110_2\} = 01110100_2 = 116 = \bar{X}_1Y_1\bar{X}_0$
$B5=\{01_2, 10_2, 1111_2\} = 01111101_2 = 125 = \bar{X}_1Y_1X_0\bar{Y}_0$
$B6=\{10_2, 10_2, 1111_2\} = 11110101_2 = 245 = X_1Y_1\bar{X}_0\bar{Y}_0$
$B7=\{11_2, 10_2, 1111_2\} = 11111101_2 = 253 = X_1Y_1X_0\bar{Y}_0$

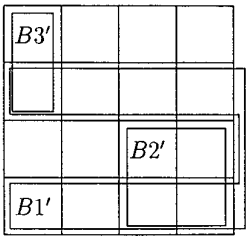
(c) Blocks represented by bincodes.

**FIG. 1.** A $2^2 \times 2^2$ binary image.

Using a logical expression to represent each bincode, the whole image can be represented by ORing each expression. As a result, the image in Fig. 1 can be a sum-of-products (SOP) represented by the logical expression $\overline{X}_1\overline{Y}_1\overline{X}_0\overline{Y}_0 + \overline{X}_1\overline{Y}_1X_0\overline{Y}_0 + X_1\overline{Y}_1 + \overline{X}_1Y_1\overline{X}_0 + \overline{X}_1Y_1X_0\overline{Y}_0 + X_1Y_1\overline{X}_0\overline{Y}_0 + X_1Y_1X_0\overline{Y}_0$. However, in order to distinguish two consecutive logical products, we put one $\delta$ symbol at the end of a logical product. For example, to store the bincodes in Fig. 1, we need only store 110100 for block $B3$. Since the size of the image is $4 \times 4$, each bincode has eight bits. All other blocks in Fig. 1 need eight bits to be represented, the total being 54 bits. In the worst case, the proposed SOP logical expression uses fewer bits when compared to that of bincodes representation. It is also easy to recover the original binary image from an SOP form.

## 3. NEW METHODS

Based on the prceding SOP logical expression, this section presents logicodes and restricted logicodes to represent binary images and shows that set operations on images can be performed by applying logical operations directly on these codes. First, a minimization technique is employed to reduce the storage required by the preceding SOP logical expression. Then, each product term is represented by a number, named a "logicode." However, a representation as a logicode may represent more than one disjoint black block, for instance, not appropriate in connected-component labeling [17, 18], we also present a "restricted logicode" which does not represent a minimized logical expression, but only a single black block, maximal by inclusion.




$B1' = \overline{Y}_0 = 00000001_2 = 1$
$B2' = X_1\overline{Y}_1 = 11010000_2 = 208$
$B3' = \overline{X}_1 Y_1 \overline{X}_0 = 01110100_2 = 116$

**FIG. 2.** Blocks represented by logicodes.

### 3.1. *Logicode*

The basic concept of the proposed logicodes is to minimize the preceding SOP logical expression. Since an SOP is the logical representation of bincodes, the logicodes do compress the bincodes further. In Sarkar's codes, the input is the original images, each pixel in the image encoded by a Gray code-based code. In addition, the logicodes are easier to extract the related geometrical coordinates because the binary representations of $x$ and $y$, where $(x, y)$ is the geometrical location of the black block, is geometrically preserved in each product of the SOP. While in Sarkar's Gray code-based method, it is harder to extract the related geometrical coordinates [20].

Consider, for instance, the example in Fig. 1. *OR* the logical expressions of $B1$ and $B2$ to get $\overline{X}_1\overline{Y}_1\overline{Y}_0$; simplify it by combining it with the lower half of $B3$, thus obtaining $\overline{Y}_1\overline{Y}_0$. $B1$, $B2$, $B5$, $B6$, $B7$, the lower half of $B2$, and the lower half of $B4$ can be combined in the logical expression $\overline{Y}_0$. By using this scheme, the image of Fig. 1 can be represented by $\overline{Y}_0 + X_1\overline{Y}_1 + \overline{X}_1 Y_1 \overline{X}_0$ (see Fig. 2). It is rather straightforward to convert the above logicodes into the bincodes in a reverse direction.

