Pergamon

Pattern Recognition, Vol. 29, No. 9, pp. 1507–1518, 1996

Printed in Great Britain
0031-3203/96 $15.00 + .00

0031-3203(95)00167-0

FASTER NEIGHBOR FINDING ON IMAGES REPRESENTED BY BINCODES

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, Republic of China

(*Received* 8 *February* 1995; *in revised form* 13 *November* 1995; *received for publication* 5 *December* 1995)

Abstract—The Interpolation-Based Bintree (IBB) is a new encoding scheme for representing binary images and is storage-saving. In the IBB only the black leaves are saved, called bincodes, as the compressed codes for the image. In this paper a fast neighbor finding algorithm is presented on bincodes. Given a sequence of bincodes with size n, our algorithm is performed in $O((l_{max} - l_{min} + 1) \times n)$ time with the same memory complexity, where l_{max} and l_{min} are the maximal level and the minimal level, respectively, at which the given bincodes reside. Our algorithm is faster than the previous one [C.-Y. Huang and K.-L. Chung, *Pattern Recognition* **28**(3), 409–420 (1995)]. Experimental results for a practical version are included. The experimental values confirm our theoretical results. Copyright © 1996 Pattern Recognition Society. Published by Elsevier Science Ltd.

Bincode Interpolation-based bintree Linear bintree Neighbor finding Complexity analysis

1. INTRODUCTION

Neighbor finding is one of the key problems in image processing and pattern analysis. Based on the result of neighbor finding, many applications thus can be easily performed, e.g. connected component labeling, perimeter, Euler number, and so on. Given quadtrees or linear quadtrees, some efficient neighbor finding algorithms have been presented. A thorough survey is provided in reference (2).

The linear bintree is an alternate encoding scheme for representing images. It has been shown to be very simple and to save storage.[3,4] The Interpolation-Based Bintree (IBB) was first proposed by Ouksel and Yaagoub[3] by combining the features of some existing representations such as linear quadtrees, binary trees and interpolation-based codes. It is one of the linear bintree. The IBB is based on subdividing recursively the image into two equal-sized subimages, right and left or top and down, until the elements of the created subimages are either black or white, where each subimage is represented by a node. Following the subdivision process, the nodes corresponding to black subimages are converted into IBB codes. The final codes are called bincodes, which are unique integers assigned for black blocks.

Recently, Huang and Chung[1] presented fast algorithms such as intersection, union, complement and neighbor finding on bincodes. Given two sets of bincodes, B_1 and B_2, the intersection and union operation algorithms is performed in $O(n_1 + n_2)$ time, where n_1 is the size of B_1 and n_2 is the size of B_2; the complement operation for B_1 is performed in $O(n_1)$ time; the four-neighbor finding and the diagonal neighbor finding for B_1 is accomplished in $O(n_1 \log n_1)$ time.

In this paper we present a faster algorithm for neighbor finding on bincodes. Given a sequence of bincodes with size n, our algorithm is performed in $O((l_{max} - l_{min} + 1) \times n)$ time with the same memory complexity, where l_{max} and l_{min} are the maximal level and the minimal level, respectively, at which the given bincodes reside. Our algorithm is faster than the previous one.[1] Our algorithm can be applied to handle the three-dimensional (3-D) case. Experimental results for a practical version are included. The experimental values confirm our theoretical results.

2. PRELIMINARIES

Given a $2^N \times 2^N$ image, the root node of the bintree represents the whole image. If the root node is gray, two sons are added. Each son represents half of the image which is covered by his father. This process is repeated recursively for each son until a son is white or black. If a subdivision is either black or white, its corresponding node is an external node; otherwise, it is an internal node. A node at height h corresponds to a $2^{N-\lfloor h/2 \rfloor} \times 2^{N-\lceil h/2 \rceil}$ block. When h is even, the block is square; when h is odd, this block is rectangular. Due to the structure of bintrees, a $2^N \times 2^N$ image will have maximal height $2N$, 2^{2N} external nodes and $2^{2N} - 1$ internal nodes at most. Naturally, the bintree can be easily implemented by a pointer-type data structure.[5]

† Corresponding author. Email: klchung@cs.ntit. edu.tw.
This research was supported in part by the National Science Council of R.O.C. under contracts NSC85-2213-E011-099 and NSC85-2121-M011-002.

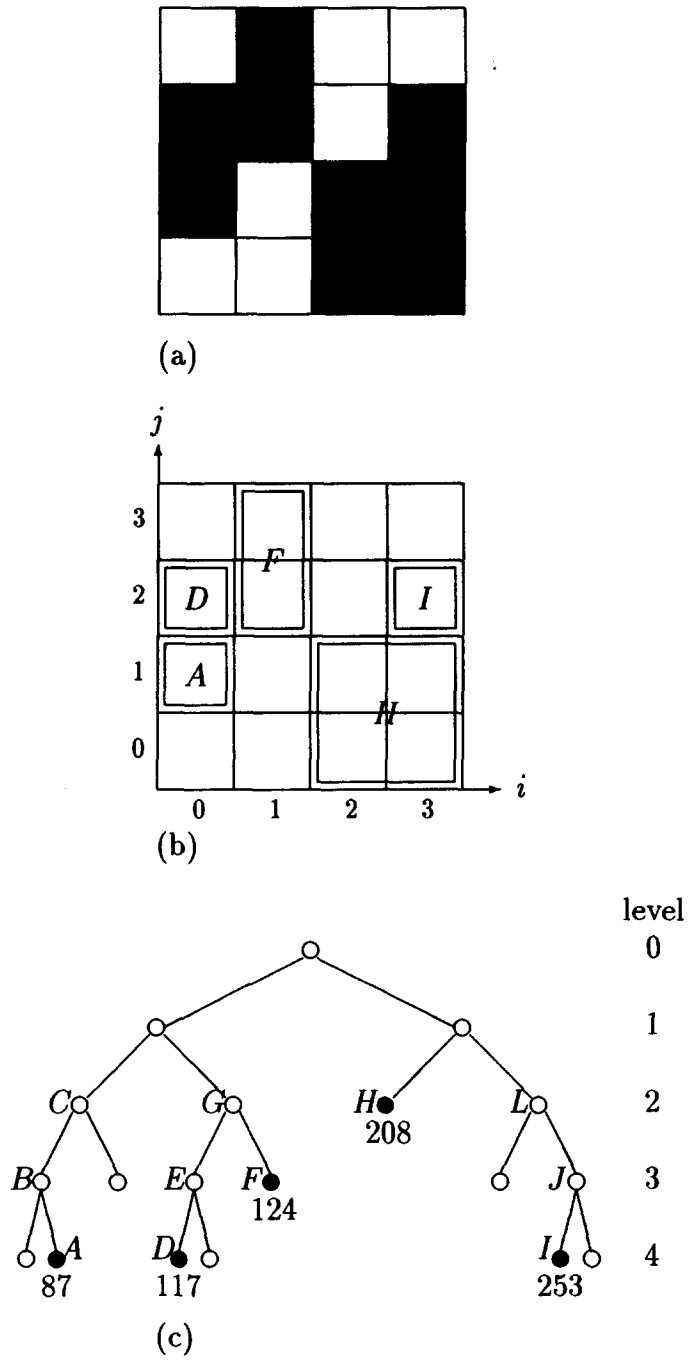

Fig. 1. A $2^2 \times 2^2$ binary image. (a) Binary image. (b) Blocks example. (c) Bintree and bincodes of (a).

