Pergamon

Pattern Recognition, Vol. 28, No. 3, pp. 409–420, 1995
Elsevier Science Ltd
Copyright © 1995 Pattern Recognition Society
Printed in Great Britain. All rights reserved
0031-3203/95 $9.50 + .00

0031-3203(94)00102-2

FAST OPERATIONS ON BINARY IMAGES USING INTERPOLATION-BASED BINTREES†

CHI-YEN HUANG and KUO-LIANG CHUNG‡
Department of Information Management, National Taiwan Institute of Technology, No. 43, Sec. 4, Keelung Rd., Taipei, Taiwan 10672, R.O.C.

(*Received* 23 *November* 1993; *in revised form* 17 *June* 1994; *received for publication* 17 *August* 1994)

Abstract—The Interpolation-Based Bintree (IBB) is a new encoding scheme for representing binary images. This encoding scheme combines the features of linear quadtrees, binary trees, and interpolation-based codes. The implementation of the IBB is shown to be very simple and storage-saving. Based on the IBB structure, this paper presents some fast sequential algorithms for set operations (intersection, union, and complement), 4-neighbors finding, and diagonal neighbors finding, respectively. Given two sets of bincodes, B_1 and B_2, with respect to two images, the set operations can be performed in $O(n + m)$ time, where n is the size of B_1 and m is the size of B_2; the complement operation for $B_1(B_2)$ can be performed in $O(n)(O(m))$ time; and the 4-neighbors finding and the diagonal neighbors finding for $B_1(B_2)$ can be accomplished in $O(n \log n)(O(m \log m))$ time.

Bincode Image compression Interpolation-based bintree Neighbor finding Set operation

1. INTRODUCTION

Representing and manipulating binary images are two important issues in those fields of pattern recognition, image processing, computer graphics, computational geometry, geographic information systems and robotics. According to the literature,[1] encoding techniques of binary images can be classified into three categories, namely tree, string and set of codes. The first type, tree, represents a binary image as a tree structure and stores it by using pointer-type data structure. For example, quadtree belongs to this type. Quadtree is a very successful coding scheme and its manipulations have been studied intensively in recent years.[2–5] The second type, string, represents a binary image as a series of strings. For example, chain codes and run-length codes belong to this type. Finally, the third type, set of codes, represents binary images as a series of codes such as linear quadtrees[5–8] and Interpolation-Based Bintrees (IBBs).[9]

The IBB is first proposed by Ouksel and Yaagoub.[9] The IBB has been shown to be very simple and storage-saving (0–25% space utilization improvement over the linear quadtrees encoding method in empirical comparisons[5,9,10]). This method is based on subdividing the binary image into two equal-sized subimages, right and left or up and down recursively until the element of the created subimages is either black or white, where each subimage is represented by a node. After this subdivision work has been done, we can convert these external black nodes, which represent the black blocks, into the interpolation-based codes. These final codes are called bincodes which are unique integers assigned for black blocks.

Based on the IBB structure, this paper presents some fast sequential algorithms for set operations (intersection, union and complement), 4-neighbors finding, and diagonal neighbors finding, respectively. Given two sets of bincodes, B_1 and B_2, with respect to two images, the set operations can be performed in $O(n + m)$ time with $O(n + m)$ memory space, where n is the size of B_1 and m is the size of B_2; the complement operation for $B_1(B_2)$ can be performed in $O(n)(O(m))$ time with $O(n)(O(m))$ memory space; the 4-neighbors finding and the diagonal neighbors finding for all bincodes of $B_1(B_2)$ can be accomplished in $O(n \log n)(O(m \log m))$ time with $O(n)(O(m))$ memory space.

This paper gives a review of the IBB in Section 2. Then we describe some fast sequential algorithms for solving the problems of set operations and neighbors finding in Section 3 and Section 4, respectively. Finally, some conclusions are addressed in Section 5.

2. REVIEW OF THE IBB

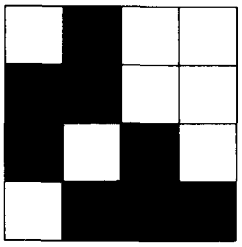
In this section, we first introduce the data structure of bintree,[11] then the IBB is reviewed. The bintree is a hierarchical data structure for storing binary images. Given a $2^N \times 2^N$ binary image, its bintree representation is a binary tree structure with maximal height $2N$. The root node of the bintree represents the whole image. If one image is totally black (white), then the root node is labeled with 1(0). Otherwise, the root node is gray

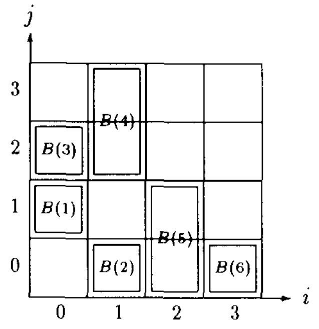
† This research was supported in part by the National Science Council of R.O.C.
‡ Author to whom all correspondence should be addressed.

and its two sons are added. Each son represents half of the image, which is covered by his father in the sense of spatial structure. This subdivision process is then repeated recursively for each of the two sons. In each recursive step, one son is corresponding to a subdivision of his father. If a subdivision is either black or white, then its corresponding node is an external node; otherwise it is a gray node (internal node). A node at height $h, h \leq 2N$, is corresponding to a $2^{N-\lfloor h/2 \rfloor} \times 2^{N-\lceil h/2 \rceil}$ block. When h is even, this block is square; when h is odd, this block is rectangular. According to the data structure of bintree, a $2^N \times 2^N$ binary image will have at most 2^{2N} external nodes and $2^{2N}-1$ internal nodes. Naturally, the bintree can be implemented by pointer-type data structure.(11)

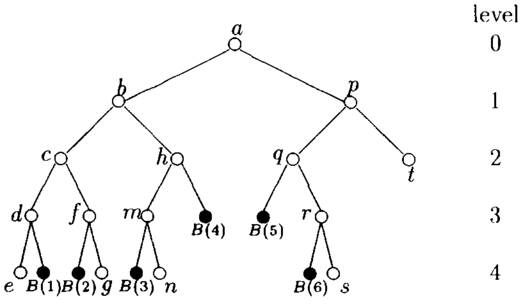
Considering a $2^2 \times 2^2$ binary image as shown in Fig. 1(a), the corresponding example of blocks is shown in Fig. 1(b). According to the above bintree's description, the bintree of Fig. 1(a) is illustrated in Fig. 1(c).

The **IBB** is based on the bintree structure and represents the bintree as an ordered collection of external


(a) Binary image



(b) Blocks example



(c) Bintree structure

Fig. 1. A $2^2 \times 2^2$ binary image.

black nodes. Each external black node is described by a numerical record converted by its location and level.