We use the same logical interpretation in the bincode to encode each logicode. Each logical function of $X_i$ or $Y_i$ in one product denotes one of the following states: (1) the logical variable itself ($X_i$ or $Y_i$); (2) the complement of the variable ($\overline{X}_i$ or $\overline{Y}_i$); and (3) a "don't-care" symbol. Two bits are used to represent the logical function $f_i$ ($g_i$) of a binary variable $X_i$ ($Y_i$). That is, we use at most a $4m$-bit number $(a_{4m-1}a_{4m-2}\ldots a_1a_0)$ to represent a logicode, where

$$\begin{aligned}
a_{4i+3}a_{4i+2} &= 01, \quad \text{if } f_i(X_i) = \overline{X}_i,\\
&= 11, \quad \text{if } f_i(X_i) = X_i,\\
&= 00, \quad \text{if } f_i(X_i) = \delta;\\
a_{4i+1}a_{4i} &= 01, \quad \text{if } g_i(Y_i) = \overline{Y}_i,\\
&= 11, \quad \text{if } g_i(Y_i) = Y_i,\\
&= 00, \quad \text{if } g_i(Y_i) = \delta.
\end{aligned}$$

For example, in Fig. 2, block $B1' = \overline{Y}_0 = \delta\delta\delta\overline{Y}_0$ is represented by the binary number 00000001. The leftmost 2 bits, 00, denote that $X_1$ is a "don't-care" symbol; the next 2 bits, 00, denote that $Y_1$ is a "don't-care" symbol; the 2 bits, 00, at positions 3 and 2 denote that $X_0$ is a "don't-care" symbol; the 2 bits, 01, at positions 1 and 0 denote the $\overline{Y}_0$ symbol.

Actually, the two-bit representation allows a fourth state. In [4, 24], the fourth state is used to denote the invalid logical function of the variable which may occur, e.g., in the operations without valid results.

A logicode is composed of two parts, namely, the $X$-component and the $Y$-component, as follows:

$$\begin{aligned}
X\text{-component} &= f_{m-1}(X_{m-1})f_{m-2}(X_{m-2})\cdots f_1(X_1)f_0(X_0),\\
Y\text{-component} &= g_{m-1}(Y_{m-1})g_{m-2}(Y_{m-2})\cdots g_1(Y_1)g_0(Y_0),
\end{aligned}$$

where the functions $f_i$ and $g_i$, $0 \le i \le m-1$, have been defined previously. To store a logicode, we can store the $X$-component and $Y$-component separately and use only one $\delta$ to represent a sequence of trailing $\delta$'s. By using this representation scheme, block $B1'$ of Fig. 2 is stored as $\delta\delta\overline{Y}_0$ (= 000001). The first $\delta$ denotes the $X$-component and the subsequent $\delta\overline{Y}_0$ denotes the $Y$-component. By the same argument, block $B2'$ is stored as $X_1\delta\overline{Y}_1\delta$ and $B3'$ is stored as $\overline{X}_1\overline{X}_0Y_1\delta$. The whole image needs only 22 (= 6 + 8 + 8) bits.

To find the intersection, the union of two images, and the complement of an image with respect to its background (i.e., set of white elements), we apply logical operations on these logicodes. We use the logicodes in Fig. 2 and Fig. 3 as the inputs to demonstrate how the set operations work. AND the logicodes of the two images to obtain

$$\begin{aligned}
&(\overline{Y}_0 + X_1\overline{Y}_1 + \overline{X}_1Y_1\overline{X}_0)\\
&\cdot (X_1\overline{X}_0 + \overline{Y}_1Y_0 + \overline{Y}_1\overline{X}_0 + \overline{X}_1Y_1X_0)\\
&= X_1\overline{X}_0\overline{Y}_0 + \overline{Y}_1\overline{X}_0\overline{Y}_0 + \overline{X}_1Y_1X_0\overline{Y}_0 + X_1\overline{Y}_1\overline{X}_0 + X_1\overline{Y}_1Y_0.
\end{aligned}$$