Considering the $2^2 \times 2^2$ binary image as shown in Fig. 1(a), the corresponding blocks are shown in Fig. 1(b) and the bintree is illustrated in Fig. 1(c).

The IBB is based on the bintree structure and represents the image as an ordered collection of external black nodes. Each external black node is described by a numerical record. Given a $2^N \times 2^N$ binary image, let a node at level l of a bintree correspond to a black block in the binary image at location (i, j), then the bincode Q is derived as follows:[3]

(1) Convert i and j to binary strings $i_{N-1}i_{N-2}\ldots\ldots i_1i_0$ and $j_{N-1}j_{N-2}\ldots\ldots j_1j_0$, 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 convert it to a binary string $s_{2N-1}s_{2N-2}\ldots s_1 s_0$, where $s = \sum_{k=0}^{2N-1}(s_k \times 2^k)$.

(3) The bincode is given by $Q = \sum_{k=0}^{N-1}(i_k \times 2^{4k+3}) + \sum_{k=0}^{N-1}(j_k \times 2^{4k+1}) + \sum_{k=0}^{2N-1}(s_k \times 2^{2k})$.

Here, i is termed the column_index and j is called the row_index.

For example, block A in Fig. 1 is at location (0, 1) and at level 4. According to the above bincode conversion scheme, we obtain $i = 0 = (i_1 i_0)_2 = (00)_2$, $j = 1 = (j_1 j_0)_2 = (01)_2$ and $s = 2^4 - 2^0 = 15 = (s_3 s_2 s_1 s_0)_2 = (1111)_2$, then the bincode of block A is $(i_1 s_3 j_1 s_2 i_0 s_1 j_0 s_0)_2 = (01010111)_2 = 87$. We traverse black nodes of the bintree in preorder and simultaneously calculate its bincodes, then bincodes of Fig. 1(a) are represented by an ordered and increasing sequence of numerical records $\langle 87, 117, 124, 208, 253\rangle$.

3. NEIGHBOR FINDING ON BINCODES

Following the definition used by Samet and Tamminen[6] we define the black block Q to be a four-neighbor of the black block P if Q and P share a common edge and the size of Q is equal to or larger than that of P and the black block R to be a diagonal neighbor of the black block P if R and P share a common vertex. As shown in Fig. 1(b), block D is the north neighbor of block A and block F is the east neighbor of D, but D is not the west neighbor of F because the size of D is smaller than that of F. Further, block F is the northeast neighbor of block A; block A is the southwest neighbor of block F.

In this section, we only focus on the four-neighbor finding. However, the diagonal neighbor finding can be easily derived by the same way. For simplicity, we assume that the input data is a sequence of bincodes and the associated levels and locations.

3.1. *Find bincodes of equal-sized adjacent blocks*

In this subsection the method for finding the bincodes of equal-sized adjacent blocks of a given bincode is presented. Following the same notations used in reference,[7] let the symbols $\wedge$, $|$, $\ll$ and $'$ be bitwise operations for *anding*, *oring*, *shifting to left* and *complementing*, respectively. Assume any one of these four operations needs one unit time. In addition, let $t_u = \sum_{k=0}^{N-1}(2^{4k} + 2^{4k+1} + 2^{4k+2}) = (\underbrace{01110111\ldots0111}_{4N})_2$, $t_v = \sum_{k=0}^{N-1}(2^{4k} + 2^{4k+2} + 2^{4k+3}) = (\underbrace{11011101\ldots1101}_{4N})_2$ and $\Delta m = (\underbrace{000\ldots001}_{4N})_2$.

The bincode of the east equal-sized adjacent block of the bincode Q can be obtained in $O(1)$ time and is shown below.

Theorem 1. Given a bincode Q at level l, the bincode of the east equal-sized adjacent block, say Q_e, is given by $(((Q|t_u) + (\Delta m \ll (4[(2N - l))/2] + 3))) \wedge t'_u)|(Q \wedge t_u)$ and it takes $O(1)$ time.

Proof. Let Q be a bincode at level l and location (i, j), we have that Q_e is at level l and location (i^*, j), where $i^* = i + 2^{[(2N-l)/2]}$. $Q|t_u$ preserves the binary representation of column_index of Q and sets the other bits to be 1. $\Delta m \ll (4\lfloor 2N - l/2\rfloor + 3)$ equals to $(00\ldots0\ \underbrace{100\ldots0}_{4(N-[l/2])+4})_2$; $(Q|t_u) + (\Delta m \ll (4[(2N - l)/2] + 3))$ equals to $(i^*_{N-1}k_{3N-1} \times k_{3N-2}k_{3N-3}i^*_{N-2}k_{3N-4}k_{3N-5}k_{3N-6}\ldots i^*_{N-\lfloor l/2\rfloor}\ldots 0k_2k_1k_0)_2$, where $i^* = (i^*_{N-1}i^*_{N-2}\ldots i^*_{N-\lfloor l/2\rfloor}00\ldots0)_2$ and $k_x \in \{0, 1\}$ for $0 \le x \le 3N-1$; $(((Q|t_u) + (\Delta m \ll (4[(2N - l)/2] + 3))) \wedge t'_u$ extracts the binary representation of column_index of Q_e and sets the other bits to be 0; $Q \wedge t_u$ preserves the other bits excepting the binary representation of column_index of Q_e. Since only a few bitwise operations are needed, it takes $O(1)$ time. □

Return to Fig. 1, the east equal-sized adjacent block of D is the lower half of F. From $D = 117 = (01110101)_2$, $t_u = (01110111)_2$, $\Delta m = (00000001)_2$ and $4[(2N - l)/2] + 3 = 3$, we have $D|t_u = (01110111)_2$ and $(D|t_u) + (\Delta m \ll (4[(2N - l)/2] + 3)) = (01111111)_2$, then $((D|t_u) + (\Delta m \ll (4[(2N - l)/2] + 3))) \wedge t'_u = (00001000)_2$. From $D \wedge t_u = (01110101)_2$, it follows that $(((D|t_u) + (\Delta m \ll (4[(2N - l)/2] + 3))) \wedge t'_u)|(D \wedge t_u) = (01111101)_2 = 125$, which is the bincode of the east equal-sized adjacent block of D, i.e. the lower half of F.

Given a bincode, the bincodes of the corresponding west, south and north equal-sized adjacent blocks can be obtained by the same way.