Given a $2^N \times 2^N$ binary image, if the node at level l of a bintree corresponds to a black block in the binary image at location (i, j), then the bincode b is derived by the following bincode conversion scheme:

(1) convert i and j to binary strings $i_{N-1}i_{N-2}\cdots i_0$ and $j_{N-1}j_{N-2}\cdots i_0$, respectively, where $i = \sum_{k=0}^{N-1}(i_k \times 2^k)$ and $j = \sum_{k=0}^{N-1}(j_k \times 2^k)$;
(2) compute $s = 2^{2N} - 2^{2N-l}$ and then convert s to a binary string $s_{2N-1}s_{2N-2}\cdots s_0$, where $s = \sum_{k=0}^{2N-1}(s_k \times 2^k)$; and
(3) The bincode b is given by $\sum_{k=0}^{N-1}(i_k \times 2^{4k+3}) + \sum_{k=0}^{N-1}(j_k \times 2^{4k+1}) + \sum_{k=0}^{2N-1}(s_k \times 2^{2k})$.

Return to Fig. 1(a). The block $B(1)$ is at location $(0, 1)$ and at level 4. According to the above bincode conversion scheme, we obtain that $i=0=(i_1i_0)_2=(00)_2, j=1=(j_1j_0)_2=(01)_2$, and $s=15=(s_3s_2s_1s_0)_2=(1111)_2$, then the bincode of $B(1)$ can be represented by $(01010111)_2 = 87$. Consider block $B(4)$ which is at location $(1, 2)$ and at level 3. Similarly, we obtain that $i=1=(i_1i_0)_2=(01)_2$, $j=2(j_1j_0)_2=(10)_2$, and $s=14=(s_3s_2s_1s_0)_2=(1110)_2$. Further, the bincode of block $B(4)$ is represented by $(01111100)_2 = 124$.

We traverse black nodes of the bintree in a preordered way and calculate their bincodes simultaneously, then the bincodes of Fig. 1(a) are represented by the ordered sequence of numerical records $\langle 87, 93, 117, 124, 212, 221 \rangle$. Throughout this paper, the **IBB** and bincodes represent the same thing and can be used exchangeably.

3. SET OPERATIONS

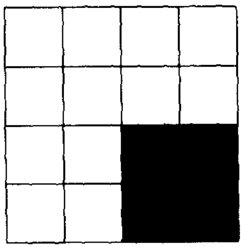
Based on the representation of bincodes described above, some fast sequential algorithms for set operations (intersection, union and complement) are presented in this section. We assume that all related binary images have been translated into bincodes and the associated level of each bincode is known.

3.1. *Intersection union*

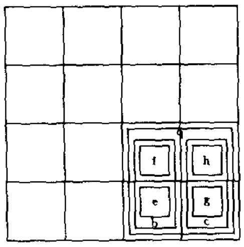
We start with analyzing some special properties of bincodes.

Lemma 1.(9) Given a $2^N \times 2^N$ binary image, if q is the bincode of an internal node in the bintree and b and c are bincodes of the left son and right son of q, then $b = q + 2^{2(2N-l-1)}$ and $c = q + 3 \times 2^{2(2N-l-1)}$, where l is the level of node q.

Let b be the left son of q and c be the right son of q, by Lemma 1, it is observed that $q < b < c$. Similarly, suppose e is the left son of b, f is the right son of b, g is the left son of c, and h is the right son of c. By Lemma 1, we obtain that $e = q + 2^{2(2N-l-1)} + 2^{2(2N-(l+1)-1)}$, $f = q + 2^{2(2N-l-1)} + 3 \times 2^{2(2N-(l+1)-1)}$, $g = q + 3 \times 2^{2(2N-l-1)} + 2^{2(2N-(l+1)-1)}$, and $h = q + 3 \times 2^{2(2N-l-1)} + 3 \times 2^{2(2N-(l+1)-1)}$. After comparing these



(a) Binary image



(b) Blocks example

Fig. 2. The covering example.

seven bincodes, we have $q < b < e < f < c < g < h$. Informally, we have the following lemma.

Lemma 2. (Increasing property.) The bincodes of a binary image form a strictly increasing sequence.

Lemma 3. (Covering property.) Let q be a bincode at level l, if some bincodes fall in $[q, q + (4^{2N-l} - 1)]$ then these bincodes are covered by bincode q in a sense of spatial structure.

Proof. By Lemma 1, we begin with the subdivision process for q, then this process is repeated downward level by level. By Lemma 2, finally we obtain an increasing sequence $\langle q, q + 2^{2(2N-l-1)}, q + 2^{2(2N-l-1)} + 2^{2(2N-(l+1)-1)}, \ldots, q + \sum_{k=l}^{2N-1}(3 \times 2^{2(2N-k-1)})\rangle$. Since $\sum_{k=l}^{2N-1}(3 \times 2^{2(2N-k-1)}) = 4^{2N-l} - 1$, the sequence can be rewritten as $\langle q, q + 2^{2(2N-l-1)}, q + 2^{2(2N-l-1)} + 2^{2(2N-(l+1)-1)}, \ldots, q + (4^{2N-l} - 1)\rangle$. We complete the proof. Q.E.D.

For convenience, $q + (4^{2N-l} - 1)$ is said to be the rightmost coverage of q.

For example, consider a $2^2 \times 2^2$ binary images as shown in Fig. 2(a) and the corresponding blocks are shown in Fig. 2(b). By the previous conversion scheme, the bincode of q is 208. By Lemma 1 and Lemma 2, the bincodes of blocks q, b, e, f, c, g and h are represented by the increasing sequence $\langle 208, 212, 213, 215, 220, 221, 223\rangle$. Since $N = 2$ and the level of black block q is $l = 2$, we have $q + (4^{2N-l} - 1) = 223$. By Lemma 3, all blocks q, b, e, f, c, g and h are covered by the block q.

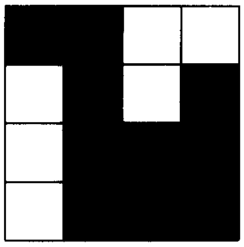
According to the description of image subdivision process for the construction of bintree, it is not hard to obtain the following lemma.

Lemma 4. In the representation of IBBs, any two subimages of the binary image do not exist partially covering relationship.

We sketch our basic concept for performing the intersection and union by the following example (see Fig. 3). Given an image $I(B_1)$ of Fig. 3(a), the bincodes of the first image are represented by $B_1 = \langle B_1(1), B_1(2), \ldots, B_1(5)\rangle = \langle 92, 119, \ldots, 253\rangle$, and the related levels of the bincodes are given by $l_1 = \langle l_1(1), l_1(2), \ldots, l_1(5)\rangle = \langle 3, 4, \ldots, 4\rangle$. First, it takes $O(1)$ time to compute the rightmost coverage for each bincode by using Lemma 3. For example, the rightmost coverage of 92 is 95 which is derived by $92 + (4^{4-3} - 1)$. As a result, the computed rightmost coverages are denoted by $Z_1 = \langle Z_1(1), Z_1(2), \ldots, Z_1(5)\rangle = \langle 95, 119, \ldots, 253\rangle$. In general, it takes $O(n)$ time to compute the rightmost coverages for all bincodes of B_1 if the size of B_1 is n. In Fig. 3(b), the bincodes of the second image are represented by $B_2 = \langle B_2(1), B_2(2), \ldots, B_2(6)\rangle = \langle 87, 93, \ldots, 221\rangle$ and the related levels are given by $l_2 = \langle l_2(1), l_2(2), \ldots, l\,2(6)\rangle = \langle 4, 4, \ldots, 4\rangle$. By the same arguments on the first image, we have $Z_2 = \langle Z_2(1), Z_2(2), \ldots, Z_2(6)\rangle = \langle 87, 93, \ldots, 221\rangle$. Similarly, it takes $O(m)$ time to compute the rightmost coverages for all bincodes of B_2 if the size of B_2 is m.