The resulting image is shown in Fig. 4. Instead of applying the minimization technique, only basic logical operations are applied. Looking at this figure, we see that the term $X_1\overline{Y}_1\overline{X}_0$ for $A4$ is redundant. The redundant term may be removed if we apply the logical minimization method to the resulting logical expression. OR now the above two sets of logicodes to obtain

$$\begin{aligned}
&(\overline{Y}_0 + X_1\overline{Y}_1 + \overline{X}_1Y_1\overline{X}_0)\\
&+ (X_1\overline{X}_0 + \overline{Y}_1Y_0 + \overline{Y}_1\overline{X}_0 + \overline{X}_1Y_1X_0)\\
&= (\overline{Y}_0 + \overline{Y}_1Y_0) + (\overline{X}_1Y_1\overline{X}_0 + \overline{X}_1Y_1X_0) + X_1\overline{Y}_1\\
&+ X_1\overline{X}_0 + \overline{Y}_1\overline{X}_0
\end{aligned}$$

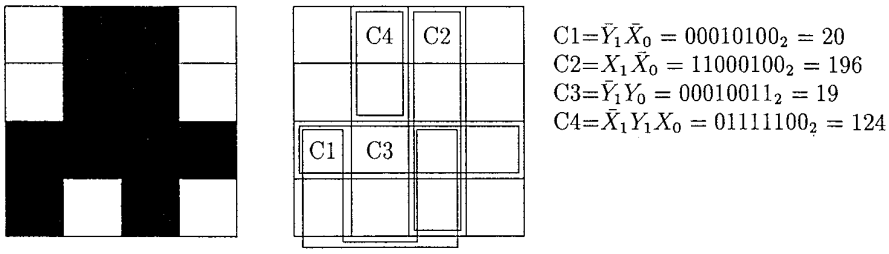



**FIG. 3.** The second $2^2 \times 2^2$ binary image.

$$
\begin{aligned}
&= \overline{Y}_0 + (\overline{Y}_1 + \overline{X}_1 Y_1) + (\overline{Y}_1 + X_1\overline{Y}_1) + X_1\overline{X}_0 \\
&\quad + (\overline{Y}_1 + \overline{Y}_1\overline{X}_0) \\
&= \overline{Y}_0 + \overline{Y}_1 + (\overline{X}_1 + X_1\overline{X}_0) \\
&= \overline{Y}_0 + \overline{Y}_1 + \overline{X}_1 + \overline{X}_0.
\end{aligned}
$$

The resulting image is shown in Fig. 5.

To find the complement of an image, we can use De Morgan's law. For example, the complement of the logicodes in Fig. 1 is

$$
\begin{aligned}
&\overline{(\overline{Y}_0 + X_1\overline{Y}_1 + \overline{X}_1 Y_1\overline{X}_0)} \\
&= Y_0 \cdot (\overline{X}_1 + Y_1) \cdot (X_1 + \overline{Y}_1 + X_0) \\
&= X_1Y_1Y_0 + \overline{X}_1\overline{Y}_1Y_0 + \overline{X}_1X_0Y_0 + Y_1X_0Y_0 \\
&= N1 + N2 + N3 + N4.
\end{aligned}
$$

The resulting image is shown in Fig. 6. The redundant term $\overline{X}_1X_0Y_0$ for $N3$ may be removed if the minimization technique is employed.

### 3.2. *Restricted Logicode*

Logicodes, derived from minimizing bincodes, have better space utilization when compared to the bincode representation. They are also efficient in performing some set operations as described in the preceding subsection. However, since a logicode may represent some disjoint black blocks, it is difficult to use it in some applications such as connected component labelling. Thus, we impose some restrictions to make each of the resulting code to represent a connected black block.

A logicode represents two disjoint black blocks separated by $x$ ($y$) axis, if $X_j \neq \delta$ ($Y_j \neq \delta$), $0 \leq j < m - 1$, is the least significant variable in the $X$-component ($Y$-component) which is not a $\delta$, and there is only one $\delta$ at position $i$ for $i > j$. If there are two such $i$'s (either in the $X$-component or in the $Y$-component), the logicode represents four disjoint black blocks. In general, if there are $k \geq 0$ such $i$'s (either in the $X$-component or in the $Y$-component), the logicode represents $2^k$ disjoint black blocks. For example, in Fig. 2, block $B_1'$ has two disjoint black blocks separated by the $y$ axis. Its $Y_1$ is *don't-care*, but $Y_0$ is not.