Corollary 1. Given a bincode Q at level l, the bincode of the west equal-sized adjacent block is given by $(((Q \wedge t'_u) - (\Delta m \ll (4[(2N - l)/2] + 3))) \wedge t'_u)|(Q \wedge t_u)$, and it takes $O(1)$ time; the bincode of the south equal-sized adjacent block is given by $(((Q \wedge t'_v) + (\Delta m \ll (4[(2N - l)/2] + 1))) \wedge t'_v)|(Q \wedge t_v)$; the bincode of the north equal-sized adjacent block is given by $(((Q|t_v) + (\Delta m \ll (4[(2N - l)/2] + 1))) \wedge t'_v)|(Q \wedge t_v)$.

3.2. *Find four-neighbors for bincodes*

We now illustrate our concept of four-neighbor finding. For easy description, we define that given a block A and a black block B; B is termed the quasi-neighbor of A if A and B share a common edge and the size of B is equal to or larger than that of A.

As shown in Fig. 2, we first define the black block H to be the east equal-sized quasi-neighbor of block M. If M is black, H is also the east neighbor of M; if M is white, H is not the east neighbor of M. Since M is gray, we search the descendants of M further. By the top–down approach, we see that H is the east neighbor of U. Given a bincode, its all ancestors whose levels are larger than or equal to $l_{\min}$ are defined to be the possible candidates of the bincode, where $l_{\min}$ is the minimal level at which the given bincodes reside. Here, all ancestors of one bincode include the bincode itself. Consider the searching path, M is the first possible candidate; R is the second possible candidate; U is the exact possible candidate since it is black.

3.2.1. *Obtain possible candidates.* Return to Fig. 1. Since $l_{\min} = 2$, the possible candidates of A are A, B and

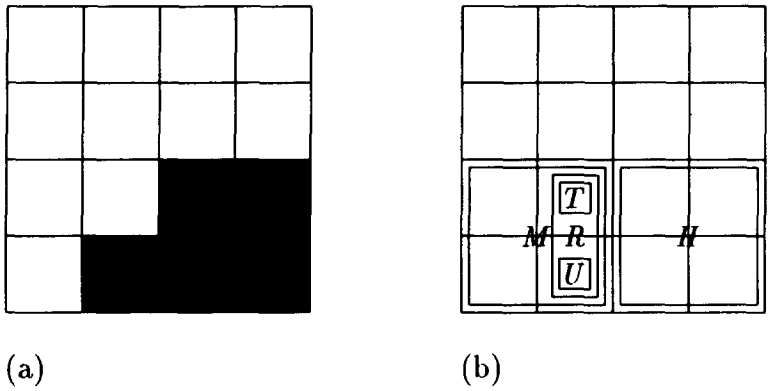

Fig. 2. Another $2^2 \times 2^2$ binary image. (a) Binary image. (b) Blocks example.

C; the possible candidates of D are D, E and G; the possible candidates of F are F and G; the possible candidates of H is H itself. Therefore, all possible candidates of Fig. 1 are A, B, C, D, E, G, F, G, H, I, J and L. It is observed that some possible candidates are redundant, for example, G is the common possible candidate of D and F. We need the following two lemmas in order to obtain all possible candidates without redundancy.

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

For convenience, $Q + (4^{2N-l} - 1)$ is termed the rightmost coverage of bincode Q.

Lemma 2.[3] 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 the right son of Q, respectively, then $B = Q + 2^{2(2N-l-1)}$ and $C = Q + 3 \times 2^{2(2N-l-1)}$, where l is the level of Q.

Given a sequence of bincodes, the method for obtaining all possible candidates without redundancy is: for the first bincode, by using Lemma 1 and 2, we calculate its all ancestors in a bottom-up manner until the father of one ancestor covers the next bincode or the level of one ancestor is equal to l_{min}, then the next given bincode is processed successively. From the first given bincode 87 in Fig. 1 and the second bincode 117, by Lemma 1, the first bincode 87 does not cover the second bincode 117 because the rightmost coverage of 87 is 87 $[= 87 + (4^{(4-4)} - 1)]$. Thus, bincode 87 is a possible candidate. By Lemma 2, the father of bincode 87 is bincode 84 $[87 = 84 + 3 \times 2^{(2(4-3-1))}]$ at level 3 $(= 4 - 1)$. The rightmost coverage of 84 is 87 $[= 84 + (4^{(4-3)} - 1)]$. Since it does not cover bincode 117, bincode 84 is also a possible candidate. The father of bincode 84 is bincode 80 $[84 = 80 + 2^{(2(4-2-1))}]$ at level 2. The rightmost coverage is 95 $[= 80 + (4^{(4-2)} - 1)]$, so bincode 80 is also a possible candidate. Since bincode 80 is at level 2 $(= l_{min})$, we check the second given bincode 117 and the third bincode 124, subsequentially. By the same arguments, we have that all possible candidates of Fig. 1 are bincodes 87, 84, 80, 117, 116, 124, 112, 208, 253, 252 and 240, which correspond to nodes A, B, C, D, E, F, G, H, I, J and L, respectively. Note that the black possible candidates are the given bincodes.

It is observed that given a sequence of bincodes of size n with respect to an $m \times m$ image, if all given bincodes are at the same level, the size of all possible candidates is $O(n)$; in the worst case (see Fig. 3), the

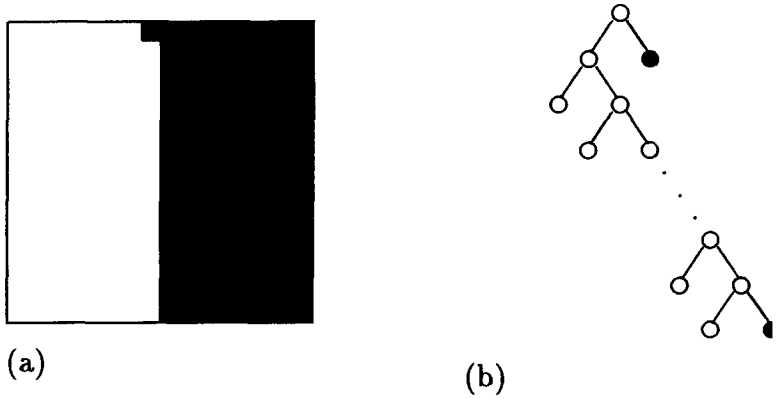

Fig. 3. The worst case example. (a) Binary image. (b) The corresponding bintree.

size of all possible candidates is $O(n \times \log m)$. In general, the size of all possible candidates is $O((l_{max} - l_{min} + 1) \times n)$. Consider the corresponding bintree of one image, when the size of the black leaf nodes is equal to or larger than that of the white leaf nodes, it follows that the size of all possible candidates will be less than or equal to $3 \times n$. Upon finding a possible candidate, the next possible candidate can be obtained in $O(1)$ time, so we have the following result.

Lemma 3. Finding all possible candidates can be performed in $O((l_{max} - l_{min} + 1) \times n)$ time.

3.2.2. *Construct hashing table.* We now want to construct a hashing table for storing these possible candidates in order to perform four-neighbor finding efficiently.