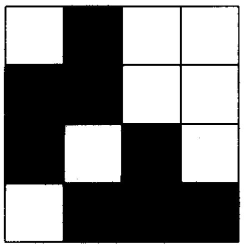
From $B_1(1) = 92$, $Z_1(1) = 95$, $B_2(1) = 87$, and $Z_2(1) = 87$, we know that $I(92)$ and $I(87)$ are disjoint, where $I(92)$ and $I(87)$ denote the black blocks with respect to the bincodes 92 and 87, so the smaller bincode 87 is put into the union set. Next, from $B_1(1) = 92$, $Z_1(1) = 95$, $B_2(2) = 93$, and $Z_2(2) = 93$, by Lemma 3, we have that $I(93)$ is covered by $I(92)$, so bincode 93 is put into the intersection set and bincode 92 is put into the union set. From $B_1(2) = 119$, $Z_1(2) = 119$, $B_2(3) = 117$, and $Z_2(3) = 117$, we have that $I(B_1(2))(= I(119))$ and $I(B_2(3))$ $(= I(117))$ are disjoint, then the smaller bincode 117 is included in the union set. Later, the bincode 119 is also included in the union set because $I(B_1(2))(= I(119))$ and $I(B_2(4))(= I(124))$ are disjoint too and 119 is smaller than 24. Next, since $B_1(3) = B_2(4) = 124$ and $Z_1(3) = Z_2(4) = 127$, the bincode 124 is copied to the intersection set and the union set. Next, from $B_1(4) = 208$, $Z_1(4) = 223$, $B_2(5) = 212$, and $Z_2(5) = 215$, bincode 212 $(= B_2(5))$ is put into the intersection set because bincode 212 is covered by bincode 208. In addition, by Lemma 3, we know that $I(B_2(6))$ is also covered by $I(B_1(4))$, so $221(= B_2(6))$ is put into the intersection set too. Further, $208(= B_1(4))$ is included in the union set. Finally, the remaining bincode of the first image, 253, is included in the union set directly. As a result, the intersection set is $\langle 93, 124, 212, 221\rangle$ [see Fig. 3(c)] and the union set is $\langle 87, 92, 117, 119, 124, 208, 245\rangle$. Suppose the size of B_1 is n and the size of B_2 is m, it needs $O(n + m)$ time to obtain the intersection set and the union set.

Furthermore, we want to use less number of bincodes to represent the union set. By Lemma 1, the first bincode 87 in the union set is equal to $84 + 3 \times (4^{4-3-1})$ and the second bincode 92 in the union set is equal to $80 + 3 \times (4^{4-2-1})$. Bincodes 87 and 92 cannot be compressed into a single bincode because they have no brother-relationship each other. Similarly, the second bincode 92 and the third bincode 117 cannot be compressed into a single bincode too. By Lemma 1, the



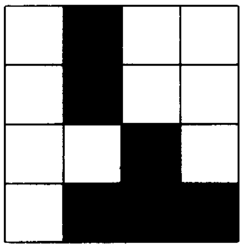
B_1	92	119	124	208	253
l_1	3	4	3	2	4
Z_1	95	119	127	223	253

(a) $I(B_1(i))$



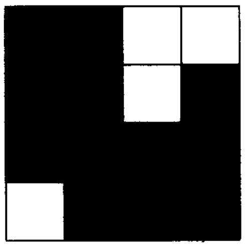
B_2	87	93	117	124	212	221
l_2	4	4	4	3	3	4
Z_2	87	93	117	127	215	221

(b) $I(B_2(j))$



bincodes =
⟨93, 124, 212, 221⟩

(c) The intersection of (a) and (b)



bincodes =
⟨87, 92, 112, 208, 253⟩

(d) The union of (a) and (b)

Fig. 3. An intersection/union example.

third bincode 117 and the fourth bincode 119 are brothers each other since $117 = 116 + (4^{4-3-1})$ and $119 = 116 + 3 \times (4^{4-3-1})$. Therefore, these two bincodes are compressed into a single bincode 116. Further, the bincode 116 and the fifth bincode 124 are compressed into a bincode 112 since they are brothers each other too. Finally, the compact union set equals ⟨87, 92, 112, 208, 245⟩. During the compression process, each of the external nodes and internal nodes in the bintree will be traversed at most twice, so the time complexity required is $O(n + m)$.

Based on the previous description, our algorithm for performing the intersection and union operations is illustrated in Fig. 4. We have the following theorem.

Theorem 5. Our intersection/union algorithm can be accomplished in $O(n + m)$ time with $O(n + m)$ memory space.

3.2. *Complement*

Given a binary image I, the complement of I is an image which converts white (black) pixels in I into black (white) pixels. In this subsection, we present a linear time algorithm to perform the complement of the bincodes of I.

