A Quadtree Medial Axis Transform

HANAN SAMET *University of Maryland*

ABSTRACT: ***Quadtree skeletons are exact representations of the image and are used because they are observed to yield space efficiently and a decreased sensitivity to shifts in contrast with the quadtree. The* QMAT *can be used as the underlying representation when solving most problems that can be solved by using a quadtree. An algorithm is presented for the computation of the* QMAT *of a given quadtree by only examining each* BLACK *node's adjacent and abutting neighbors.***

Hanan Samet's research interests are data structures, image processing, programming languages, artificial intelligence, and database management systems.
Arpa Network Address: hjs @ cvl

The support of the Defense Advanced Research Projects Agency and the U.S. Army Night Vision Laboratory under Contract DAAG-53-76C-0138 (DARPA Order 3206) is gratefully acknowledged.

Author's Present Address: Hanan Samet, Computer Science Department, University of Maryland, College Park, Maryland 20742

1. INTRODUCTION

There are a number of methods of representing images [15] among which are borders, arrays, and skeletons. We are interested in the quadtree [9]. It is interchangeable with borders, arrays, and runs [4, 17, 20–22]. Methods for the computation of geometrical properties such as connectivity, perimeter, genus, and distance exist for quadtrees [3, 18, 19, 23, 26]. In this paper, we demonstrate the usefulness of the chessboard distance transform in computing the skeleton [23] and medial axis transform [1, 2, 14, 15] of an image represented by a quadtree.

The quadtree is a member of a class of region representations that involves various types of "maximal blocks" that are contained in a given region. For example, we can represent a region R as a linked list of the runs (of pixels) in which R meets the successive rows of the array [16]. Here each "block" is a 1-by-m rectangle, where m is the run length; the runs are the largest such blocks that R contains, and R is determined by specifying the initial points (or centers) and lengths of the runs. The two-dimensional analog of the run length representation is called the *medial axis transformation* [1, 14]. In this representation, R is represented by a set of maximal blocks (square or any other shape) that it contains; R is determined by specifying the centers and radii of those blocks.

The quadtree approach to image representation is an attempt to exploit the maximal block concept in a more systematic manner. Given a $2^n \times 2^n$ array of unit pixels, we repeatedly subdivide the array into quadrants, subquadrants, etc., until we obtain blocks (possibly single pixels) which consist entirely of single values (e.g., gray level). This process is represented by a tree of out-degree 4 in which the root node represents the entire array, the four sons of the root node represent the quadrants, and the terminal nodes correspond to those blocks of the array for which no further subdivision is necessary. The nodes at level K (if any) represent blocks of size $2^K \times 2^K$ and are often referred to as nodes of size 2^K. Thus a node at level 0 corresponds to a single pixel in the image, while a node at level n is the root of the quadtree. For example, Figure 1b is a block decomposition of the region in Figure 1a while Figure 1c is the corresponding quadtree. In general, we will be dealing with two values, 1 and 0, where BLACK and WHITE square nodes in the tree represent blocks consisting entirely of 1's and 0's respectively. Circular nodes, also termed GRAY nodes, denote nonterminal nodes.

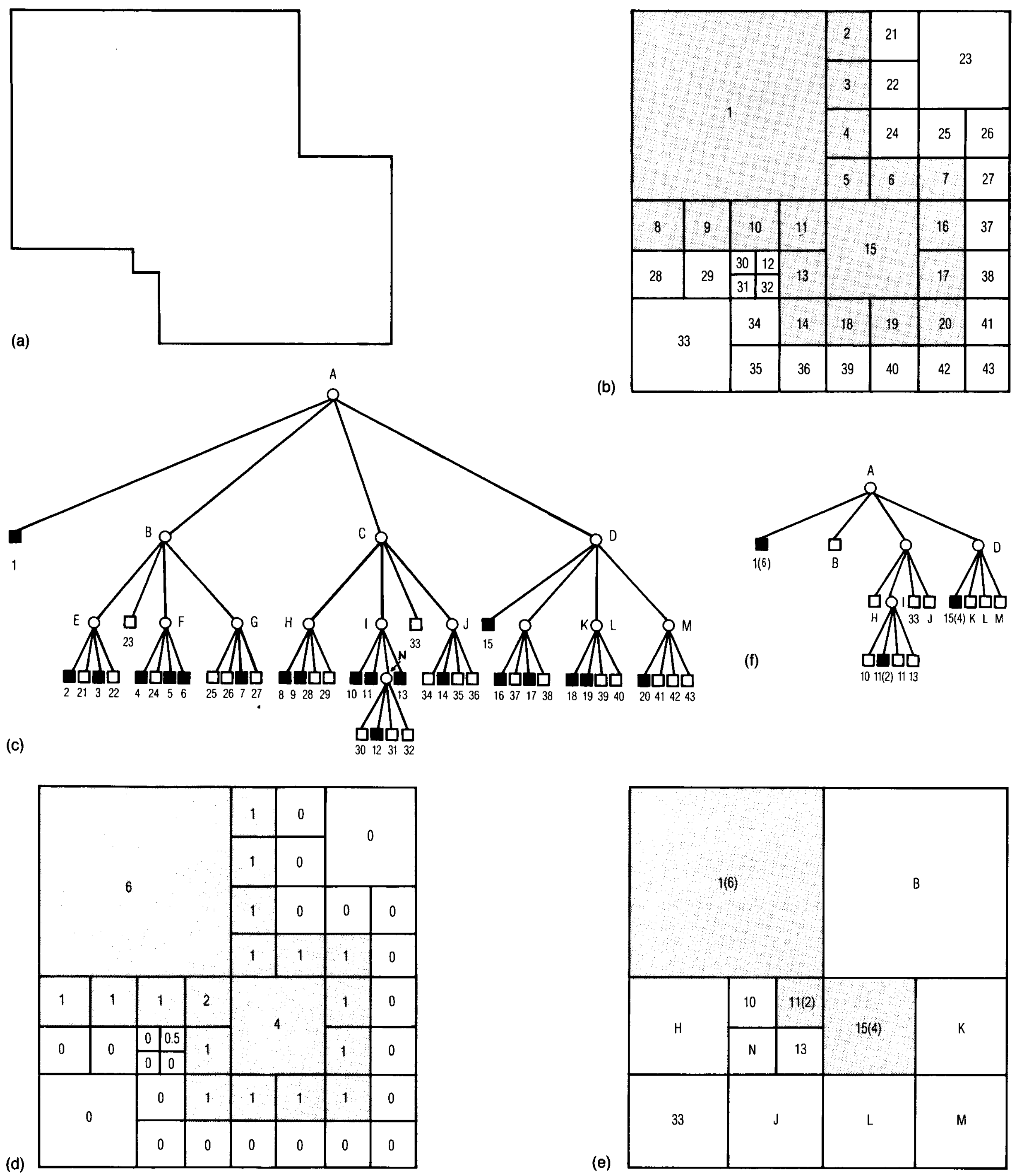


FIGURE 1. An image, its maximal blocks, the corresponding quadtree, the chessboard distance transform, the block decomposition of the QMAT, and the QMAT. Blocks in the image and in the QMAT are shaded. (*a*) Sample image. (*b*) Block decomposition of the image in (a). (c) Quadtree representation of the blocks in (b). (*d*) Chessboard distance transform of (b). (e) Block decomposition of the QMAT of (b). Radius values are within parentheses. (*f*) QMAT representation of the blocks in (b). Radius values are within parentheses.

The quadtree finds a number of uses in computer graphics and image processing. In particular, Warnock's algorithm [13, 28, 29] for hidden surface elimination is based on such a principle—i.e., it successively subdivides the picture into smaller and smaller squares in the process of searching for areas to be displayed. It is relatively compact [10] and well-suited to operations such as union and intersection [5, 6, 27]. It has been used to develop a variety of algorithms for manipulation of objects in two dimensions [7] as well as three dimensions [8, 11, 30].