Given a $2^N \times 2^N$ image and a hashing table with bucket size $S(=4^K$, where K is an integer, $1 \leq K \leq 4$ and $K < N)$, our bucket-allocation strategy is described as follows. Let the first bucket contain $S-1$ possible bincodes and each of the others contain S possible bincodes. The possible bincodes at levels above $2K$ are allocated to the first bucket since there are $2^{2K} - 1$ $(=S-1)$ possible candidates at most. The next two buckets contain the possible candidates at level $2K$ because it is of size 2^{2K+1} $(=2S)$ at most. Continuing this way, we see that the possible bincodes in a bucket are all located at the same level except those in the first bucket. Given a $2^2 \times 2^2$ binary image, all possible bincodes are shown in Fig. 4(a), where the number incident to the node is the bincode and its corresponding column_index and row_index. Suppose the size of the bucket is 4, we first assign bincodes 0, 64

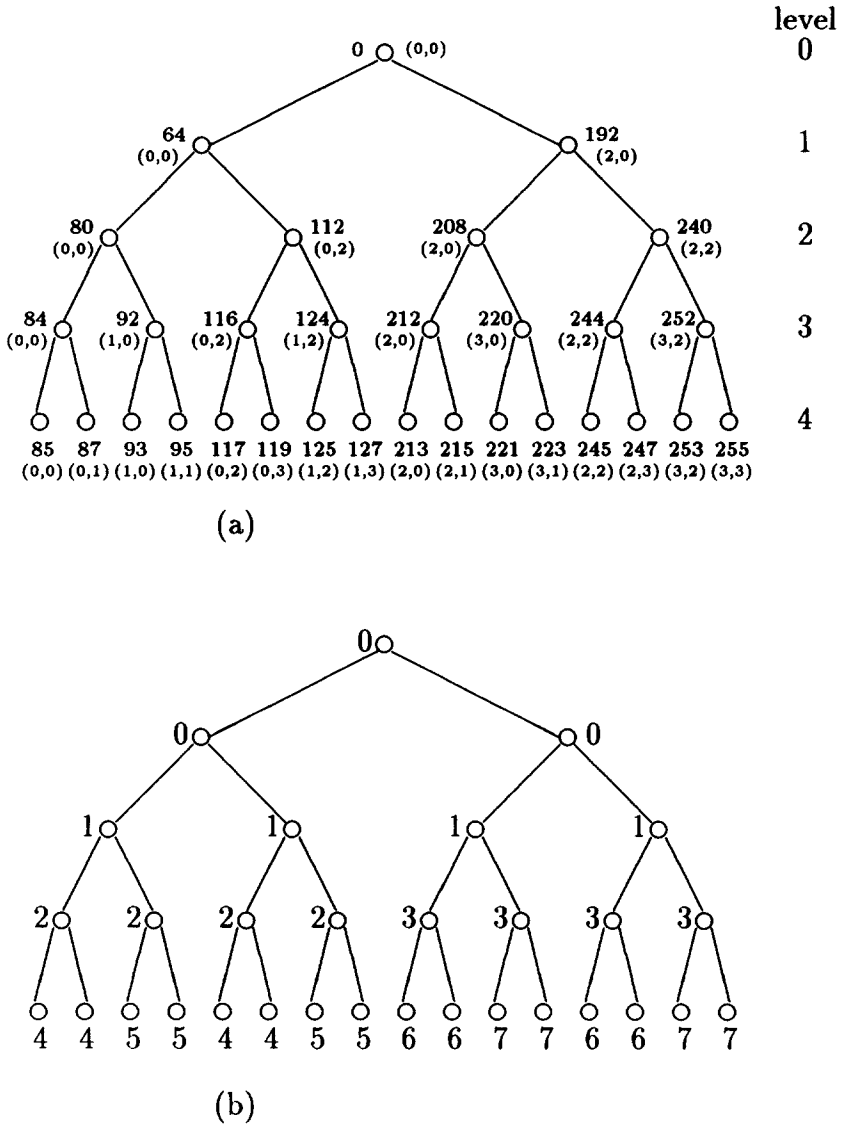

Fig.4. A bucket-allocation example. (a) All possible bincodes of a $2^2 \times 2^2$ binary image. (b) Associated bucket numbers when the size of the bucket is 4.

and 192 to bucket 0. At level 2, the bincodes 80, 112, 208 and 240 are assigned to bucket 1. At level 3, the bincodes 84, 92, 116 and 124 are assigned to bucket 2; bincodes 212, 220, 244 and 252 are assigned to bucket 3. As shown in Fig. 4(b), the number of incident to the node is the corresponding bucket number. Return to Fig. 1(c). Since it does not have any possible candidate at level 0 and level 1, C, G, H, L, B, E, F, J, A, D and I are allocated to bucket 0, 0, 0, 0, 1, 1, 1, 2, 3, 3 and 6, respectively.

Let P be a bincode at level $l_p (l_p \geq l_{\min})$ and location (i_p, j_p). Given a hashing table, each bucket with size S, following the previous bucket-allocation strategy, the bucket number of P, b_p, can be obtained by the following hash function:

$$b_p = 0 \quad \text{for } l_p < 2K;$$

$$b_p = 2^{l_p - 2K} + \lfloor i_p \times 2^{l_p - N - 2K} \rfloor + \lfloor j_p \times 2^{(l_p)/2 - N - 2K} \rfloor - \lfloor 2^{l_{\min} - 2K} \rfloor \quad \text{for } l_p \geq 2K \text{ and } l_p \text{ is even;}$$

$$b_p = 2^{l_p - 2K} + \lfloor i_p \times 2^{l_p - N - 2K} \rfloor + \lfloor j_p \times 2^{(l_p - 1/2) - N - 2K} \rfloor - \lfloor 2^{l_{\min} - 2K} \rfloor \quad \text{for } l_p \geq 2K \text{ and } l_p \text{ is odd.}$$

Naturally, the bincodes at levels smaller than $2K$ are allocated to bucket 0. We next consider the bincodes at levels larger than or equal to $2K$, that is, $l_p \geq 2K$. When l_p is odd (even), because there are $(2^{l_p} - 1) - (2^{2K} - 1)((2^{l_p} - 1) - (2^{2K} - 1))$ bincodes at levels smaller than l_p and larger than or equal to $2K$, the leading bucket number at level l_p is $2^{l_p - 2K}$ $(=((2^{l_p} - 1) - (2^{2K} - 1))/4^k) + 1)(2^{l_p - 2K}$ $(=((2^{l_p} - 1) - (2^{2K} - 1))/4^k) + 1)$. The possible bin- codes at level l_p are allocated to buckets by the order of column_index major first then row_index major. $[(i_p/2^{(2N - l_p)/2}) \times 2^{l_p/2}/4^K]$ $(= [i_p \times 2^{l_p - N - 2K}])([(i_p/2^{(2N - (l_p + 1))/2}) \times 2^{(l_p - 1)/2}/4^K]$ $(= [i_p \times 2^{l_p - N - 2K}]))$ calculates the number of buckets used at the column_indices smaller than i_p at level l_p; $[(j_p/2^{(2N - l_p)/2})/4^K]$ $(= [j_p \times 2^{(l_p/2) - N - 2K}])$ $([(j_p/2^{(2N - l_p - 1)/2}/4^K)](= [j_p \times 2^{(l_p - 1)/2 - N) - 2K}]))$ calculates the bucket num- ber which is the row_index j_p with to column_index i_p. The buckets occupied by the possible bincodes at levels smaller than level $l_{\min}$ will not be used if $l_{\min} \geq 2K$ so we minus $[2^{l_{\min} - 2K}]([2^{l_{\min} - 2K}])$.