For bincode b, we define the highest ancestor of bincode b to be a bincode which corresponds to the largest subimage which covers the subimage represen-

```
Algorithm : Intersection and Union
Input: $B_1 = \langle B_1(1), B_1(2), ..., B_1(n)\rangle$, $B_2 = \langle B_2(1), B_2(2), ..., B_2(m)\rangle$,
       and the associated levels.
Output: The intersection set and the compact union set.

Step_1. \* Calculate the rightmost coverage of each bincode for $B_1$ and $B_2$. *\
        For $i \leftarrow 1$ to $n$ do
            $Z_1(i) \leftarrow B_1(i) + (4^{2N-l_1(i)} - 1)$
        end
        For $j \leftarrow 1$ to $m$ do
            $Z_2(j) \leftarrow B_2(j) + (4^{2N-l_2(j)} - 1)$
        end

Step_2. \* Find the intersection set and the union set. *\
        \* $T$ denotes the intersection set and $U$ denotes the union set. *\
        $T \leftarrow \phi$; $U \leftarrow \phi$; $i \leftarrow 1$; $j \leftarrow 1$
        While $i \le n$ and $j \le m$ do
            Case
              :$Z_1(i) < B_2(j)$ or $B_1(i) > Z_2(j)$:
                                    \* $I(B_1(i))$ and $I(B_2(j))$ are disjoint. *\
                If $B_1(i) < B_2(j)$
                then
                    $U \leftarrow U \cup \{B_1(i)\}$; $i \leftarrow i + 1$
                else
                    $U \leftarrow U \cup \{B_2(j)\}$; $j \leftarrow j + 1$
              :$B_1(i) = B_2(j)$:      \* $I(B_1(i))$ and $I(B_2(j))$ are equivalent. *\
                $T \leftarrow T \cup \{B_1(i)\}$; $U \leftarrow U \cup \{B_1(i)\}$; $i \leftarrow i + 1$; $j \leftarrow j + 1$
              :$B_1(i) > B_2(j)$ and $Z_1(i) \le Z_2(j)$:    \* $I(B_1(i)) \subset I(B_2(j))$. *\
                While $B_1(i) > B_2(j)$ and $Z_1(i) \le Z_2(j)$ and $i \le n$ do
                    $T \leftarrow T \cup \{B_1(i)\}$; $i \leftarrow i + 1$
                end        \* end while *\
                $U \leftarrow U \cup \{B_2(j)\}$; $j \leftarrow j + 1$
              :$B_2(j) > B_1(i)$ and $Z_2(j) \le Z_1(i)$:    \* $I(B_2(j)) \subset I(B_1(i))$. *\
                While $B_2(j) > B_1(i)$ and $Z_2(j) \le Z_1(i)$ and $j \le m$ do
                    $T \leftarrow T \cup \{B_2(j)\}$; $j \leftarrow j + 1$
                end        \* end while *\
                $U \leftarrow U \cup \{B_1(i)\}$; $i \leftarrow i + 1$
            end        \* end case *\
        end        \* end while *\

            \* Processing the remaining bincodes of $B_1$ or $B_2$. *\
            While $i \le n$ do
                $U \leftarrow U \cup \{B_1(i)\}$; $i \leftarrow i + 1$
            end
            While $j \le m$ do
                $U \leftarrow U \cup \{B_2(j)\}$; $j \leftarrow j + 1$
            end

Step_3. \* Compress the union set. *\
            \* The union set formed in step_2 is $\{U(k) | k = 1, 2, ..., q\}$. *\
            \* $p(k)$ and $s(k)$ represent the indices of the predecessive bincode
               and the successive bincode of the $k$-th bincode. *\
            For $k = 1$ to $q$ do
                $p(k) \leftarrow k - 1$; $s(k) \leftarrow k + 1$
            end
            $k \leftarrow 2$
            While $k \le q$ do
                If $k < q$ and $U(s(k)) - U(k) = 2 \times 4^{2N-l(s(k))}$
                then
                    $U(k) \leftarrow U(k) - 4^{2N-l(k)}$; $l(k) \leftarrow l(k) - 1$; delete $U(s(k))$;
                    $s(k) \leftarrow s(k) + 1$; $p(s(k)) \leftarrow k$
```

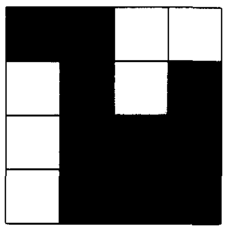
Fig. 4. The intersection/union algorithm.

```
    else
      If $U(k) - U(p(k)) = 2 \times 4^{2N-l(k)}$
      then
        $U(k) \leftarrow U(k) - 3 \times 4^{2N-l(k)}$; $l(k) \leftarrow l(k) - 1$; delete $U(p(k))$;
        $p(k) \leftarrow p(k) - 1$; $s(p(k)) \leftarrow k$
      else
        $k \leftarrow s(k) + 1$
end        \* end while *\
```

Fig. 4(*Continued*)

ted by bincode b but does not cover the subimage represented by the successive bincode of bincode b. Note that the highest ancestor of the last bincode is the root. The complement of b is defined to be the bincodes which represent the subimages of the highest ancestor of b, excepting the subimage of b and the subimages of the highest ancestors of all the previous bincodes of b.

As shown in Fig. 1(c), the highest ancestor of the first black block $B(1)$ is d. Excepting $B(1)$, the white block e covered by d forms the complement of $B(1)$, so the bincode e is assigned to be the complement of $B(1)$. Using the same definition, the highest ancestor of the second black block $B(2)$ is c. Therefore, g is assigned to be the complement of $B(2)$. Similarly, m is the highest ancestor of $B(3)$, then n is assigned to be the complement of $B(3)$. The highest ancestor of the fourth black block $B(4)$ is b. Thus, the complement of $B(4)$ is empty, ϕ. The complement of $B(5)$ is ϕ too because the highest ancestor of $B(5)$ is itself. For $B(6)$, the highest ancestor of $B(6)$ is a, so s and t are assigned to be the complement of $B(6)$. As a result, $\langle e, g, n, s, t\rangle$ constitutes the complement of all black blocks, i.e. forms the complement of the bincodes of Fig. 1(a). In addition, these five bincodes $\langle e, g, n, s, t\rangle$ form an increasing sequence since by Lemma 1, we have $e < g < n < s < t$.



bincodes =

$\langle 92,\ 119,\ 124,\ 208,\ 253\rangle$

(a) A $2^2 \times 2^2$ binary image

(1) 92: $T'_1 = \langle 84\rangle$

(2) 119: $T'_2 = \langle 117\rangle$

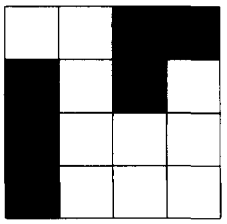
(3) 124: $T'_3 = \phi$

(5) 208: $T'_4 = \phi$

(6) 253: $T'_5 = \langle 244,\ 255\rangle$

$$T' = \bigcup_{k=1}^{5} T'_k = \langle 84,\ 117,\ 244,\ 255\rangle$$

(b) Simulation process



bincodes =

$\langle 84,\ 117,\ 244,\ 255\rangle$

(c) The complement of (a)

Fig. 5. A complement example.

We now sketch the main concept for finding the complement of one image by an example illustrated in Fig. 5. In what follows, we want to find the complement of the image of Fig. 5(a).

Consider the first two bincodes 92 and 119. By Lemma 1, the bincode of the brother of 92 is 84 ($92 = 80 + 3 \times (4^{4-2-1})$ and $84 = 80 + 4^{4-2-1}$), which can be computed in $O(1)$ time. Since bincode 92 is the first bincode and 84 is the left brother of bincode 92, thus, we save 84 as the temporary complement of bincode 92 directly, which is denoted by $T'_1 = \langle 84\rangle$. In other words, the subimage represented by bincode 80 has been traversed. Next, we check the subimage of bincode 112 because bincode 112 is the brother of bincode 80 ($80 = 64 + 4^{4-1-1}$ then $112 = 64 + 3 \times (4^{4-1-1})$). The rightmost coverage of 112 is 127 ($127 = 112 + (4^{4-2} - 1)$). By Lemma 3, then we know that 112 covers the next given bincode 119, the work for finding the complement of bincode 92 is finished. The real complement of bincode 92 is 84 only. We also have $T'_1 = \langle 84\rangle$.

Now, we check the second bincode 119 and the third bincode 124. By Lemma 1, the left brother of bincode 119 is bincode 117 ($119 = 116 + 3 \times (4^{4-3-1})$ then $117 = 116 + 4^{4-3-1}$). Since the subimage of 117 does not have been checked, 117 is assigned to be the temporary complement of bincode 119. Next, we check bincode 124 ($116 = 112 + 4^{4-2-1}$ then $124 = 112 + 3 \times (4^{4-2-1})$) which is the brother of bincode 116. Since the rightmost coverage of 124 is 127 ($127 = 124 + (4^{4-3} - 1)$), we know that it covers the next bincode 124. As a result, the real complement of bincode 119 is determined. $T'_2 = \langle 117\rangle$ denotes this complement.