The main result of this paper is the definition of a new data structure termed the QMAT which is a quadtree skeleton with a set of distance values. It yields a partition of the image into a set of nondisjoint squares having sides whose lengths are sums of powers of 2. This is in contrast with the quadtree which is a partition of an image into a set of disjoint squares having sides of lengths which are powers of 2. Our motivation is not to study skeletons for the usual purpose of obtaining an approximation to an image. Thus, our decisions with respect to the type of distance metric [2, 15] that we employ (i.e., a chessboard distance metric rather than a Euclidean distance metric) are not influenced by customary considerations such as sensitivity to relative orientation, etc. Instead, the quadtree skeleton is shown to yield an exact representation of the image. It is used because it is observed to yield space efficiency and a decreased sensitivity to shifts in contrast with the quadtree. We also observe that the QMAT can be used to solve most problems which can be solved using the quadtree representation. Furthermore, the QMAT may be useful in performing operations such as propagation, shrinking, and matching [15].

Section 2 contains a discussion of skeletons and medial axis transforms and their adaptation to the quadtree. In Section 3, we discuss a number of algorithms for the computation of the QMAT and prove their equivalence. Sections 4 and 5 contain a formal description of the best of these algorithms and an analysis of its execution time. The chosen algorithm is motivated by a close inspection of the geometrical properties of the quadtree method of representation. We use a variant of ALGOL 60 [12] in the formal presentation of the algorithm. Reconstruction of quadtree from its QMAT is straightforward and is not treated here.

2. MEDIAL AXIS TRANSFORMS, QUADTREES, AND SKELETONS

We are given an image where the set of points in a certain region is labeled S and the sets of points outside of the region is labeled $\bar{S}$ (analogous to BLACK and WHITE respectively). We say that for a point x and a set V, the distance (according to a suitably defined distance metric, *d*) from x to the nearest point of V is $d(x, V) = \min\{d(x, y) \mid y \in V\}$. Two points, x and y, are said to be neighbors if $d(x, y) = 1$. We are interested in a subset of S, *T*, such that all elements of *T* have a distance from $\bar{S}$ which is a local maximum. In other words, for each point in *T*, no neighboring point in S but not in *T* has a greater distance from $\bar{S}$. The set of points comprising *T* is said to constitute a skeletal description of S. As an example, consider the rectangle in Figure 2 whose skeleton consists of line segments labeled *a*, *b*, *c*, *d*, and *e*. If we know the points of the skeleton and their associated distance values, then we can reconstruct S exactly. The set of points comprising the skeleton and their associated values is termed the medial axis transform (MAT). The MAT of S provides a concise method of defining and representing S. For example, using a Euclidean distance metric [2, 15], the MAT corresponding to a circle is a

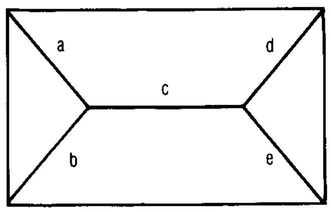


FIGURE 2. A rectangle and its skeleton using d_E.

point at its center having a distance value equal to the circle's radius.

Clearly, the definition of the distance metric plays an important role in determining the form of the MAT. The most commonly known distance metric is the Euclidean distance:

$$d_E(p, q) = \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2}$$

whose maximal blocks are discs. Two other metrics which are more commonly known in digital picture processing are the Absolute Value distance (also known as the city block distance):

$$d_A(p, q) = |p_x - q_x| + |p_y - q_y|$$

whose maximal blocks are diamonds, and the Maximum Value distance (also known as the chessboard distance):

$$d_M(p, q) = \max \{|p_x - q_x|, |p_y - q_y|\}$$

whose maximal blocks are squares. Note that in any case, the MAT determines the entire image although it is true that a point in the image may lie in more than one maximal block. Figures 3c and 3d show the MATs of the rectangle of Figure 3a using d_A and d_M respectively. Figure 3b shows d_A and d_M for the rectangle, which in this example, are identical. Note that the MATs in Figures 2c and 2d have distance values that are one less than the values in Figure 2b. This is due to the MAT being defined as extending to the last pixel within the image rather than to the first pixel outside the image [15].

Maximal blocks can be of any size and at any position. Thus, they are somewhat unwieldy as primitive elements for representation purposes since the process of determining them may be complex. The quadtree approach to image representation is an attempt to exploit the maximal block concept in a more systematic manner by constraining the sizes of the blocks and the positions of their centers.

In [23, 26], the concept of distance is applied to a quadtree. In particular, it was shown in [23] that the chessboard distance metric is especially suitable for a quadtree since it has the property that given a point p, the set of points {q} such that $d(p, q) \leq t$ is a square. The chessboard distance transform for a quadtree, *DIST*, was defined as a function that yields for each BLACK block in the quadtree *the chessboard distance from the center of the block to the nearest point which is on a BLACK–WHITE border*. More formally, letting x be the center of a BLACK block *b*, z be a point on the border of the WHITE block *W* (say *B*(*W*)), we have:

$$F(b, W) = \min_{z \in B(W)} d(x, z)$$

$$\mathrm{DIST}(b) = \min_W F(b, W)$$

We also say that DIST of a WHITE block is zero and that the border is BLACK for the purpose of the computation of F and DIST.

We are now ready to define the Quadtree Medial Axis Transform (QMAT). We first define the quadtree skeleton. Let the set of BLACK blocks in the image be denoted by *B*. For each BLACK block, b_i, let $S(b_i)$ be the part of the image

1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1

(a)

1	1	1	1	1	1	1	1	1	1
1	2	2	2	2	2	2	2	2	1
1	2	3	3	3	3	3	3	2	1
1	2	3	4	4	4	4	3	2	1
1	2	3	3	3	3	3	3	2	1
1	2	2	2	2	2	2	2	2	1
1	1	1	1	1	1	1	1	1	1

(b)

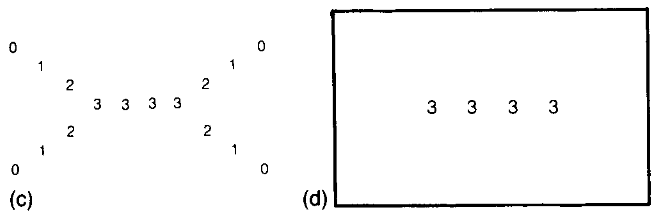


FIGURE 3. A binary array representation of an image, its distance transform, and its MAT. (*a*) Image. (*b*) d_A and d_M for the image in (a). (*c*) MAT of the image in (a) using d_A. (*d*) MAT of the image in (a) using d_M.

spanned by a square with side width 2*DIST(b_i) centered about b_i. The *quadtree skeleton* consists of the set, T, of BLACK blocks satisfying the following properties:

(1) area(B) = UNION(S(t_i))
(2) for any $t_j \in T \nexists b_k \in B(b_k \neq t_j) \ni S(t_j) \subseteq S(b_k)$
(3) $\forall b_i \in B \ \exists t_j \in T \ni S(b_i) \subseteq S(t_j)$

Property (1) insures that the entire image is spanned by the skeleton. Property (2) is termed the subsumption property and we say that b_j is *subsumed* by b_k when $S(b_j) \subseteq S(b_k)$. Property (2) means that the elements of the quadtree skeleton are the blocks with the largest distance transform values. This is a stronger requirement than the stipulation that given $t_j \in T \nexists t_k \in T(k \neq j) \ni S(t_j) \subseteq S(t_k)$ as will be seen below. Such a stipulation only insures that there are no extraneous elements with respect to subsumption of the image in a given quadtree skeleton. Property (3) insures that no block in B and not in T requires more than one element of T for its subsumption—e.g., one half of the block is subsumed by one element of T and the other half is subsumed by another element of T.

THEOREM 1: The quadtree skeleton of an image is unique.