In our hash function, the bucket number of a possible candidate is obtained by its level, column_index and row_index. From the bincode conversion scheme, the level, column_index and row_index of a bincode can be extracted from the bincode in $O(N)$ time. It follows that the hashing table for storing all possible candidates can be constructed in a total of $O((l_{\max} - l_{\min} + 1) \times n \times N)$ time. We now improve the time complexity from $O((l_{\max} - l_{\min} + 1) \times n \times N)$ to $O((l_{\max} - l_{\min} + 1) \times n)$.

Given a bincode Q at level l and location (i, j), if bincode P is the father of Q, the level l_p and location (i_p, j_p) of P can be obtained by the following formulas in $O(1)$ time:

$$l_p = l - 1;$$

$i_p = i$ and $j_p = j$ if Q is the left son of P;

$i_p = i$ and $j_p = j - 2^{(2N - l)/2}$ if l is even and Q is the right son of P;

$i_p = i - 2^{(2N - (l+1))/2}$ and $j_p = j$ if l is odd and Q is the right son of P.

On the contrary, given a bincode P at level l_p and location (i_p, j_p), the level l and location (i, j) of Q can be obtained in a similar way if it is known when Q is the right son of P or the left son of P.

Recall that the possible candidates of Fig. 1 are 87, 84, 80, 117, 116, 124, 112, 208, 253, 252 and 240. Since the possible candidate 87 is at level 4 and location (0, 1), by our hash function, the possible candidate 87 is allocated to bucket 3 $(b_p = 2^{4-2} + \lfloor 0 \times 2^{4-2-2} \rfloor + \lfloor 1 \times 2^{2-2-2} \rfloor - \lfloor 2^{2-2} \rfloor = 4 + 0 + 0 - 1 = 3)$. Next, the possible candidate 84 is at level 3 and location (0, 0) $(l_p = l - 1 = 4 - 1 = 3,\ i_p = i = 0$ and $j_p = j - 2^{(2N-l)/2} = 1 - 2^0 = 0)$. Thus, it is allocated to bucket 1 $(b_p = 2^{3-2} + [0 \times 2^{3-2-2}] + \lfloor 0 \times 2^{(3-1)/2-2-2} \rfloor - \lfloor 2^{2-2} \rfloor = 2 + 0 + 0 - 1 = 1)$. The possible candidate 80 is allocated to bucket 0 because it is at level 2 and location (0, 0) $(l_p = l - 1 = 3 - 1 = 2,\ i_p = i = 9$ and $j_p = j = 0$, then $b_p = 2^{2-2} + \lfloor 0 \times 2^{2-2-2} \rfloor + \lfloor 0 \times 2^{(2/2)-2-2} \rfloor - \lfloor 2^{2-2} \rfloor = 1 + 0 + 0 - 1 = 0)$. By the same way, all the remaining possible candidates will be assigned to the proper buckets. The constructed hashing table maintained by pointers is illustrated in Fig. 5, where the alphabets shown in the right upper corners, namely, A, B, C, ..., and L, denote the corresponding nodes in the bintree of Fig. 1(c); four fields at the bottom of each rectangular record are the east, west, south and north quasi-neighbors of that possible candidate, respectively, which will be discussed later. For saving space, the associated levels and locations of the possible candidate are omitted in the figure. In order to increase the utilization of memory, all buckets being empty (e.g. buckets 4 and 5 in Fig. 5) could be marked out and allocated to other possible candidates.

Based on this improved version, we have the following lemma.

Lemma 4. Constructing a hashing table for storing all possible candidates can be accomplished in $O((l_{\max} - l_{\min} + 1) \times n)$ time.

3.2.3. *Obtain equal-sized quasi-neighbors.* After all possible candidates have been allocated in the hashing table, we want to find equal-sized quasi-neighbors of each possible candidate. We now calculate the bincode of the east equal-sized adjacent block of the first given bincode 87. From $Q = 87 = (01010111)_2$, $t_u = (01110111)_2$, $\Delta m = (00000001)_2$, and $4\lfloor (2N - l)/2 \rfloor + 3 = 3$, by Theorem 1, we have $Q|t_u = (01110111)_2$, $(Q|t_u) + (\Delta m \ll (4\lfloor (2N - l)/2 \rfloor + 3)) = (01111111)_2$, then $((Q|t_u) + (\Delta m \ll (4\lfloor (2N - l)/2 \rfloor + 3))) \wedge t'_u = (00001000)_2$. From $Q \wedge t_u = (01010111)_2$, we have $(((Q|t_u) + (\Delta m \ll (4\lfloor (2N - l)/2 \rfloor + 3))) \wedge t'_u)|(Q \wedge t_u) = (01011111)_2 = 95$. Thus, bincode 95 is the east equal-sized adjacent block of bincode 87. On the other hand,

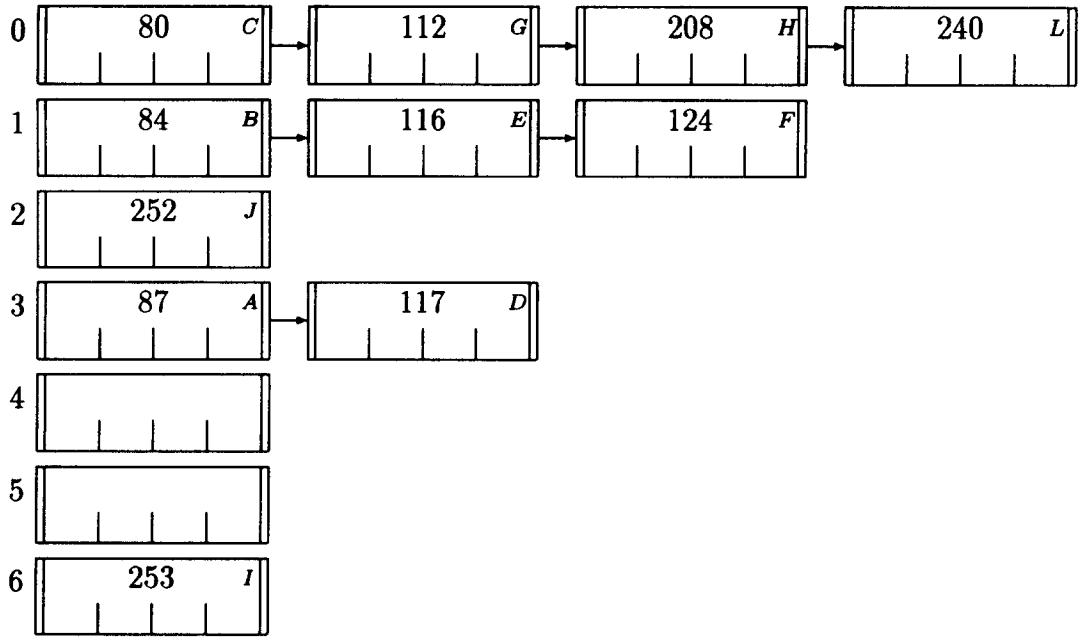

Fig. 5. The constructed hashing table of Fig. 1.

we have that bincode 87 is the west equal-sized quasi-neighbor of bincode 95.