Next, we check the bincodes 124 and 208. Bincode 116 is the left brother of 124 ($124 = 112 + 3 \times (4^{4-2-1})$ then $116 = 112 + 4^{4-2-1}$). We bypass 116 because the subimage covered by 116 has been checked. By Lemma 1, we have that the subimage represented by bincode 112 has been checked, then we check bincode 80 ($112 = 64 + 3 \times 4^{4-1-1}$ then $80 = 64 + 4^{4-1-1}$) subsequently. Similarly, we bypass 80 because the subimage covered

```
Algorithm : Complement
Input: Bincodes $\langle B(1), B(2), ..., B(n)\rangle$ and the associated levels.
Output: The complement set $T'$.
Step_1. \* Initialization. *\
        \* $T_i'$ denotes the complement of bincode $B(i)$. *\
        \* $\langle H(1), H(2), ..., H(n)\rangle$ denote the bincodes whose corresponding
           subimages have been checked. *\
        For $i \leftarrow 1$ to $n$ do
            $T_i' \leftarrow \phi$; $H(i) \leftarrow -1$
        end
        $T' \leftarrow \phi$; $k \leftarrow 1$
Step_2. \* Find the complement for each bincode. *\
        For $i \leftarrow 1$ to $n$ do
            $b \leftarrow B(i)$; $v \leftarrow l(i)$
            While $v > 0$ do
                $r \leftarrow b \bmod 4^{2N-v+1}$
                If $r = 3 \times 4^{2N-v}$
                then      \* $b$ is the right brother. *\
                    $D \leftarrow b - 2 \times 4^{2N-v}$      \* $D$ is the left brother of $b$. *\
                    If $D \neq H(k)$
                    then      \* $D$ does not have been checked. *\
                        $T_i' \leftarrow \{D\} + T_i'$; $b \leftarrow b - r$; $v \leftarrow v - 1$
                    else      \* $D$ has been checked. *\
                        $H(k) \leftarrow -1$; $k \leftarrow k - 1$; $b \leftarrow b - r$; $v \leftarrow v - 1$
                else      \* $b$ is the left brother. *\
                    $D \leftarrow b + 2 \times 4^{2N-v}$; $Z \leftarrow D + (4^{2N-v} - 1)$
                                              \* $D$ is the right brother of $b$. *\
                    If $(i + 1) > n$ or $B(i + 1) > Z$      \* i.e., $B(i + 1) \notin [D, Z]$. * \
                    then      \* $D$ does not cover the next bincode. *\
                        $T_i' \leftarrow T_i' + \{D\}$; $b \leftarrow b - r$; $v \leftarrow v - 1$
                    else      \* $D$ covers the next bincode. *\
                        $k \leftarrow k + 1$; $H(k) \leftarrow b$; $v \leftarrow 0$
            end    \* end while *\
        end    \* end for *\
Step_3. \* Union the complements. *\
        For $i \leftarrow 1$ to $n$ do
            $T' \leftarrow T' \cup T_i'$
        end
```

Fig. 6. The complement algorithm.

by 80 also has been checked. In other words, by Lemma 1, the subimage represented by bincode 64 has been checked. Next, we check bincode 192, which is the right brother of bincode 64 ($64 = 0 + 4^{4-0-1}$ then $192 = 0 + 3 \times (4^{4-0-1})$). Bincode 192 covers the next bincode 208 since the rightmost coverage of 192 is 255. The final complement of bincode 124 is empty, ϕ. $T_3' = \phi$ denotes this complement. By the same arguments, we have that $T_4' = \phi$ and $T_5' = \langle 244, 255\rangle$. After collecting these complements for those bincodes, the complement of Fig. 5(a) is $T' = \bigcup_{k=1}^{5} T_k' = \langle 84, 117, 244, 255\rangle$, which is shown in Fig. 5(b). Note that T' is also an increasing sequence.

Following the above detailed simulation, the concept for finding the complement of the bincode can be described as follows. Given a bincode, b, let c be the brother of b, d be the father of b and c, and e be the brother of d. We first check bincode c. After the bincode c has been checked, the temporary complement for the subimage represented by d is found. Then we can check another subimage which is represented by e subsequently. This approach can be applied iteratively until the highest ancestor of b is reached. Finally, the complement of bincode b will be found. Meanwhile, we record the highest ancestor of b since the subimage represented by the highest ancestor of b has been checked and it needs not be checked any longer for finding the complement of the successive bincodes. This concept can be generalized to find the complement of all bincodes and we have the following theorem.

Theorem 6. The complement of an image is the union of the complements of bincodes of the image.

Our complement algorithm is shown in Fig. 6.

According to above description, each of the external nodes and internal nodes in the bintree will be traversed at most twice. As a result, the complement algorithm can be done in $O(n)$ time, where n is the number of bincodes in the IBB. In addition, the required memory is $O(n)$. Then we have the following result.

Theorem 7. Our complement algorithm can be done in $O(n)$ time with $O(n)$ memory space.

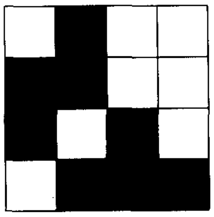
4. NEIGHBORS FINDING

Neighbor finding is important in the field of manipulations on binary images. Following the definition used in reference (3), we define the black block b to be the 4-neighbor of the black block c if blocks b and c share a common edge and the size of block b is equal to or larger than that of block c. The black block d is the diagonal neighbor of block c if block d shares a common corner with block c and it does not relate with its size. In this section, we present two fast algorithms for solving the problems of the 4-neighbor finding and the diagonal neighbor finding, respectively.

4.1. *4-neighbor finding*

According to the previous definition, as shown in Fig. 2(b), block f is the north neighbor of block e; block g is the east neighbor of block e; block c is the east neighbor of block e too. In the following, we first present the east neighbor finding algorithm. Furthermore, this algorithm can be modified to solve all problems of the 4-neighbor finding.

For the purpose of finding the east neighbor of bincode c, we first define the *potential* east neighbor of bincode c. It can be white, gray, or black and shares the east edge of bincode c and its size is equal to the size of bincode c. For example, in Fig. 2(b), block g is the *potential* east neighbor of block e but block c is not



bincodes =

⟨87, 93, 117, 124, 212, 221⟩

(a) Binary image

$B(k)$	87	93	117	124	212	221
$i(k)$	0	1	0	1	2	3
$j(k)$	1	0	2	2	0	0
$l(k)$	4	4	4	3	3	4
$Z(k)$	87	93	117	127	215	221
$i^*(k)$	1	2	1	2	3	4
$B^*(k)$	95	213	125	244	220	–
$E(k)$	–	212	124	–	–	–

(b) East neighbors finding

Fig. 7. An east neighbors finding example.

the *potential* east neighbor of block e because the size of c is greater than that of e.

After the *potential* east neighbor of each bincode is determined, by Lemma 3, the binary search method can be used to find the bincode which covers the *potential* east neighbor. If the bincode exists, the east neighbor is found; otherwise, it does not have the east neighbor bincode.