PROOF: Assume that the quadtree skeleton is not unique. Let T_1 and T_2 both be quadtree skeletons of the same image, i.e., B = UNION(S(t_{1i})) $t_{1i} \in T_1$ and B = UNION(S(t_{2j})) $t_{2j} \in T_2$. Assume, without loss of generality, that $\exists t_{1k} \in T_1 \ni t_{1k} \notin T_2$. Therefore, by Property (3) $\exists t_{2'} \in T_2(t_{2'} \neq t_{1k}) \ni S(t_{1k}) \subseteq S(t_{2'})$. However, this contradicts Property (2) which stipulates that for any $t_{1i} \in T_1 \nexists b_j \in B(b_j \neq t_{1i}) \ni S(t_{1i}) \subseteq S(b_j)$. Hence, the quadtree skeleton of an image is unique. *Q.E.D.*

The QMAT of an image is *the quadtree whose BLACK nodes correspond to the BLACK blocks comprising the quadtree skeleton and their associated chessboard distance transform values.* All remaining nodes in the QMAT are WHITE and GRAY with distance value zero. For example, Figure 1d contains the chessboard distance transform corresponding to the region given in Figure 1a. Figures 1e and 1f contain the block and tree representations respectively of the QMAT of Figure 1a.

We now make the following observations with the aid of Figure 1. The squares spanned by the chessboard distance transform of the blocks of the QMAT have sides whose lengths are sums of powers of 2 and are not necessarily disjoint. This is in contrast with the quadtree which is a partition of an image into a set of disjoint squares having sides whose lengths are powers of 2. For example, Block 11 is subsumed by both Blocks 1 and 15.

Recall that the subsumption property, i.e., Property (2), means that the elements of the QMAT are the blocks with the largest distance transform values. For example, for Figure 1, the quadtree skeleton consists of Blocks 1, 11, and 15 rather than Blocks 1, 12, and 15 since Block 12 is subsumed by Block 11. The latter result would have been legal had we defined the quadtree skeleton as the set, T, of BLACK blocks such that area (B) = UNION(S(t_i)) $t_i \in T$ and for any $t_j \in T \nexists t_k \in T(k \neq j) \ni S(t_j) \subseteq S(t_k)$. Such a definition has several undesirable implications. First, it means that the quadtree skeleton of an image is no longer unique, e.g., for Figure 1 both {1, 11, 15} and {1, 12, 15} would be legal quadtree skeletons. Second, it leads to a QMAT which contains more nodes, e.g., WHITE node N in Figure 1f would be replaced by a GRAY node having BLACK son 12 and WHITE sons 30, 31, and 32.

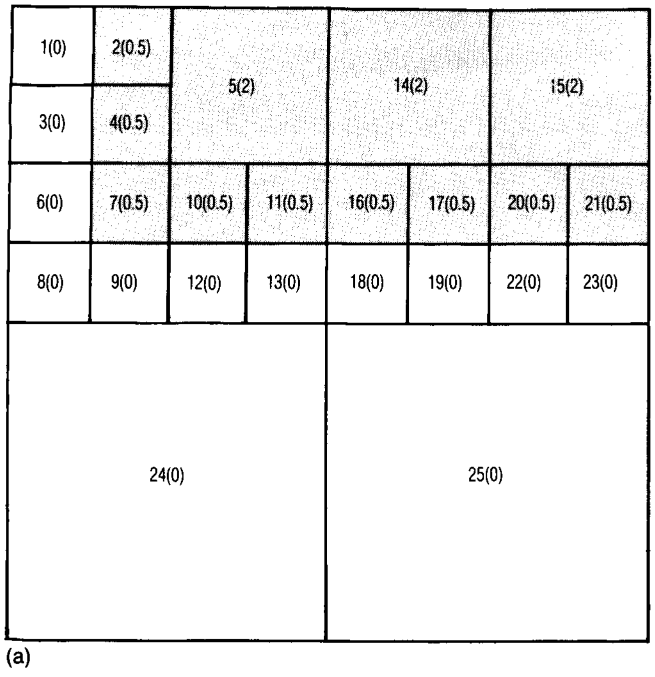
The fact that the border of the image is assumed to be BLACK results in minimizing the number of nodes in the QMAT. Without this assumption, Block 1 would be of radius 4 and would not lead to the subsumption of Blocks 2, 3, 4, 8, 9, and 10. Note that Blocks 5 and 11 are subsumed by Block 15 anyway so that their subsumption is not dependent on our assumption.

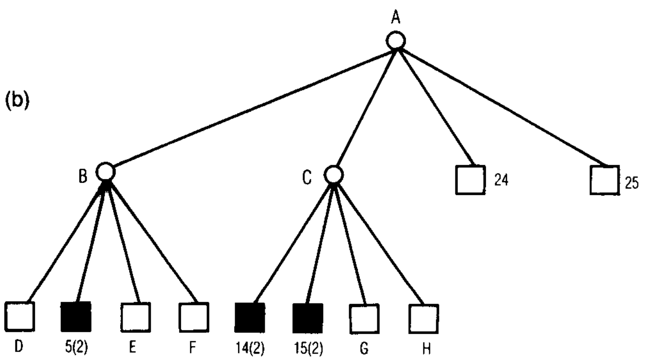
Before proceeding any further it is appropriate to make a few additional comments about Property (3) of the quadtree skeleton definition. This property does not yield a minimal set of blocks. For example, in the image of Figure 4a, Property (3) requires that the quadtree skeleton contain Blocks 5, 14, and 15 while in actuality Blocks 5 and 15 are sufficient since together they subsume Block 14. Thus, if we were interested in a minimal set of blocks we would modify Property (3) as in (3′) below:

$$(3') \ \nexists t_j \in T \ni S(t_j) \subseteq \text{UNION}(S(t_k)) \ t_k \neq t_j$$

The reason we do not use the definition of a quadtree skeleton which yields the minimal set of blocks is twofold. First, by virtue of the definition of the QMAT, the tree size of the QMAT would be unaffected by using Property (3′) instead of Property (3) since the only difference is that the additional blocks are represented by BLACK nodes rather than WHITE nodes, e.g., node 14 in Figures 4b and 4c. That this is always true can be seen easily by observing that for a node to be extraneous by virtue of Property (3′), it must be subsumed by its neighbors which must themselves be BLACK. Thus the extraneous node when represented by a WHITE node cannot be merged with its neighbors to yield a larger node and must remain a part of the QMAT. Second, as will be seen in Section 3, the QMAT creation algorithm is considerably simpler when Property (3) is used.

The QMAT representation can be used as an alternative data structure for the representation of an image. In particular, it has the property that for any image it requires, at most, as many nodes as the quadtree. This is obvious when we recall that each node in the QMAT corresponds to one or more nodes of the quadtree and that each member of the quadtree skeleton is a node in the quadtree. Of course, the QMAT does require that the DIST value be stored with each node. As an example of the savings in storage, consider the image in Figure 1a. The QMAT, shown in Figure 1f, requires 17 nodes while the quadtree, shown in Figure 1c, requires 57 nodes.





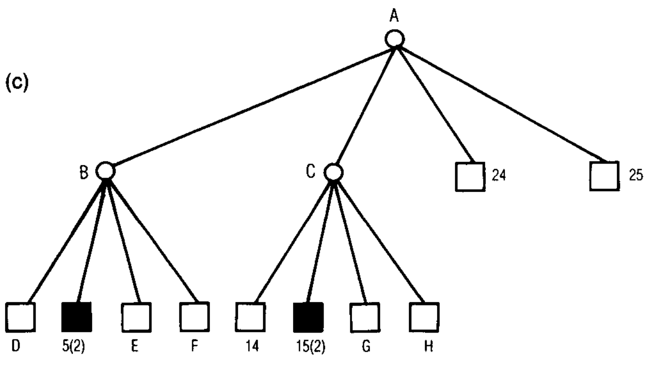


FIGURE 4. An image and its corresponding QMATs using Properties (3) and (3′) for the quadtree skeleton definitions. Blocks in the QMAT are shaded. (*a*) Image. The value of the chessboard distance transform is within parentheses. (*b*) QMAT of the image in (a) using Property (3). Radius values are within parentheses. (c) QMAT of the image in (a) using Property (3′). Radius values are within parentheses.

An interesting property of the QMAT is that there is a class of images for which it requires a minimum number of nodes regardless of the image resolution. Clearly, if the image is all WHITE or all BLACK, then both the quadtree and the QMAT require a single node. However, when a $2^n \times 2^n$ image consists of a BLACK square of side length $2^n - 1$, then the advantage of the QMAT over the quadtree in terms of space utilization is at a maximum. For example, consider the image in Figure 5a and its quadtree and QMAT in Figures 5b and 5c, respectively. The quadtree requires 45 nodes while the QMAT requires only 5 nodes.