The level, column_index and row_index of one equal-sized adjacent block of the given bincode can be obtained by using the following method in $O(1)$ time.

Given a bincode Q at level l and location (i, j), let l_e, l_w, l_s and l_w be the levels of the east, west, south and north equal-sized adjacent blocks of Q, respectively; (i_e, j_e), (i_w, j_w), (i_s, j_s) and (i_n, j_n) be the locations of the east, west, south and north equal-sized adjacent blocks, respectively. It is not hard to derive $l_e = l_w = l_s = l_n = l$, $i_e = i + 2^{\lfloor (2N-l)/2 \rfloor}$, $i_w = i - 2^{\lfloor (2N-l)/2 \rfloor}$, $i_s = i_n = i$, $j_e = j_w = j$, $j_s = j - 2^{\lfloor (2N-L)/2 \rfloor}$, and $j_n = 2^{\lfloor (2N-l)/2 \rfloor}$.

Using the above formulas, the level of the bincode 95 is 4 and the location is (1, 1)($l_e = l = 4$, $i_e = i + 2^{\lfloor (2N-l)/2 \rfloor} = 0 + 2^0 = 1$ and $j_e = j = 1$). Using the hash function, $b_p = 2^{4-2} + [1 \times 2^{4-2-2}] + \lfloor 1 \times 2^{2-2-2} \rfloor - \lfloor 2^{2-2} \rfloor = 4 + 1 + 0 - 1 = 4$. We see that bincode 95 will be in bucket 4 if it exists, that is, bincode 95 is a possible candidate. We ignore it because bincode 95 does not exist in bucket 4. The given bincode 87 does not have the west equal-sized adjacent block because the column_index i_w is out of the boundary of the image ($i_w = i - 2^{\lfloor (2N-l)/2 \rfloor} = 0 - 2^0 = -1$). By Corollary 2, we have $t'_v = (00100010)_2$, $Q \wedge t'_v = (00000010)_2$, $(Q \wedge t'_v) - (\Delta m \ll (4 \lfloor (2N - l)/2 \rfloor + 1)) = (00000000)_2$, then $((Q \wedge t'_v) - (\Delta m \ll (4\lfloor (2N - l)/2 \rfloor + 1))) \wedge t'_v = (00000000)_2$. Further, we have $(((Q \wedge t'_v) - (\Delta m \ll (4\lfloor (2N - l)/2 \rfloor + 1))) \wedge t'_v) | Q \wedge t_v) = (01010101)_2 = 85$ since $Q \wedge t_v = (01010101)_2$. The given bincode 87 would be assigned be the north equal-sized quasi-neighbor of bincode 85. We ignore bincode 85 because it is not a possible candidate. By Corollary 1, we have that bincode 117 is the north equal-sized adjacent block of the given bincode 87. The bincode 117 is at level 4 and location (0, 2)($l_n = l = 4$, $i_n = i = 0$ and $j_n = j + 2^{\lfloor (2N-l)/2 \rfloor} = 1 + 1 = 2$), then $b_p = 2^{4-2} + \lfloor 0 \times 2^{4-2-2} \rfloor + \lfloor 2 \times 2^{2-2-2} \rfloor - \lfloor 2^{2-2} \rfloor = 4 + 0 + 0 - 1 = 3$.. As shown in the second record of bucket 3 in Fig. 6, 87, is assigned to be the south equal-sized quasi-neighbor of 117. After finding all the equal-sized quasi-neighbors, the updated hashing table is illustrated in Fig. 6. We have that the possible candidate 80 has an east quasi-neighbor 208; the possible candidate 240 has a south quasi-neighbor 208; the possible candidate 116 has an east quasi-neighbor 124;...and so on.

Since it takes $O(1)$ time to find the equal-sized adjacent block of a given bincode and the corresponding bucket number of this equal-sized adjacent block in each direction, we have the following result.

Lemma 5. Finding all equal-sized quasi-neighbors for all possible candidates can be performed in $O(n)$ time.

3.2.4. *Rearrange possible candidates in the first bucket.* Recall that based on our bucket-allocation strategy described above, the possible bincodes in a bucket are all located at the same level, except those in the first bucket. In addition, supposing x and y be two integers and $y > x > 0$, the corresponding sub-images of the possible bincodes in bucket x are larger than or equal to that of the possible bincodes in bucket y. Therefore, before performing the top–down ap-

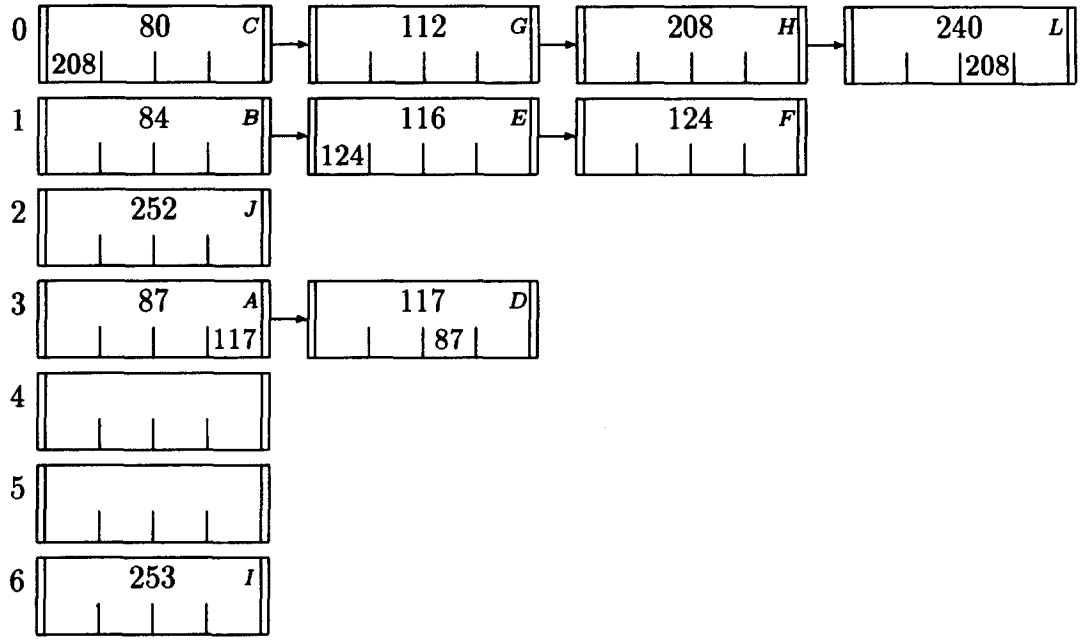

Fig. 6. The hashing table of Fig. 1 after finding all the equal-sized quasi-neighbors.

proach, we should rearrange the possible candidates in the first bucket, bucket 0 when $l_{\min} < 2K$. This rearrangement can be accomplished easily by using the quicksort methods.[8]

3.2.5. *Downloading.* For each gray possible candidate which has the equal-sized quasi-neighbor, we next want to search its descendants adjacent to its equal-sized quasi-neighbors. Our method for searching the proper descendants is demonstrated by an example of Fig. 2. For the gray block M, we first obtain that M has an equal-sized quasi-neighbor H, then this message being that H is a quasi-neighbor is downloaded to the proper sons of M, say R. That is, R has a quasi-neighbor H. Since R is still gray, the meassage is downloaded further to his proper sons, say T and U. We bypass T, because it is white and have that U has a neighbor H.