As shown in Fig. 1(b), the bincodes ⟨$B(1)$, $B(2)$, $B(3)$, $B(4)$, $B(5)$, $B(6)$⟩ represent the binary image of Fig. 1(a). The lower half of $B(4)$ can be defined as the *potential* east neighbor of $B(3)$ since it shares the east edge of $B(3)$ and its size equals that of $B(3)$. Further, it must be checked that which one of these six bincodes covers the lower half of $B(4)$. $B(4)$ covers this subimage, so $B(4)$ is the east neighbor of $B(3)$.

The example of the east neighbor finding is illustrated in Fig. 7. In Fig. 7(b), the bincodes of Fig. 7(a) are represented by ⟨$B(1)$, $B(2)$, $B(3)$, $B(4)$, $B(5)$, $B(6)$⟩ = ⟨87, 93, 117, 124, 212, 221⟩, the associated locations are represented by (column_index, row_index)'s, where the column_indexes are denoted by ⟨$i(1)$, $i(2)$, $i(3)$, $i(4)$, $i(5)$, $i(6)$⟩ = ⟨0, 1, 0, 2, 3⟩ and the row_indexes are denoted by ⟨$j(1)$, $j(2)$, $j(3)$, $j(4)$, $j(5)$, $j(6)$⟩ = ⟨1, 0, 2, 2, 0, 0⟩, respectively; the corresponding levels are represented by ⟨$l(1)$, $l(2)$, $l(3)$, $l(4)$, $l(5)$, $l(6)$⟩ = ⟨4, 4, 4, 3, 3, 4⟩. By Lemma 3, it needs $O(1)$ time to compute the rightmost coverage for each bincode. These bincodes' rightmost coverages are written as ⟨$Z(1)$, $Z(2)$, $Z(3)$, $Z(4)$, $Z(5)$, $Z(6)$⟩ = ⟨87, 93, 117, 127, 215, 221⟩. Next, the *potential* east neighbor of bincode $B(k)$ which is at location $(i(k), j(k))$ and at level $l(k)$ can be defined at location $(i^*(k), j(k))$ and at level $l(k)$, where $i^*(k) = i(k) + 2^{N-\lceil \frac{l(k)}{2} \rceil}$. For example, the first bincode 87 is at location (0, 1) and at level 4, so the *potential* east neighbor of bincode 87 can be defined at location (1, 1) and at level 4 since $i^*(1) = i(1) + 2^{N-\lceil \frac{l(1)}{2} \rceil} = 0 + 2^{2-\lceil \frac{4}{2} \rceil} = 0 + 1 = 1$. It needs $O(1)$ time to compute the new column_index $i^*(k)$ for each bincode. Then the new column_indexes of the *potential* east neighbors of Fig. 7(a) are specified by ⟨$i^*(1)$, $i^*(2)$, $i^*(3)$, $i^*(4)$, $i^*(5)$, $i^*(6)$⟩ = ⟨1, 2, 1, 2, 3, 4⟩. Therefore, by the bincode conversion scheme which is introduced in Section 2, the *potential* east neighbor of each bincode can be calculated easily. For example, for the first bincode 87, $B^*(1) = (i_1^* s_3 j_1 s_2 i_0^* s_1 j_0 s_0)_2 = (01011111)_2 = 95$ since $i^*(1) = 1 = (i_1^* i_0^*)_2 = (01)_2$, $j(1) = 1 = (j_1 j_0)_2 = (01)_2$, and $s = 2^4 - 2^{4-4} = 15 = (s_3 s_2 s_1 s_0)_2 = (1111)_2$. The *potential* east neighbors are written as ⟨$B^*(1)$, $B^*(2)$, $B^*(3)$, $B^*(4)$, $B^*(5)$, $B^*(6)$⟩ = ⟨95, 213, 125, 244, 220, –⟩. Totally, it takes $O(n)$ time for finding all the *potential* east neighbors of the IBB.

For each *potential* east neighbor among the successive sequence of bincodes by using the binary search algorithm,[12] we need $O(\log n)$ time for finding its covering bincode whose subimage covers this *potential* east neighbor. For example, for the first bincode 87, the *potential* east neighbor of 87 is 95 ($= B^*(1)$). It has five successive bincodes, i.e. $B(2)$, $B(3)$, $B(4)$, $B(5)$, and $B(6)$. From $B(4) = 124$ and $Z(4) = 127$ ($4 = \lceil \frac{(2+6)}{2} \rceil$), by Lemma 3, we know that the subimage represented by $B(4)$ does not cover 95 and 95 is smaller than $B(4)$. Next, from $B(2)=93$ and $Z(2)=93$ $(2=\lfloor\frac{(2+(4-1))}{2}\rfloor)$, the subimage represented by $B(2)$ does not cover 95 and 95 is larger than $B(2)$. Finally, from $B(3)=117$ and $Z(3)=117$, we have that the subimage represented by $B(3)$ does not cover 95 too. Therefore, the first bincode 87 does not have the east neighbor. For the second bincode 93, the *potential* east neighbor of 93 is 213. It has four successive bincodes, i.e. $B(3)$, $B(4)$, $B(5)$ and $B(6)$. From $B(4)=124$ and $Z(4)=127$ $(4=\lfloor\frac{(3+6)}{2}\rfloor)$, by Lemma 3, we know that the subimage represented by $B(4)$ does not cover 213 and 213 is larger than $B(4)$. Next, we try $B(5)=212$ and $Z(5)=215$ $(5=\lfloor\frac{((4+1)+6)}{2}\rfloor)$, then we have that the subimage represented by $B(4)$ covers 213 since $B(5)<213$ and $Z(5)>213$. As a result, we have that bincode 212 is the east neighbor of bincode 93. By the same arguments, all the east neighbors of bincodes can be found. The east neighbors of bincodes $\langle B(1), B(2), B(3), B(4), B(5), B(6)\rangle$ are represented by $\langle E(1), E(2), E(3), E(4), E(5), E(6)\rangle = \langle -, 212, 124, -, -, -\rangle$. We conclude that the second bincode 87 and the third bincode 117 have the east neighbor and the others do not have.

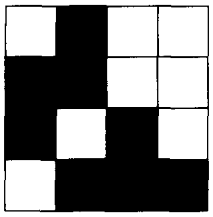
According to the above description, each bincode needs $O(\log n)$ time for finding its east neighbor bincode, where n is the number of the given bincodes. Totally, for all bincodes, it needs $O(n \log n)$ time. Our algorithm is presented in Fig. 8.

In fact, our algorithm for finding east neighbors can be modified to find the 4-neighbors easily and we have the following theorem.