In fact, for any image of such a shape the QMAT requires only 5 nodes while the quadtree requires a number of nodes which depends on the maximum level of the tree. The exact number of nodes required for such a quadtree of level *n* can be obtained by use of the following recurrence relations. Assume, without loss of generality, an image similar to that of Figure 5a, i.e., the largest block is in the *NW* quadrant. Let t_i denote the number of nodes in a quadtree of level *i* and r_{i-1} be the contribution made by the *NE* and *SW* quadrants of a quadtree of level *i*.

$$t_i = \begin{cases} 1 & i = 0 \\ 1 + 1 + 2 \cdot r_{i-1} + t_{i-1} & i \geq 1 \end{cases}$$

$$r_i = \begin{cases} 1 & i = 0 \\ 1 + 2 + 2 \cdot r_{i-1} & i \geq 1 \end{cases}$$

It can easily be shown that these relations have the following solutions:

$$r_i = 2^{i+2} - 3$$

$$t_i = 2^{i+3} - 4 \cdot i - 7$$

To see this, we observe that

$$r_i = 3 \cdot \textstyle\sum_{j=0}^{i-1} 2^j - 2^{i+1}$$
$$= 3 \cdot (2^i - 1) + 2^i = 2^{i+2} - 3$$

and substituting into t_i we have

$$t_i = 2 + 2 \cdot (2^{i+1} - 3) + t_{i-1} = 4 \cdot (2^i - 1) + t_{i-1}$$

Solving for t_i we get

$$t_i = 4 \cdot \textstyle\sum_{j=1}^{i} (2^j - 1) + 1 = 4 \cdot (2^{i+1} - 2 - i) + 1$$
$$= 2^{i+3} - 4 \cdot i - 7$$

Thus, for a quadtree of level n, the number of nodes that can be saved by using the QMAT representation is $2^{n+3} - 4 \cdot n - 12$. For example, for $n = 3$, the difference is 40 nodes, i.e., a reduction by a factor of 15.

The QMAT representation also has the property that the number of nodes necessary to represent an image is not as shift-sensitive as is the quadtree. This is a direct result of the fact that the QMAT always requires a number of nodes less than or equal to the quadtree. It is also quite apparent when we realize that the QMAT is most economical storage-wise *vis-à-vis* the quadtree when large blocks are surrounded by smaller blocks which is normally the situation when a shift operation takes place. For example, when the image of Figure 6a is shifted by one unit to the right yielding Figure 6d, its quadtree gets considerably larger. In particular, Figure 6b contains 17 nodes while Figure 6e, the quadtree corresponding to the shifted image, contains 49 nodes. However, the QMAT is not as sensitive to shifts since it always requires a number of nodes less than or equal to those contained in the quadtree. In

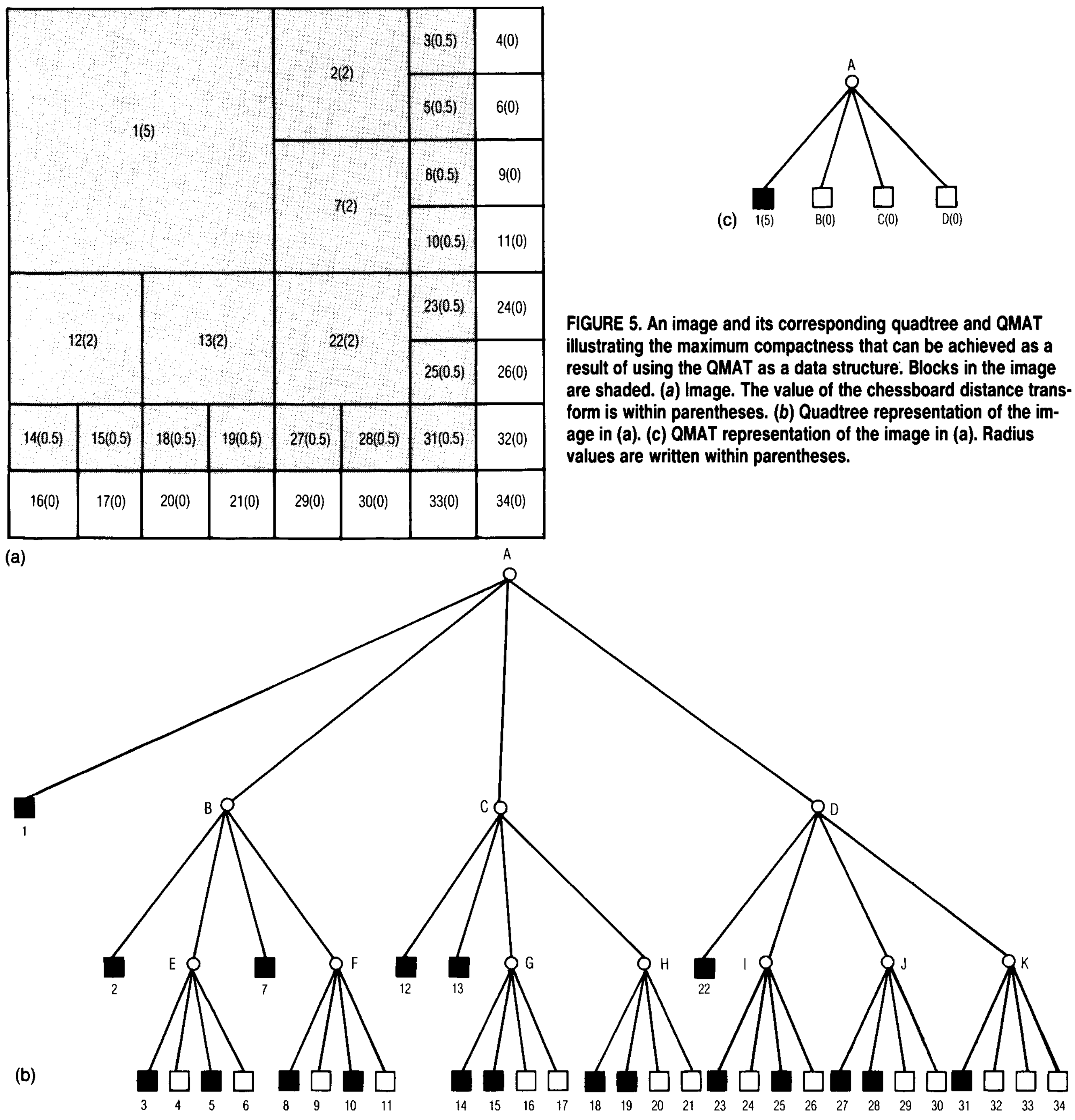


FIGURE 5. An image and its corresponding quadtree and QMAT illustrating the maximum compactness that can be achieved as a result of using the QMAT as a data structure. Blocks in the image are shaded. (*a*) Image. The value of the chessboard distance transform is within parentheses. (*b*) Quadtree representation of the image in (a). (c) QMAT representation of the image in (a). Radius values are written within parentheses.

Figure 6, the QMAT of Figure 6a, given in Figure 6c, is identical to the quadtree. However, the QMAT of the shifted image, given in Figure 6f, is considerably smaller than its corresponding quadtree as well as the QMAT of the image prior to the shift, i.e., 9 nodes versus 17 nodes. As another example, consider the image of Figure 7a which has a minimal nontrivial QMAT in terms of the number of nodes, i.e., 5 nodes. Figure 7d is the result of shifting the image of Figure 7a one unit to the right. Note that the new QMAT given in Figure 7f requires more nodes than the one corresponding to the unshifted image given in Figure 7c, i.e., 21 nodes versus 5 nodes. However, this number is less than the number of nodes in the shifted quadtree as shown in Figure 7e, i.e., 21 nodes for the QMAT versus 41 nodes for the shifted quadtree. Thus, we see that the compactness of the QMAT is also preserved when the image is subjected to shifts.