If we restrict each logicode to have $\delta$'s only at its least significant variables in both the $X$-component and the $Y$-component, the logicode represents one connected rectangular black block. As shown in Fig. 7, the $B_1'$ of Fig. 2 is subdivided into two subimages represented by the restricted logicodes $Y_1\overline{Y}_0$ and $\overline{Y}_1\overline{Y}_0$. By using the same encoding as before, $B1''$ is stored as $\delta\overline{Y}_1\overline{Y}_0$; $B2'$ is stored as $X_1\delta\overline{Y}_1\delta$; $B3'$ is stored as $\overline{X}_1\overline{X}_0Y_1\delta$; $B4'$ is stored as $\delta Y_1\overline{Y}_0$. The whole image needs 28 bits to be represented.

## 4. EXPERIMENTAL RESULTS

In the previous section, we have presented logicodes and restricted logicodes. In this section, we evaluate the performance of bincodes, logicodes, restricted logicodes,

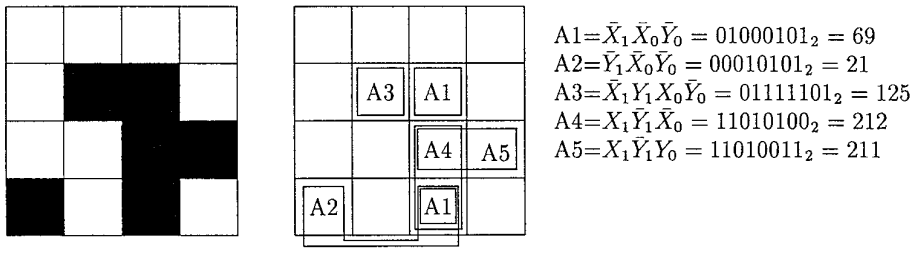



**FIG. 4.** The result of ANDing two logicodes in Fig. 1 and Fig. 3.

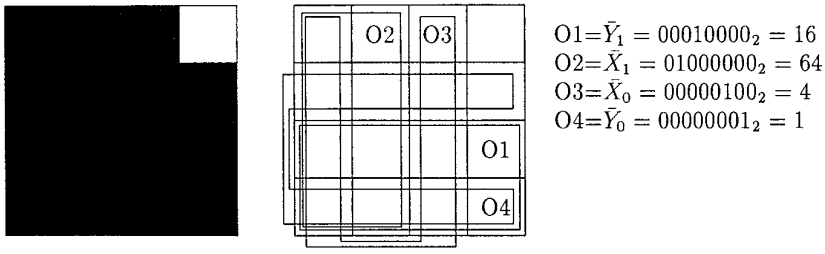



**FIG. 5.** The result of ORing two logicodes.

and Sarkar's codes on 20 real, binary images, each having 256 pixels × 256 pixels, thus requiring 8K bytes of memory. Finding the logicodes results in a minimization procedure involving 16 binary variables. As the number of variables is $n$, the number of prime implicants of one class of these logical functions is proportional to $3^n/n$ [14]. Since the search space is huge, we adopt a heuristic approach [8] to solve the minimization problem, well aware that our results may not represent an optimal solution. Even so, a substantial space utilization is achieved. We get the number of Sarkar's codes by transforming the geometrical coordinates into Gray codes first and then applying our minimization program.

There is an extreme case, the checkerboard pattern with size 256 × 256. It needs 32,768 codes for a bincode, a restricted logicode, and a Sarkar's representation, but only 2 codes for a logicode representation. This extreme case is not included in our experiments. Among these 20 sample images, texts, pictures, and a mixture of texts and pictures are included. Figure 8 illustrates one of the sample images. It needs 5311 bincodes, 2590 logicodes, 3210 restricted logicodes, or 2469 Sarkar's codes to be represented. Since each code needs 16 ternary digits, totally they require 84976, 400720, 51360, and 39504 ternary digits, respectively. It has been described in Section 3.1 that each logical function of $X_i$ or $Y_i$ in one product denotes one of the following three states: (1) the logical variable itself ($X_i$ or $Y_i$); (2) the complement of the variable ($\overline{X}_i$ or $\overline{Y}_i$); and (3) a "don't-care" symbol. Although each state is represented by two bits in a logicode in the preceding section, in fact each state can be represented by a ternary digit since the stored codes are really valid. That is, it needs $\log_2 3 = 1.585$ bits to represent a state. One product in the SOP is now represented by a short sequence of ternary digits. As a result, the SOP is still represented by a long sequence of ternary digits. Then, we transform the sequence of ternary digits into binary numbers and store the binary sequence in the secondary storage. This improved representation presents a saving of $20.77\% = 1 - (\log_2 3)/2$ with respect to the previous representation. This ternary representation is similar to the one in [20], but we discard all the trailing "don't-care" symbols, except the first one, in each product. By using this storage saving scheme, 84256, 40648, and 50420 ternary digits are needed for bincodes, logicodes, and restricted logicodes, respectively, to represent Fig. 8.