Theorem 8. For all bincodes, the 4-neighbors finding algorithm can be done in $O(n \log n)$ time with $O(n)$ memory space.

```
Algorithm : East neighbors finding
Input: Bincodes $\langle B(1), B(2), ..., B(n)\rangle$ and the associated locations and levels.
Output: The east neighbors $\langle E(1), E(2), ..., E(n)\rangle$.
Step_1. \* Calculate the rightmost coverage and the potential east
              neighbor for each bincode. *\
        \* $Z(k)$ denotes the rightmost coverage of the $k$-th bincode and
              $B^*(k)$ denotes the potential east neighbor of the $k$-th bincode. *\
        For $k \leftarrow 1$ to $n$ do
            $Z(k) \leftarrow B(k) + (4^{2N-l(k)} - 1)$; $i^*(k) \leftarrow i(k) + 2^{N-\lceil\frac{l(k)}{2}\rceil}$
            If $0 < i^*(k)$ and $i^*(k) \leq 2^N - 1$
            then
                $B^*(k) \leftarrow \Sigma_{r=0}^{N-1}(i_r^* \times 2^{4r+3}) + \Sigma_{r=0}^{N-1}(j_r \times 2^{4r+1})$
                        $+ \Sigma_{r=0}^{2N-1}(s_r \times 2^{2r})$
                \* $i^*(k) = (i_{N-1}^* i_{N-2}^* \cdots i_1^* i_0^*)_2$,
                   $j(k) = (j_{N-1} j_{N-2} \cdots j_1 j_0)_2$, and
                   $2^{2N} - 2^{2N-l(k)} = (s_{2N-1} s_{2N-2} \cdots s_1 s_0)_2$. *\
            else
                $B^*(k) \leftarrow -1$
        end
Step_2. \* Find the east neighbor for each bincode. *\
        For $k \leftarrow 1$ to $n$ do
            $E(k) \leftarrow -1$
        end
        For $k \leftarrow 1$ to $n-1$ do
            If $B^*(k) \neq -1$
            then      \* binary search algorithm is used to find
                         the east neighbor. *\
              $a \leftarrow k+1$; $b \leftarrow n$
              While $E(k) = -1$ and $a \leq b$ do
                $k' \leftarrow \lfloor\frac{a+b}{2}\rfloor$
                If $B^*(k) \geq B(k')$ and $B^*(k) \leq Z(k')$
                then
                    $E(k) \leftarrow B(k')$
                else
                    If $B^*(k) < B(k')$ then $b \leftarrow k' - 1$
                                           else $a \leftarrow k' + 1$
              end
        end
```

Fig. 8. The east neighbors finding algorithm.



bincodes =

⟨87, 93, 117, 124, 212, 221⟩

(a) Binary image

$B(k)$	87	93	117	124	212	221
$i(k)$	0	1	0	1	2	3
$j(k)$	1	0	2	2	0	0
$l(k)$	4	4	4	3	3	4
$Z(k)$	87	93	117	127	215	221
$i^*(k)$	1	2	1	2	3	4
$j^*(k)$	2	1	3	4	2	1
$B^*(k)$	125	215	127	–	252	–
$N_e(k)$	124	–	–	–	–	–

(b) Northeast neighbors finding

Fig. 9. A northeast neighbors finding example.

4.2. *Diagonal neighbor finding*

According to the definition of the diagonal neighbor, as shown in Fig. 1(b), block $B(4)$ is the northeast neighbor of block $B(1)$ and is the northwest neighbor of block $B(5)$. On the other hand, block $B(1)$ is the southwest neighbor of block $B(4)$ and block $B(5)$ is the southeast neighbor of block $B(4)$. For a bincode, it has at most four diagonal neighbors, namely the northeast neighbor, the southeast neighbor, the southwest neighbor, and the northwest neighbor. The diagonal neighbor finding algorithm can be obtained from the modification of the 4-neighbor finding algorithm because the key idea and the approach for finding diagonal neighbor are same as the 4-neighbor algorithm. In this subsection, we mainly describe the northeast neighbor finding algorithm. However, this algorithm also can be modified to solve the problem of all diagonal neighbors finding.

For a given bincode p, we define q to be the *potential* northeast neighbor of bincode p when the size of q is 1×1 and it can be white or black and q shares the northeast corner of bincode p. For example, in Fig. 2(b), the subimage represented by block h can be defined as the *potential* northeast neighbor of block e since block h shares the northeast corner of block e and its size is 1×1. By Lemma 3, the binary search method is used to find the bincode which covers q and shares the northeast corner of p. If the bincode exists, the northeast neighbor of p is found. Otherwise, p does not have the northeast neighbor bincode.

As shown in Fig. 2(b), the subimage represented by block h can be defined as the *potential* northeast neighbor of block e. Blocks h and c cover this subimage. h is connected by the northeast corner of e but c is not, so h is the northeast neighbor of e but c is not.

For detailed description, we take an example of Fig. 9 to demonstrate our concept. As shown in Fig. 9(b), the bincodes of Fig. 9(a) are represented by $\langle B(1), B(2), B(3), B(4), B(5), B(6)\rangle = \langle 87, 93, 117, 124, 212, 221\rangle$, the associated (column_index, row_index)'s are $\langle i(1), i(2), i(3), i(4), i(5), i(6)\rangle = \langle 0,1,0,1,2,3\rangle$ and $\langle j(1), j(2), j(3), j(4), j(5), j(6)\rangle = \langle 1,0,2,2,0,0\rangle$, respectively, and the associated levels are $\langle l(1), l(2), l(3), l(4), l(5), l(6)\rangle = \langle 4,4,4,3,3,4\rangle$. The bincodes' rightmost coverages are $\langle Z(1), Z(2), Z(3), Z(4), Z(5), Z(6)\rangle = \langle 87, 93, 117, 127, 215, 221\rangle$, by Lemma 3, it takes $O(1)$ time to compute the rightmost coverage for each bincode.

Next, for bincode $B(k)$, which is at location $(i(k), j(k))$ and at level $l(k)$, the *potential* northeast neighbor of the bincode $B(k)$ can be defined at location $(i^*(k), j^*(k))$ and at level $2N$, where $i^*(k) = i(k) + 2^{N-\lceil\frac{l(k)}{2}\rceil}$ and $j^*(k) = j(k) + 2^{N-\lceil\frac{l(k)}{2}\rceil}$. For example, the first bincode 87 is at location $(0, 1)$ and at level 4. The *potential* northeast neighbor of bincode 87 can be defined at location $(1, 2)$ and at level 4 since $i^*(1) = i(1) + 2^{N-\lceil\frac{l(1)}{2}\rceil} = 0 + 1 = 1$, $j^*(1) + 2^{N-\lceil\frac{l(1)}{2}\rceil} = 1 + 1 = 2$, and $2N = 4$. It takes $O(1)$ time for computing the new column_index and the new row_index for each bincode. The new column_indexes of the *potential* northeast neighbors of Fig. 9(a) are specified by $\langle i^*(1), i^*(2), i^*(3), i^*(4), i^*(5), i^*(6)\rangle = \langle 1, 2, 1, 2, 3, 4\rangle$ and the new row_indexes of the *potential* northeast neighbors of Fig. 9(a) are specified by $\langle j^*(1), j^*(2), j^*(3), j^*(4), j^*(5), j^*(6)\rangle = \langle 2, 1, 3, 4, 2, 1\rangle$.