Now, downloading can be performed from the first possible candidate of the first bucket one by one until all gray possible candidates are processed. For each gray possible candidate, the rules are described as follows:

(1) The east (west) quasi-neighbor information is downloaded to its right (left) son if the level of the gray possible candidate is even and is downloaded to its two sons if the level of the gray possible candidate is odd.

(2) The south (north) quasi-neighbor information is downloaded to its two sons if the level of the gray possible candidate is even and is downloaded to its left (right) son if the level of the gray possible candidate is odd.

As shown in Fig. 6, for the first gray possible candidate 80, by Rule (1), its east quasi-neighbor information 208 is downloaded to the right son 92 since by Lemma 2, $92 = 80 + 3 \times 2^{2(4-2-1)}$. The bincode 92 is at level 3 and position (1, 0) ($l = l_p + 1 = 2 + 1 = 3$, $i = i_p + 2^{(2N-(l+1))/2} = 0 + 2^0 = 1$ and $j = j_p = 0$). We ignore it because 92 (not a possible candidate) does not exist in bucket 1 ($b_p = 2^{3-2} + \lfloor 1 \times 2^{3-2-2} \rfloor + \lfloor 0 \times {}^{(3-1)/2-2-2} \rfloor - \lfloor 2^{2-2} \rfloor = 2 + 0 + 0 - 1 = 1$). We bypass the second gray possible candidate 112, since it does not have any neighbor. For the third gray possible candidate 240, based on Rule (2), its south quasi-neighbor information 208 is downloaded to 244 and 252 ($244 = 240 + 2^{2(4-2-1)}$ and $252 = 240 + 3 \times 2^{2(4-2-1)}$). Thus, 208 is assigned to be the south quasi-neighbor of 252 (see the first record of bucket 2 in Fig. 7). Nevertheless, 244 is ignored because it does not exist in bucket 2. For 116, its east quasi-neighbor information 124 is assigned to be the east quasi-neighbor of possible candidate 117 at bucket 3. For 252, its south quasi-neighbor information 208, which was just described, is assigned to be the south quasi-neighbor of possible candidate 253 at bucket 6. After downloading all quasi-neighbors information, the updated hashing table is illustrated in Fig. 7 and we have the following result.

Lemma 6. The downloading work can be performed in $O((l_{\max} - l_{\min} + 1) \times n)$ time.

Up to here, all black neighbors of the given bincodes have been found. From Fig. 7, we see that the bincode 87 has a north neighbor 117; the bincode 117 has an east neighbor 124 and a south neighbor 87; the bincode 253 has a south neighbor 208.

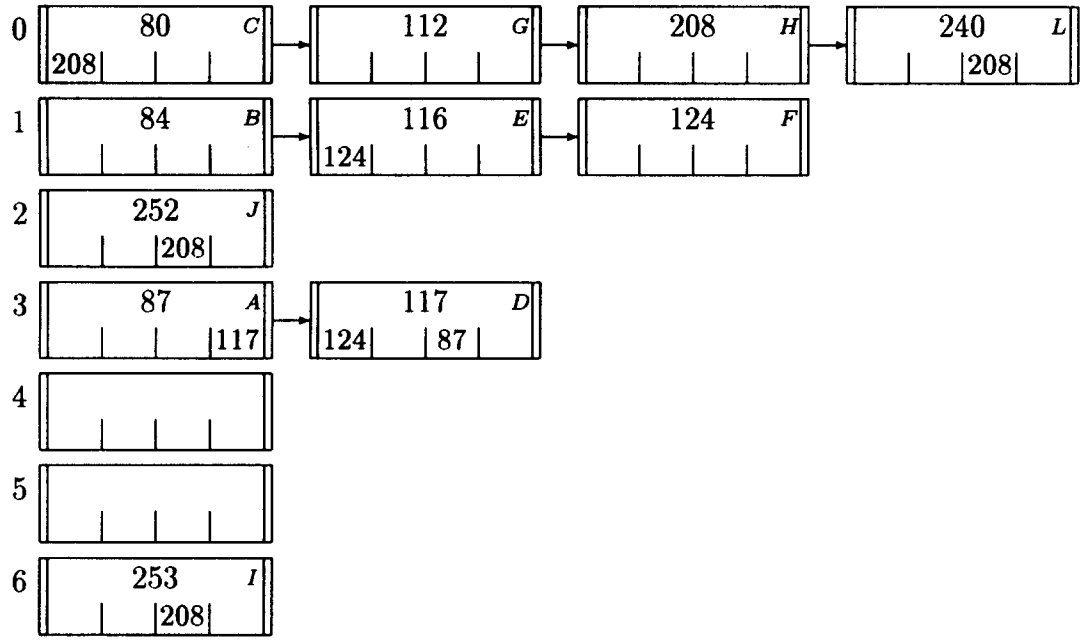

Fig.7. The updated hashing table of Fig. 1 after downloading quasi-neighbors.

For all given bincodes, our algorithm for finding all black four-neighbors is described as follows.

Algorithm. Four-neighbor finding
Input: Given bincodes with levels and locations.
Output: The four-neighbors.

Step 1. Finding all possible candidates.
Step 2. Constructing a hashing table for storing all possible candidates.
Step 3. In the hashing table, for each direction, calculating equal-sized adjacent blocks, say Xs, of each given bincode, say Y. Y is assigned to be the equal-sized quasi-neighbor of Xs.
Step 4. When $l_{\min} < 2K$, rearranging the sequence of the possible candidates of the first bucket to be a decreasing sequence based on the sizes of their corresponding subimages.
Step 5. From the first bucket, for each gray possible andidate which has the neighbor, downloading its quasi-neighbor information to his proper descendants.

By Lemma 3, Step 1 takes $O((l_{\max} - l_{\min} + 1) \times n)$ time. By Lemma 4, Step 2 takes $O((l_{\max} - l_{\min} + 1) \times n)$ time. By Lemma 5, Step 3 takes $O(n)$ time . By using the quicksort method, Step 4 can be done in $O(S \log S) = O(1)$ time since S is specified to be a constant. By Lemma 6, Step 5 takes $O((l_{\max} - l_{\min} + 1) \times n)$ time. The total complexity is shown below.