3. ALGORITHMS FOR THE COMPUTATION OF THE QMAT

Properties (2) and (3) of the quadtree skeleton definition of Section 2 suggest the following simple two-step algorithm (termed "A") for determining the QMAT. At the end of the

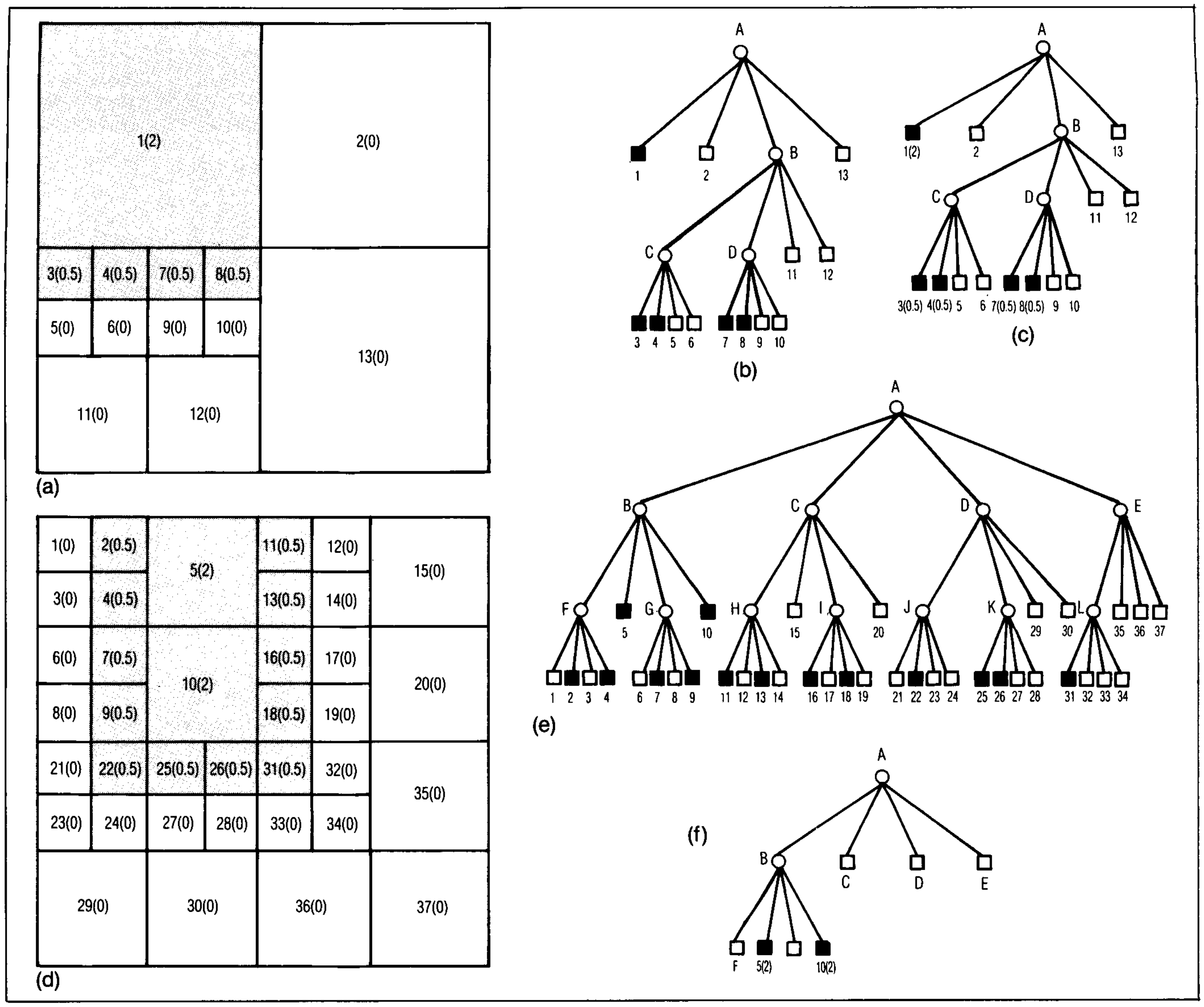


FIGURE 6. An image and its corresponding quadtree and QMAT, and the result of shifting it by one unit to the right. Blocks in the image are shaded. (*a*) Image. The value of the chessboard distance transform is within parentheses. (*b*) Quadtree representation of the image in (a). (c) QMAT representation of the image in (a). Radius values are within parentheses. (*d*) The image in (a) shifted by one unit to the right. The value of the chessboard distance transform is within parentheses. (e) Quadtree representation of the image in (d). (*f*) QMAT representation of the image in (d). Radius values are within parentheses.

algorithm, the set T contains the BLACK blocks comprising the QMAT.

Algorithm A

1. Sort the BLACK blocks in increasing order by value of their chessboard distance transform forming the set T, i.e., $\text{DIST}(t_i) \le \text{DIST}(t_{i+1})\ t_i \in T$
2. Starting with $i = 1$: For each $t_i \in T \ni \exists t_j (i < j)$ and $S(t_i \subseteq S(t_j)$, remove t_i from T.

From a computation complexity standpoint, Algorithm A is quite costly since it involves sorting the BLACK blocks as well as examining whether or not a block is subsumed by the remaining blocks. Instead, we use the following algorithm (termed "B") which traverses the quadtree in postorder, i.e., the sons of a node are visited first, and determines for each node corresponding to a BLACK block, P, whether $S(P) \subseteq S(Q)$ where Q is one of P's eight neighbors in the N, NE, E, SE, S, SW, W, and NW directions.

Algorithm B

1. Sort the BLACK nodes in postorder, forming the set T.
2. Starting with $i = 1$: For each $t_i \in T$ determine if $\exists t_j (i \neq j) \in T \ni t_j$ is a neighbor of t_i in one of the directions $\{N, NE, E, SE, S, SW, W, NW\}$ and $S(t_i) \subseteq S(t_j)$, then remove t_i from T.

The remainder of the discussion elaborates on properties of Algorithm B. In general, whenever a BLACK block is subsumed by one of its neighbors, then it appears in the QMAT as a WHITE block. Once all the sons of a GRAY node have been processed, and if in the QMAT they all correspond to WHITE blocks, then the GRAY node is changed to correspond to a WHITE block (e.g., GRAY node N of Figure 1c having sons 30, 12, 31, and 32 is changed to correspond to a WHITE block in Figure 1f).

At this point, it is appropriate to examine the notion of subsumption in a more rigorous manner. Given adjacent nodes P and Q corresponding to BLACK blocks appearing at levels L_P and L_Q respectively in the quadtree such that $L_Q > L_P$, and letting $D(P, Q) = \mathrm{DIST}(Q) - 2 \uparrow (L_Q - 1) - 2 \uparrow (L_P - 1)$, then P is said to be subsumed by Q if $D(P, Q) = \mathrm{DIST}(P)$. It should be clear that for P to be subsumed by Q, $D(P, Q)$ cannot be greater than DIST(P) since this would contradict the definition of the chessboard distance transform, *i.e.*, P would have a closer BLACK-WHITE border point than does Q although P is constrained by the value of $D(P, Q)$ to be entirely contained in the square of side width 2*DIST(Q) centered at Q. Clearly, when $D(P, Q) < \mathrm{DIST}(P)$, P is not subsumed by Q.

When $D(P, Q) = \mathrm{DIST}(P)$, there are two cases to consider. If $\mathrm{DIST}(P) = 2 \uparrow (L_P - 1)$, then P is adjacent to the outer border of S(Q) and thus no BLACK blocks can be subsumed by P (e.g., BLACK Block 9 in Figure 1b is adjacent to the outer border of the square spanned by Block 1). Thus, changing Block P from BLACK to WHITE will not affect the detection of subsumption of other nodes.