Based on the parameters of $(i^*(k), j^*(k))$ and $2N$, the *potential* northeast neighbor of bincode $B(k)$, denoted by $B^*(k)$, can be calculated by $O(1)$ time by using the bincode conversion scheme. The *potential* northeast neighbors are written as $\langle B^*(1), B^*(2), B^*(3), B^*(4), B^*(5), B^*(6)\rangle = \langle 125, 215, 127, -, 252, -\rangle$. Finally, the binary search algorithm is used. By the same arguments described in Section 4.1, the northeast neighbor of each bincode can be found in $O(\log n)$ time. The northeast neighbors of $\langle B(1), B(2), B(3), B(4), B(5), B(6)\rangle$ are represented by $\langle N_e(1), N_e(2), N_e(3), N_e(4), N_e(5), N_e(6)\rangle = \langle 124, -, -, -, -, -\rangle$. As shown in Fig. 9(b), the first bincode 87 has a northeast neighbor 124.

The northeast neighbor finding algorithm is shown in Fig. 10. Extending this algorithm directly, we have the following result.

Theorem 9. For all bincodes, the diagonal neighbors finding algorithm can be done in $O(n \log n)$ time with $O(n)$ memory space.

```
Algorithm : Northeast neighbors finding

Input: Bincodes $\langle B(1), B(2), ..., B(n)\rangle$ and the associated locations and levels.
Output: The northeast neighbors $\langle N_e(1), N_e(2), ..., N_e(n)\rangle$.

Step_1. \* Calculate the rightmost coverage and the potential east
           neighbor for each bincode. *\
        \* $Z(k)$ denotes the rightmost coverage of the $k$-th bincode and
           $B^*(k)$ denotes the potential northeast neighbor of the
           $k$-th bincode. *\
        For $k \leftarrow 1$ to $n$ do
           $Z(k) \leftarrow B(k) + (4^{2N-l(k)} - 1)$;
           $i^*(k) \leftarrow i(k) + 2^{N-\lceil \frac{l(k)}{2} \rceil}$; $j^*(k) \leftarrow j(k) + 2^{N-\lfloor \frac{l(k)}{2} \rfloor}$
           If $0 < i^*(k)$ and $i^*(k) \leq 2^N - 1$ and $0 < j^*(k)$ and $j^*(k) \leq 2^N - 1$
           then
              $B^*(k) \leftarrow \Sigma_{r=0}^{N-1}(i_r^* \times 2^{4r+3}) + \Sigma_{r=0}^{N-1}(j_r^* \times 2^{4r+1})$
                        $+ \Sigma_{r=0}^{2N-1}(1 \times 2^{2r})$
              \* $i^*(k) = (i_{N-1}^* i_{N-2}^* \cdots i_1^* i_0^*)_2$ and
                 $j^*(k) = (j_{N-1}^* j_{N-2}^* \cdots j_1^* j_0^*)_2$. *\
           else
              $B^*(k) \leftarrow -1$
        end

Step_2. \* Find the northeast neighbor for each bincode. *\
        For $k \leftarrow 1$ to $n$ do
           $N_e(k) \leftarrow -1$
        end
        For $k \leftarrow 1$ to $n - 1$ do
           If $B^*(k) \neq -1$
           then     \* binary search algorithm is used to find
                       the northeast neighbor. *\
              $a \leftarrow k + 1$; $b \leftarrow n$
              While $N_e(k) = -1$ and $a \leq b$ do
                 $k' \leftarrow \lfloor \frac{a+b}{2} \rfloor$
                 If $B^*(k) \geq B(k')$ and $B^*(k) \leq Z(k')$
                 then
                    If $i(k') = i^*(k)$ and $j(k') = j^*(k)$ then $N_e(k) \leftarrow B(k')$
                                                            else $a \leftarrow b + 1$
                 else
                    If $B^*(k) < B(k')$ then $b \leftarrow k' - 1$
                                        else $a \leftarrow k' + 1$
              end
        end
```

Fig. 10. The northeast neighbors finding algorithm.

5. CONCLUSIONS

Space minimization is an important consideration in image representation. According to the studies,[9,10] the IBB (bintree) is shown to be space utilization improvement from 0 to 25% over the linear quadtrees encoding method. Therefore, the IBB can be viewed as a competitive encoding structure. In this paper, some fast sequential algorithms for set operations, 4-neighbors finding, and diagonal neighbors finding of the IBB are presented. These algorithms are designed by using some important properties of bincodes, namely the increasing property and the covering property. More complicated image manipulations such as pattern matching on IBB's are our future research topics.

Acknowledgments—The authors would like to thank the reviewers for their valuable comments and thank Mr Yu-Wei Chen for his help for testing all the algorithms of this paper.

REFERENCES

1. H. Samet, *Applications of Spatial Data Structures*. Addison Wesley, New York (1990).
2. G. M. Hunter and K. Steiglitz, Operations on images using quad trees, *IEEE Trans. on Pattern Analysis and Machine Intelligence* **1**(2), 145–153 (1979).
3. H. Samet and M. Tamminen, Computing geometric properties of images represented by linear quadtrees, *IEEE Trans. on Pattern Analysis and Machine Intelligence* **7**, 229–240 (1985).
4. G. Schrack, Finding neighbors of equal size in linear

quadtrees and octrees in constant time, *CVGIP: Image Understanding* **55**(3), 221–230 (1992).

5. M. Tamminen, Comment on Quad- and Octrees', *Commu. ACM* **27**(3), 248–249 (1984).
6. C. Dyer, The space efficiency of quadtrees, *Comput. Graphics Image Process* **19**(4), 335–348 (1982).
7. I. Gargantini, An effective way to represent quadtrees, *Comm. of ACM* **25**(12), 905–910 (1982).
8. H. Samet, The quadtree and related hierarchical data structures, *Computing Survey* **16**(2), 187–260 (1984).
9. M. A. Ouksel and A. Yaagoub, The interpolation-based bintree and encoding of binary images, *CVGIP: Graphical Models and Image Processing* **54**(1), 75–81 (1992).
10. C. A. Shaffer, R. Juvvadi and L. S. Health, Generalized comparison of quadtree and bintree storage requirements, *Image and Vision Computing* **11**(7), 402–412 (1993).
11. K. Knowlton, Progressive transmission of gray-scale and binary pictures by simple, efficient and lossless encoding schemes, *Proc. IEEE* **68**, 885–896 (1990).
12. G. Brassard and P. Bratley, *Algorithmics: Theory and Practice.* Prentice-Hall, Englewood Cliffs, New Jersey (1988).

About the Author—CHI-YEN HUANG received the B.S. degree in industrial engineering from Chung Yuan Christian University and the M.S. degree in industrial management from the National Taiwan Institute of Technology. He now is a Ph.D. candidate in the Department of Information Management of the National Taiwan Institute of Technology. His current research interests include computer vision and parallel processing. He is a student member of the IEEE Computer Society.

About the Author—KUO-LIANG CHUNG received the B.S., M.S. and Ph.D. degrees in computer science and information engineering from National Taiwan University. He now is an Associate Professor in the Department of Information Management of the National Taiwan Institute of Technology. His current research interests include parallel and distributed computing, image and vision processing, matrix computations and computer graphics. He obtained the 1990 Outstanding Paper Award from the Computer Society of the Republic of China and the 1992 Outstanding Research Award from the National Science Council of the Republic of China. Dr Chung is a member of the IEEE Computer Society and the SIAM Society.