Theorem 2. Given a sequence of bincodes of size n, our algorithm for the four-neighbor finding can be accomplished in $O((l_{\max} - l_{\min} + 1) \times n)$ time with memory size $O((l_{\max} - l_{\min} + 1) \times n)$.

Following the definition of the diagonal neighbor described above, by modifying our algorithm slightly, the diagonal neighbors can be found easily in terms of the same time complexity.

4. EXPERIMENTATIONS

In order to gain more insight into the performance of our algorithm, some experimental results for a practical version are included. We first provide five sequences of bincodes with respect to five $2^8 \times 2^8$ binary images. These bincodes are generated randomly. Our programs are coded in *C-compiler* programming language and are executed on a SUN/SPARC-2 workstation.

Let the size of the bucket be 16 ($=4^2$). The experimental results are shown in Table 1, where "sec" denotes second; (b)/(a) denotes the ratio of the time spent in the four-neighbors finding to the size of the possible candidates. From Table 1, it is observed that the execution time of the four-neighbor finding is almost linearly proportional to the size of possible candidates. These results confirm our theoretical analysis. Next, we take two maps shown in Figs 8 and 9, respectively, to evaluate our algorithm. Figure 8 is the floodplain map and Fig. 9 is the Taiwan map. Both images are of resolution $2^8 \times 2^8$. Similarly, the size of the bucket is set to be 16 ($=4^2$). The experimental results are shown in Table 2. The results also confirm our theoretical analysis.

Table 1. Experimental results for random images

The size of given bincodes	(a) The size of possible candidates	(b) The execution time for four-neighbor finding (s)	(b)/(a)
2720	8019	1.12	0.000140
4911	15765	2.19	0.000139
4613	14841	2.02	0.000136
4968	14982	2.13	0.000142
3925	13031	1.79	0.000137

Fig. 8. Floodplain map.

5. DISCUSSIONS AND CONCLUSIONS

In this paper, we assume that the input data of our algorithm is a sequence of bincodes and the associated levels and locations. However, the associated levels and locations can be easily extracted from bincodes if the input data only has bincodes. This extraction work needs $O(N \times n)$ time. In this case, the total time complexity is $O(N \times n)$ since $2N \geq l_{max} - l_{min} + 1$, where the image is of size $2^N \times 2^N$. In fact, our algorithm can be applied to handle the 3-D case, where the 3-D image is represented by the 3-D bincodes.[3]

The significance of neighbor finding is due to its popular use in the area of image processing and pattern analysis. Our main contribution is to present a faster neighbor algorithm. The result of this paper can also be applied to many, applications on bincodes, such as connected component labeling, perimeter, Euler number, and so on, to achieve better performance.

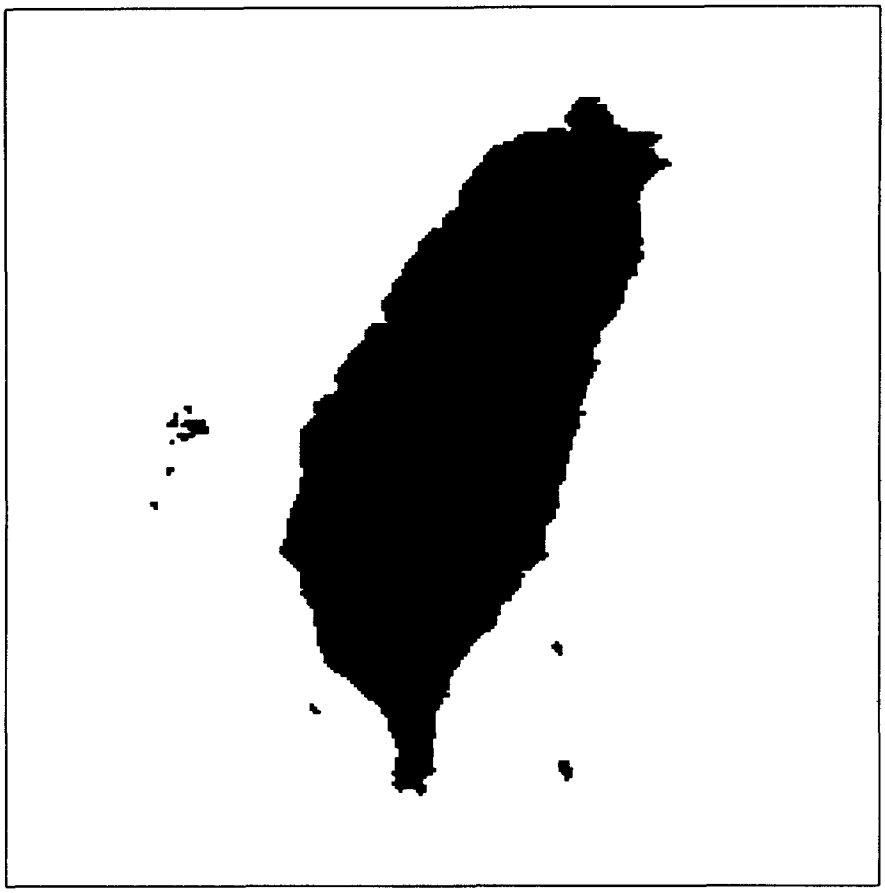

Fig. 9. Taiwan map.

Table 2. Experimental results for real maps

Image	The size of given bincodes	(a) The size of possible candidates	(b) The execution time for four-neighbor finding (s)	(b)/(a)
Floodplain	807	2454	0.31	0.000126
Taiwan	459	1453	0.20	0.000138

REFERENCES

1. C.-Y. Huang and K.-L. Chung, Fast operations on binary images using interpolation-based bintrees, *Pattern Recognition* **28**(3), 409–420 (1995).
2. H. Samet, *Applications of Spatial Data Structures*. Addison Wesley, New York (1990).
3. M. A. Ouksel and A. Yaagoub, The interpolation-based bintree and encoding of binary images, *CVGIP: Graph. Models Image Process.* **54**(1), 75–81 (1992).
4. C. A. Shaffer, R. Juvvadi and L. S. Health, Generalized comparison of quadtree and bintree storage requirements, *Image Vis. Comput.* **11**(7), 402–412 (1993).
5. K. Knowlton, Progressive transmission of gray-scale and binary pictures by simple, efficient, and lossless encoding schemes, *Proc. IEEE* **68**, 885–896 (1980).
6. H. Samet and M. Tamminen, Computing geometric properties of images represented by linear quadtrees, *IEEE Trans. Pattern Anal. Mach. Intell.* **7**(2), 229–240 (1985).
7. G. Schrack, Finding neighbors of equal size in linear quadtrees and octrees in constant time, *CVGIP: Image Understanding* **55**(3), 221–230 (1992).
8. R. Sedgewick, *Algorithms*. Addison Wesley, New York (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. Since 1995, he has been a Full 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 Republic of China and the Outstanding Research Awards from the National Science Council of Republic of China in 1992 and 1994, respectively. In 1995, he received the National Taiwan Institute of Technology Teaching Excellence Award. Dr Chung is a member of the IEEE Computer Society and the SIAM Society.