Among the sample images, four are pure text images, seven are line-drawn figure images, three are picture images, and six are combinations of text and figures. Table 1 illustrates the total codes required for those representations and the related performance. For the four text images, they need 19,945 bincodes; 11,759 logicodes; 14,497 restricted logicodes; 11,294 Sarkar's codes. The ratios denote the number of the logicodes, restricted logicodes, and Sarkar's codes, respectively, over the number of bincodes, and the corresponding ratios are 59%, 73%, and 57%. It is shown that the regular structure of text images and line-drawn figures offers more opportunity to be minimized. Totally, for these 20 sample images, the ratios are 60%, 72%, and 57%. Logicodes perform 40% better than the bincodes; the restricted logicode representation has 28% space utilization improvement over the bincode representation.

Table 2 illustrates the number of ternary digits required

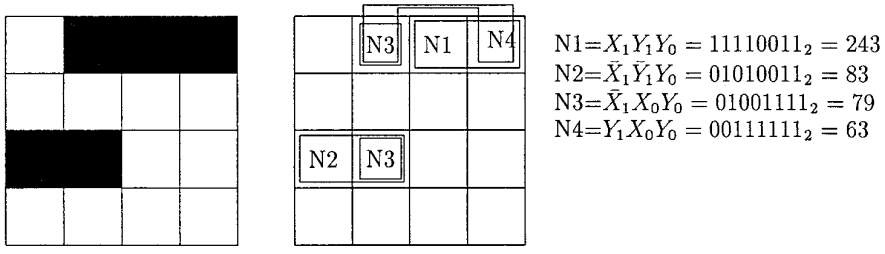
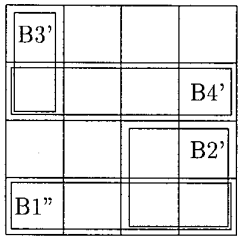



**FIG. 6.** The complement of logicodes in Fig. 1.




$B1''=\bar{Y}_1\bar{Y}_0 = 00010001_2 = 17$
$B2'=X_1\bar{Y}_1 = 11010000_2 = 208$
$B3'=\bar{X}_1Y_1\bar{X}_0 = 01110100_2 = 116$
$B4'=Y_1\bar{Y}_0 = 00110001_2 = 49$

**FIG. 7.** Restricted logicodes.

for bincodes, logicodes, restricted logicodes, and Sarkar's codes, where the ratios denote the number of bincodes, restricted logicodes, and Sarkar's codes, respectively, over the number of logicodes, and the corresponding ratios are 166%, 119%, and 95%.

According to Tables 1 and 2, the experimental results show that the Gray code-based method may have about 5% space saving when compared to the logicodes, but the Sarkar's codes need additional code conversion steps in some geometrical operations such as the centroid calculation [20]. In some special cases such as the image consisting of a mass of checkerboard patterns mentioned above, e.g., the cloth with mesh-texture, the logicodes have better space saving. In addition, one product in Sarkar's codes may represent a set of some disjoint blocks. Except for spending a little more space, our proposed codes do have three advantages mentioned in the next section again.

## 5. CONCLUSIONS

We presented two new compression methods for binary images. Experimental results show that our two new methods have better space utilization when compared to the well-known bincode representation. Logical operations can be applied to perform set operations on the proposed representations. In [24, 4], we showed that using logical expressions to represent binary images facilitates the hardware implementation for manipulating bincode-based images. In [24], nine fundamental image operations are involved; in [4], eight geometrical transformations are investigated. In fact, since the bincodes and our two proposed codes can be transformed into each other, the related image algorithms developed in the bincodes can be used by the proposed codes transparently.