However, if $\mathrm{DIST}(P) > 2 \uparrow (L_P - 1)$, *i.e.*, the second case to be considered when $D(P, Q) = \mathrm{DIST}(P)$, then P is not adjacent

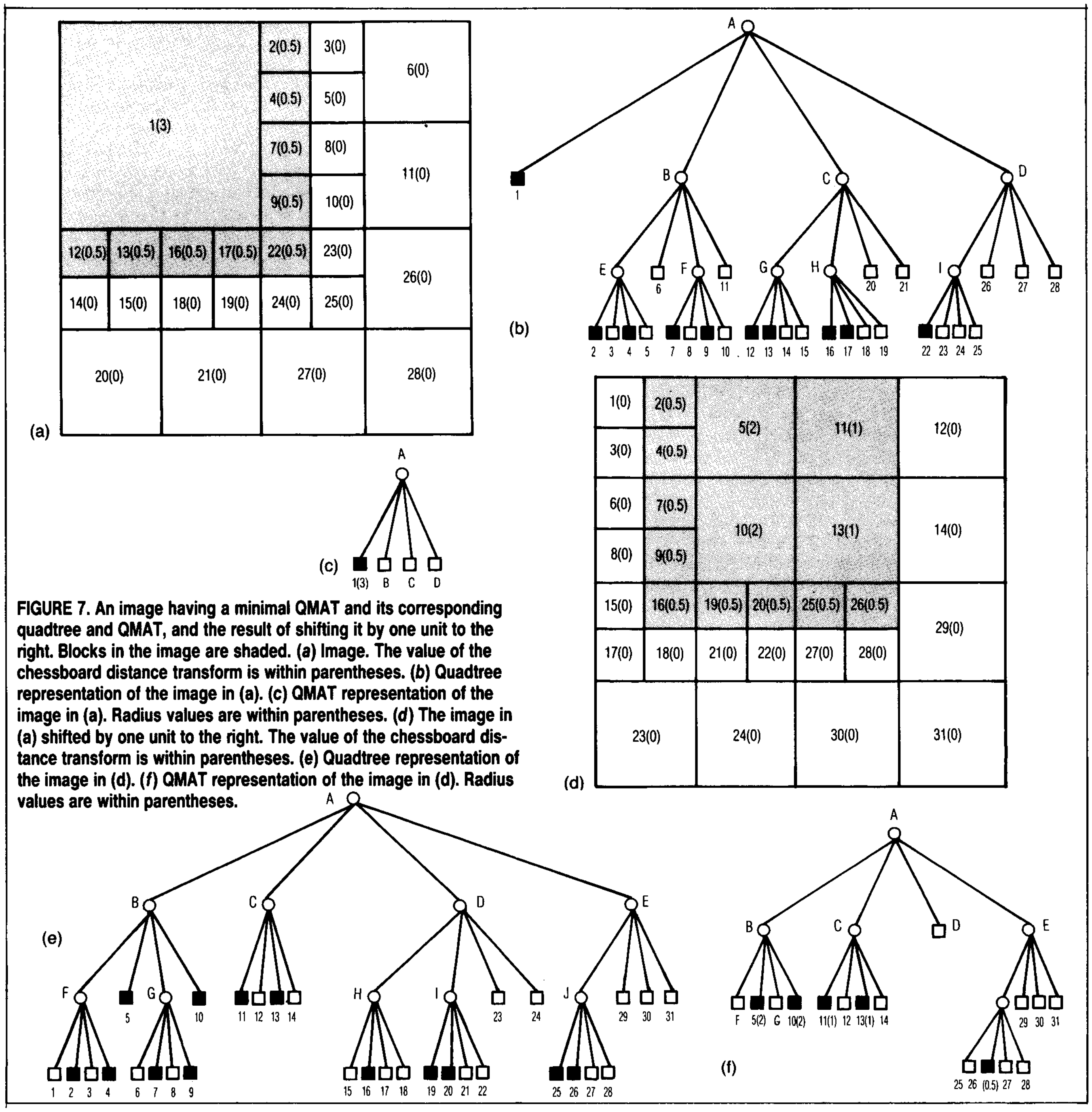


FIGURE 7. An image having a minimal QMAT and its corresponding quadtree and QMAT, and the result of shifting it by one unit to the right. Blocks in the image are shaded. (*a*) Image. The value of the chessboard distance transform is within parentheses. (*b*) Quadtree representation of the image in (a). (c) QMAT representation of the image in (a). Radius values are within parentheses. (*d*) The image in (a) shifted by one unit to the right. The value of the chessboard distance transform is within parentheses. (e) Quadtree representation of the image in (d). (*f*) QMAT representation of the image in (d). Radius values are within parentheses.

to the outer border of S(Q). This means that some blocks which are subsumed by Q can only be detected by virtue of being subsumed by P since they are not adjacent to Q. Denote these blocks by U(P, Q). It can be shown that all elements of U(P, Q) satisfy the following properties:

1. Q is larger than P.
2. Each element of U(P, Q) is smaller than P.
3. If Q is adjacent to P along side R of the BLACK block corresponding to P, then U(P, Q) is equal to the blocks subsumed by the opposite side, denoted by OPSIDE(R), of P's block.
4. If Q abuts the corner formed by sides R and T of the BLACK block corresponding to P, then U(P, Q) is equal to the blocks subsumed by sides OPSIDE(R) and OPSIDE(T) of P's block as well as the blocks subsumed by the corner formed by these sides of P's block.

Properties (1)–(4) imply that all elements of U(P, Q) are in the space spanned by FATHER(P), i.e., they are in the region spanned by the brothers of P. This means that an algorithm that processes a GRAY son prior to its BLACK or WHITE brothers insures that the QMAT is formed by examining blocks for subsumption according to increasing size, i.e., smaller size first. As soon as a BLACK block is determined to be subsumed by its neighbor, its DIST and NODETYPE fields are changed to zero and WHITE respectively. This leads to the following result.

LEMMA 1: *Both Algorithms A and B result in a computation satisfying the definition of a quadtree skeleton, given in Section 2 and the QMAT of an image.*

PROOF: Algorithm A clearly satisfies Properties (1)–(3) of the definition since its steps are equivalent to the definition. To show that Algorithm B meets our requirements is slightly more complex. Properties (1) and (3) are satisfied since Algorithm B starts with the QMAT and the quadtree being identical and then systematically removes nodes whose corresponding blocks are subsumed by others. Satisfaction of Property (2) is shown as follows. Algorithm B is based on the principle that each block is subsumed by a neighboring block. It examines each adjacency and removes a BLACK block from the skeleton if it is subsumed by an adjacent block in the skeleton, i.e., one that has not yet been removed by virtue of being subsumed by yet another larger adjacent block. Properties (1)–(4) of the case when $D(P, Q) = \mathrm{DIST}(P)$ and $\mathrm{DIST}(P) > 2 \uparrow (L_P - 1)$ plus the fact that a GRAY son is processed before its BLACK and WHITE brothers insure that no block is removed from the skeleton before blocks that are subsumed by it. Thus, we see that no block in the QMAT is subsumed by another block in the quadtree. Recall from Section 2 that this is a stronger statement than not being subsumed by another node in the QMAT. Q.E.D.

THEOREM 2.: Algorithms A and B are equivalent.

PROOF: By Lemma 1, both Algorithms A and B compute the quadtree skeleton. Theorem 1 indicates that the quadtree skeleton of an image is unique and our result follows. *Q.E.D.*

The equivalence of Algorithms A and B can also be seen by observing that they both start with the smallest BLACK blocks and attempt to determine if they are subsumed by other BLACK blocks. The key to the superiority of Algorithm B is that no formal sorting is required and also that blocks that cannot possibly subsume one another are not checked for subsumption, i.e., Algorithm B only examines a maximum of eight neighboring blocks while Algorithm A examines all possible larger sized BLACK blocks. Also note the simplicity of Algorithm B that results from using Property (3) rather than (3') in the definition of a quadtree skeleton, since each block in the original image can only be subsumed in its entirety. Thus, there is no need to examine whether a node is subsumed by a set of other nodes, e.g., node 14 of Figure 1a is subsumed by nodes 5 and 15.

4. FORMAL IMPLEMENTATION OF THE ALGORITHM

Prior to describing our algorithm, it is useful to define our representation as well as some elementary operations. Let each node in a quadtree be stored as a record containing seven fields. The first five fields contain pointers to the node's father and its four sons, labeled *NW*, *NE*, *SE*, and *SW*. Given a node *P* and a son *I*, these fields are referenced as FATHER(*P*) AND SON(*P*, *I*) respectively. We can determine the specific quadrant in which a node, *P*, lies relative to its father by use of the function SONTYPE(*P*) which has a value of *I* if SON(FATHER(*P*), *I*) = *P*. The sixth field, NODETYPE, describes the contents of the block of the image which the node represents, i.e., BLACK, WHITE, or GRAY. The seventh field, DIST, indicates the value of the chessboard distance transform for the node. This field is only meaningful for BLACK nodes. WHITE and GRAY nodes are said to have a DIST value of zero. Note that this is different from the concept of node distance, i.e., for a node at level *i*, *n–i* FATHER links must be ascended to reach the root of the tree.

The four sides of a node's block are called its *N*, *E*, *S*, and *W* sides. They are also termed its boundaries and at times we speak of them as if they are directions. Figure 8 shows the relationship between the quadrants of a node's block and its boundaries. Given blocks *P* and *Q*, we say that *Q* is a *neighbor* of *P* when both of the following conditions are satisfied:

1. *P* and *Q* share a common border, even if only a corner.
2. If *Q* is a BLACK or WHITE block, then its size is greater than or equal to that of *P*, while if it is a GRAY block, then it and *P* are of equal size.

For example, in Figure 1, BLACK node 15 is the neighbor of BLACK node 11 in the eastern direction; similarly, GRAY node *K* is the eastern neighbor of node 15. Neighbors in the direction of a corner of a block are determined analogously, e.g., node 15 is a neighbor of BLACK node 14 in the *NE* direction and GRAY node *M* is a neighbor of node 15 in the *SE* direction.

The expression of operations involving a block's quadrants and boundaries is facilitated by the following predicates and functions. ADJ(*B*, *I*) is true if and only if quadrant *I* is adjacent to boundary *B* of the node's block, e.g., ADJ(*W*, *SW*) is true. REFLECT(*B*, *I*) yields the SONTYPE value of the block of equal size that is adjacent to side *B* of a block having SON-

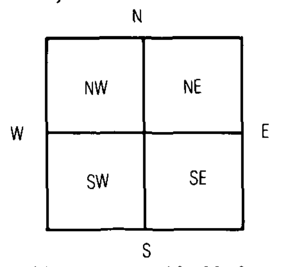


FIGURE 8. Relationship between a block's four quadrants and its boundaries.

TYPE value I, e.g., REFLECT(N, SW) = NW, REFLECT(E, SW) = SE, REFLECT(S, SW) = NW, and REFLECT(W, SW) = SE. CSIDE(B) is the side that is adjacent to side B in the clockwise direction, e.g., CSIDE(N) = E. COMMONSIDE($Q1$, $Q2$) indicates the boundary of the block containing quadrants $Q1$ and $Q2$ that is common to them (if $Q1$ and $Q2$ are not adjacent quadrants, then the value of COMMONSIDE($Q1$, $Q2$) is undefined), e.g., COMMONSIDE(NW, SW) = W. QUAD($S1$, $S2$) is the quadrant bounded by boundaries $S1$ and $S2$ (if $S1$ and $S2$ are not adjacent boundaries, then QUAD($S1$, $S2$) is undefined), e.g., QUAD(N, W) = NW. Similarly, OPQUAD(Q) is a quadrant which is diagonally facing quadrant Q, e.g., OPQUAD(NW) = SE, OPQUAD(NE) = SW.

The algorithm that is described is different from Algorithm B in that it has been modified to avoid having to process GRAY sons before processing their BLACK and WHITE brothers. Instead, whenever a BLACK block, P, has been found to be subsumed by an adjacent BLACK block, Q, then P's NODETYPE field is changed to WHITE but its DIST field is left alone. This ensures that application of the QMAT algorithm to any of P's yet unprocessed GRAY brothers will result in their subsumption by P if appropriate. Note that when Q is a genuine WHITE block, $D(P, Q)$ is negative since DIST(Q) is zero, and thus P cannot be subsumed by Q, i.e., $D(P, Q) < DIST(P)$. Once all of a GRAY node's sons have been processed, a check is made if they all correspond to WHITE blocks. If yes, then they and their father are replaced by a node having NODETYPE and DIST field values of WHITE and zero respectively. Otherwise, the DIST field of any son corresponding to a WHITE block is set to zero.

The main procedure is termed QMAT and is invoked with a pointer to the root of the quadtree representing the image and an integer corresponding to the log of the diameter of the image (e.g., n for a $2^n \times 2^n$ image array). We assume that each block's distance has already been computed by a method such as that described in [23]. QMAT traverses the tree and controls the examination of the eight neighbors of each BLACK node. Note that our algorithm results in transforming the original quadtree to a QMAT by overwriting the original quadtree. This is not necessary. An alternative algorithm would create copies of the nodes while forming the QMAT. In fact, the only modification to our algorithm is to create a copy of each node prior to examining its neighbors.

Procedure GTEQUAL_ADJ_NEIGHBOR locates a neighboring node of greater or equal size along a specified side (e.g., N, E, S, or W). If the node is on the edge of the image, then no neighbor exists in the specified direction and NULL is returned (e.g., the western neighbor of node 1 in Figure 1b). If the node is not on the edge of the image and no neighboring BLACK or WHITE node exists satisfying our size criteria, then a pointer to a GRAY node of equal size is returned (e.g., the eastern neighbor of node 1 in Figure 1b yields a pointer to GRAY node B). Procedure GTEQUAL_CORNER_NEIGHBOR is analogous to GTEQUAL_ADJ_NEIGHBOR and locates a neighboring node of greater or equal size along a corner (e.g., NW, NE, SE, or SW).