When compared to Sarkar's method, except for spending a little more space, our proposed codes do have three advantages: (1) it is easier to extract the related geometrical coordinates; (2) the bincodes can be used as direct input; and (3) each restricted logicode represents a connected black block, so the restricted logicodes are more suitable for some operations such as neighbor finding [17, 18], which

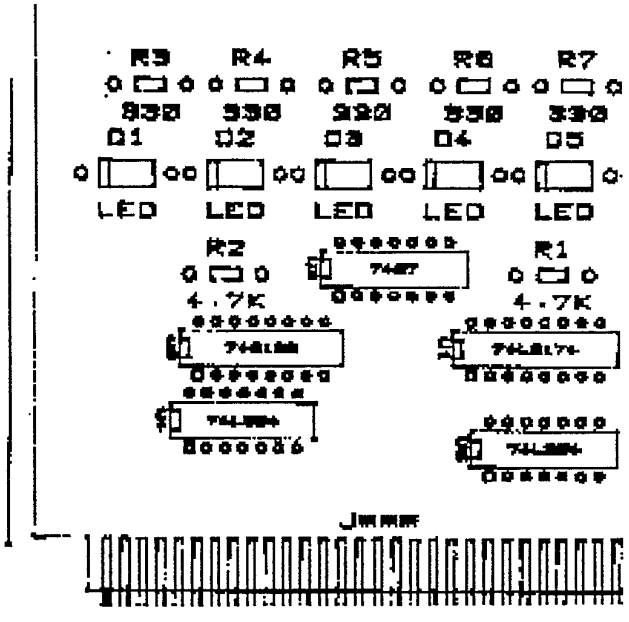



**FIG. 8.** A sample image.

**TABLE 1**
**Number of Bincodes, Logicodes, Restricted Logicodes, and Sarkar's Codes**

| | Bincodes | Logicodes | Restricted logicodes | Sarkar's codes |
|---|---|---|---|---|
| Text | 19945 | 11759 | 11497 | 11294 |
| | 1 | 59% | 73% | 57% |
| Figure | 20309 | 11980 | 13786 | 11010 |
| | 1 | 59% | 68% | 54% |
| Picture | 13373 | 9421 | 11627 | 9053 |
| | 1 | 70% | 87% | 68% |
| Mixed | 21771 | 12225 | 14374 | 11253 |
| | 1 | 56% | 66% | 52% |
| Total | 75398 | 45385 | 54284 | 42610 |
| | 1 | 60% | 72% | 57% |

**TABLE 2**
**Number of Ternary Digits for Bincodes, Logicodes, Restricted Logicodes, and Sarkar's Codes**

| | Bincodes | Logicodes | Restricted logicodes | Sarkar's codes |
|---|---|---|---|---|
| Text | 317872 | 186447 | 229984 | 180704 |
| | 170% | 1 | 123% | 97% |
| Figure | 313640 | 186824 | 214608 | 176160 |
| | 168% | 1 | 115% | 94% |
| Picture | 208684 | 148268 | 183272 | 144848 |
| | 141% | 1 | 124% | 98% |
| Mixed | 342352 | 193402 | 225960 | 180048 |
| | 177% | 1 | 117% | 93% |
| Total | 1186148 | 714941 | 853824 | 681760 |
| | 166% | 1 | 119% | 95% |

is the kernel of connected component labelling, than logicodes. Our future research topic focuses on (1) giving a new definition of the neighbors and designing an efficient neighbor-finding algorithm, (2) designing an algorithm for connected component labelling, and (3) other image manipulations.

## ACKNOWLEDGMENTS

The authors appreciate the three anonymous referees, Dr. K. Aizawa, and K. Rado for their valuable comments and corrections that helped to improve the presentation and quality of the paper. Especially, one referee spent a lot of time making all remarks directly on the manuscript. This research was supported in part by the National Science Council of R.O.C. under Contracts NSC85-2213-E003-001, NCHC86-08-015, and NSC86-2213-E011-010.

## REFERENCES

1. K. L. Chung and C. J. Wu, A fast search algorithm on modified S-trees, *Pattern Recognit. Lett.* **16,** 1995, 1159–1164.
2. K. L. Chung, J. G. Wu, and J. K. Lan, Efficient search algorithm on compact S-trees, Pattern Recognition Lett., in press.
3. K. L. Chung and C. Y. Huang, Finding neighbors on bincode-based images in $O(n \log \log n)$ time, *Pattern Recognit. Lett.* **17,** 1996, 1117–1124.
4. K. L. Chung, and J. G. Wu, Mirroring and rotating bincode-based Images, research report, Dept. of Information Management, National Taiwan Inst. of Technology, May 1996.
5. C. Dyer, The space efficiency of quadtrees, *Comput. Graphics Image Process.* **19**(4), 1982, 335–348.
6. H. Freeman, Computer processing of line-drawing images, *ACM Comput. Surveys* **6**(1), 1974, 57–97.
7. I. Gargantini, An effective way to represent quadtrees, *Commun. ACM* **25**(12), 1982, 905–910.
8. S. J. Hong, R. G. Cain, and D. L. Ostapko, MINI: A heuristic approach for logical minimization, *IBM J. Res. Dev.,* Sept., 1974, 443–458.
9. C. Y. Huang and K. L. Chung, Transformations between bincodes and the DF-expression, *Comput. & Graphics* **19**(4), 1995, 601–610.
10. C. Y. Huang and K. L. Chung, Fast operations on binary images using interpolation-based bintrees, *Pattern Recognit.* **28**(3) 1995, 409–420.
11. G. M. Hunter and K. Steiglitz, Operations on images using quad trees, *IEEE Trans. Pattern Anal. Mach. Intell.* **1**(2), 1979, 145–153.
12. W. D. Jonge, P. Scheuermann, and A. Schijf, $S^+$-Trees: An efficient structure for the representation of large pictures, *CVGIP: Image Understanding* **59,** 1994, 265–280.
13. E. McCluskey, Minimization of boolean functions, *Bell Syst. Tech. J.* **35,** 1965, 1417–1444.
14. R. E. Miller, *Switching Theory, Vol. 1: Combinational Circuits,* Wiley, New York, 1965.
15. M. A. Ouksel and A. Yaagoub, The interpolation-based bintree and encoding of binary images, *CVGIP: Graphical Models and Image Processing* **54**(1), 1992, 75–81.
16. W. V. Quine, The problem of simplifying truth functions, *Am. Math. Monthly* **59,** 1952, 521–531.
17. H. Samet, *Applications of Spatial Data Structures,* Addison–Wesley, New York, 1990.
18. H. Samet, *Design and Analysis of Spatial Data Structures,* Addison–Wesley, New York, 1990.
19. D. Sarkar, Boolean function-based approach for encoding of binary images, *Pattern Recognit. Lett.* **17**(8), 1996, 839–848.
20. D. Sarkar, Operations on binary images encoded as minimized boolean functions, *Pattern Recognit. Lett.* **18,** 1997, 455–463.
21. G. Schrack, Finding neighbors of equal size in linear quadtrees and octrees in constant time, *CVGIP: Image Understanding* **55**(3), 1992, 221–230.
22. C. A. Shaffer, R. Juvvadi, and L. S. Health, Generalized comparison of quadtree and bintree storage requirements, *Image Vision Comput.* **11**(7), 1993, 402–412.
23. M. Tamminen, Encoding pixel tress, *Comput. Vision Graphics Image Process.* **28**(1), 1984, 44–57.
24. J. G. Wu and K. L. Chung, A special-purpose processor for manipulating images represented by bincodes, research report, Dept. of Information Management, National Taiwan Inst. of Technology, Dec. 1995.

JUNG-GEN WU received the B.S., M.S., and Ph.D. degrees in Electrical Engineering from National Taiwan University of R.O.C. He is an associate professor in the Department of Computer and Information Education of the National Taiwan Normal University. From 1974 to 1976 he was a soldier in military service. His current research interests include image processing, spatial data structures, computer architecture, and parallel processing. He is a member of the IEEE and Computer Society.

KUO-LIANG CHUNG received the B.S., M.S., and Ph.D. degrees in Computer Science and Information Engineering from National Taiwan University of R. O. C. Since 1995, he has been a professor in the Department of Information Management of the National Taiwan Institute of Technology. From 1984 to 1986 he was a soldier in military service. His current research interests include image processing and pattern recognition, computer graphics and computational geometry, data and image compression, and high speed computing. He is a member of the ACM, IAPR, IEEE, and SIAM.