As an example of the application of the algorithm, consider the region given in Figure 1a. Figure 1b is the corresponding block decomposition while Figure 1c is its quadtree representation. All of the BLACK nodes have labels ranging from 1 to 20 while the WHITE nodes have labels ranging from 21 to 43. The GRAY nodes have labels ranging between A and N. The BLACK nodes have been labeled in the order in which their subsuming adjacencies were explored by procedure QMAT. Figure 1d contains the chessboard distance transform corresponding to Figure 1b. Figures 1e and 1f contain the block decomposition of the QMAT and the quadtree representation of the QMAT corresponding to Figure 1b respectively.

```
procedure QMAT(P, LEVEL);
/* Given a quadtree rooted at node P spanning a 2 ↑ LEVEL
   × 2 ↑ LEVEL space, find its corresponding quadtree medial
   axis transform */
begin
  value node P;
  value integer LEVEL;
  integer L;
  quadrant I;
  direction D;
  if BLACK(P) then
    begin
      for D in {'N', 'E', 'S', 'W'} do
        begin
          L ← LEVEL;
          GTEQUAL_ADJ_NEIGHBOR(P, D, Q, L);
          if not NULL(Q) and
            DIST(Q) − 2 ↑ (L − 1) − 2 ↑ (LEVEL −1) =
            DIST(P) then
            begin /* P is subsumed by its neighbor Q */
              NODETYPE(P) ← 'WHITE';
              return;
            end;
          L ← LEVEL;
          GTEQUAL_CORNER_NEIGHBOR(
            P, QUAD(D,CSIDE(D)), Q, L);
          if not NULL(Q) and
            DIST(Q) − 2 ↑ (L − 1) − 2 ↑ (LEVEL − 1)
            =DIST(P) then
            begin/* P is subsumed by its neighbor Q */
              NODETYPE(P) ← 'WHITE';
              return;
            end;
        end;
    end
  else if GRAY(P) then
    begin
      for I in {'NW', 'NE', 'SE', 'SW'} do
         QMAT(SON(P, I), LEVEL − 1);
      if WHITE(SON(P, 'NW')) and WHITE(SON(P, 'NE')) and
        WHITE(SON(P, 'SE')) and WHITE(SON(P, 'SW')) then
        begin /* Merge the four sons */
          NODETYPE(P) ← 'WHITE';
          for I in {'NW', 'NE', 'SE', 'SW'} do
            begin
              RETURNTOAVAIL (SON(P, I));
              SON(P, I) ← NULL;
            end;
        end
      else /* All four sons are not WHITE or subsumed */
        begin
          for I in {'NW', 'NE', 'SE', 'SW'} do
            begin
              if WHITE(SON(P, I)) then
                DIST(SON(P, I) ← Q;
            end;
        end;
    end;
end; /* WHITE nodes are left alone */
```

```
procedure GTEQUAL_ADJ_NEIGHBOR(P, D, Q, L);
/* Return in Q the neighbor of node P in horizontal or verti-
   cal direction D which is greater than or equal in size
   to P. If such a node does not exist, then a GRAY node of
   equal size is returned. If this is also impossible, then the
   node is adjacent to the border of the image and NULL is
   returned. L denotes the level of the tree at which node P is
   initially found and the level of the tree at which Q is
   finally found */
begin
  value node P;
  reference node Q;
  value direction D;
  reference integer L;
  $L \leftarrow L + 1$;
  if not NULL(FATHER(P)) and ADJ(D, SONTYPE(P)) then
    /* Find a common ancestor */
    GTEQUAL_ADJ_NEIGHBOR(FATHER(P), D, Q, L)
  else $Q \leftarrow$ FATHER(P);
  /* Follow the reflected path to locate the neighbor */
  if not NULL(Q) and GRAY(Q) then
    begin
      $Q \leftarrow$ SON(Q, REFLECT(D, SONTYPE(P)));
      $L \leftarrow L - 1$;
    end;
end;
```

```
procedure GTEQUAL_CORNER_NEIGHBOR(P, C, Q, L);
./* Return in Q the neighbor of node P in the direction of
   corner C of P which is greater than or equal in size to P. If
   such a node does not exist, then a GRAY node of equal
   size is returned. If this is also impossible, then the node is
   adjacent to the border of the image and NULL is returned.
   L denotes the level of the tree at which node P is initially
   found and the level of the tree at which Q is finally found
   */
begin
  value node P;
  reference node Q;
  value quadrant C;
  reference integer L;
  $L \leftarrow L + 1$;
  if not NULL(FATHER(P)) and SONTYPE(P) $\neq$ OPQUAD(C)
    then
    if SONTYPE(P) = C then
      GTEQUAL_CORNER_NEIGHBOR
        (FATHER(P), C, Q, L)
    else GTEQUAL_ADJ_NEIGHBOR
           (FATHER(P),
           COMMONSIDE(SONYTYPE(P), C), Q, L)
  else Q $\leftarrow$ FATHER(P);
  /* Follow the opposite path to locate the neighbor */
  if not NULL(Q) and GRAY(Q) then
    begin
      $Q \leftarrow$ SON(Q, OPQUAD(SONTYPE(P)));
      $L \leftarrow L - 1$;
    end;
end;
```

5. ANALYSIS

The running time of the QMAT computation algorithm is measured by the number of nodes that are visited and by the size of the quadtree. For each BLACK node, we must visit a minimum of one and a maximum of eight neighbors of greater than or equal size in order to determine whether the block corresponding to the node is subsumed, i.e., contained in a square centered at its neighbor (e.g., Block 14 is subsumed by Block 15 in Figure 1b). Clearly, for each BLACK node, the worst case in terms of the number of nodes that must be visited arises when the neighbor that is being sought is of equal size (e.g., the *NE* neighbor of Block 11 in Figure 1b, i.e., Block 5). Thus we only need to analyze the amount of work performed by procedures GTEQUAL_ADJ_NEIGHBOR, GTEQUAL_CORNER_NEIGHBOR, and QMAT.

Our analysis assumes a $2^n \times 2^n$ random image i.e., a BLACK node is equally likely to appear in any position and level in a quadtree. This means that all configurations of adjacent nodes of varying sizes are assumed to have equal probability. This is different from the more conventional notion of a random image which implies that every block at level 0 (i.e., pixel) has an equal probability of being BLACK or WHITE. Such an assumption would lead to a very low probability of any node corresponding to blocks of sizes larger than 1. Clearly, for such an image the quadtree is the wrong representation. The analysis closely parallels that performed in [23] for the chessboard distance transform.

LEMMA 2: *The average of the maximum number of nodes visited by each invocation of* GTEQUAL_ADJ_NEIGHBOR *is less than* 4.

PROOF: Given a node *P* at level *i* and a horizontal or vertical direction *D*, there are $2^{n-i} \cdot (2^{n-i} - 1)$ possible positions for node *P* and a neighbor at level *i* and direction *D*. Of these $2^{n-i} \cdot (2^{n-i} - 1)$ neighbor pairs, $2^{n-i} \cdot 2^0$ have their nearest common ancestor at level *n*, $2^{n-i} \cdot 2^1$ at level $n - 1, \ldots$, and $2^{n-i} \cdot 2^{n-i-1}$ at level $i + 1$. For each node at level *i* having a common ancestor at level *j*, the maximum number of nodes that will be visited by GTEQUAL_ADJ_NEIGHBOR is $(j - i) + (j - i) = 2 \cdot (j - i)$. Assuming that node *P* is equally likely to occur at any level *i* and at any of the $2^{n-i} \cdot (2^{n-i} - 1)$ positions at level *i*, then the average of the maximum number of nodes visited by GTEQUAL_ADJ_NEIGHBOR is

$$\frac{\sum_{i=0}^{n-1} \sum_{j=i+1}^{n} 2^{n-i} \cdot 2^{n-j} \cdot 2(j-i)}{\sum_{i=0}^{n-1} 2^{n-i} \cdot (2^{n-i} - 1)} \tag{1}$$

Making use of the following identities in the numerator of (1) leads to (4):

$$\sum_{j=0}^{n-1-i} \frac{j}{2^j} = 2 - \frac{n+1-i}{2^{n-1-i}} \tag{2}$$

$$\sum_{j=0}^{n-1-i} \frac{1}{2^j} = 2 \cdot \left(1 - \frac{1}{2^{n-i}}\right) \tag{3}$$

$$4/3 \cdot 2^{2n+2} - (n+1) \cdot 2^{n+2} - 4/3 \tag{4}$$

The denominator of (1) can be manipulated in a similar manner to yield

$$1/3 \cdot (2^{2n+2} - 3 \cdot 2^{n+1} + 2) \tag{5}$$

Substituting (4) and (5) into (1) results in

$$4 - \frac{3 \cdot (n-1) \cdot 2^{n+2} + 12}{2^{2n+2} - 3 \cdot 2^{n+1} + 2}$$

$$\approx 4 \text{ as } n \text{ gets large}$$

$$< 4$$

Q.E.D.

LEMMA 3: *The average of the maximum number of nodes visited by each invocation of* GTEQUAL_CORNER_NEIGHBOR *is less than* $^{16}/_{3}$.

PROOF: Given a node P at level i and a diagonal direction C, there are $(2^{n-i}-1)^2$ possible positions for node P and a neighbor at level i in direction C. Of these $(2^{n-1}-1)^2$ neighbor pairs, $4^0 \cdot (2 \cdot (2^{n-i}-1)-1)$ have their nearest common ancestor at level n, $4^1 \cdot (2 \cdot (2^{n-i-1}-1)-1)$ at level $n-1$; ... and $4^{n-i-1} \cdot (2 \cdot (2^{n-i-(n-i-1)}-1)-1)$ at level $i+1$. In order to see this, consider Figure 9 where a grid is shown for $n = 3$. If all BLACK and WHITE nodes are at level 0, then for a neighbor in the *NE* direction we see that nodes along the fifth row and fourth column have their nearest common ancestor at level 3 (i.e., 13 nodes labeled 1–13). Continuing the process for the *NW*, *NE*, *SW*, and *SE* quadrants of Figure 9, we find that all neighbor pairs contained exclusively within these quadrants have their nearest common ancestor at a level ≤ 2. In particular, for the *NW* quadrant, nodes along the third row and second column have their nearest common ancestor at level 2 (i.e., 5 nodes labeled 14–18). The *NE*, *SW*, and *SE* quadrants are analyzed in a similar manner. This process is applied to the four subquadrants of the quadrants to obtain the neighbor pairs whose nearest common ancestor is at level 1. Note that we had to consider every row in the image when analyzing diagonal neighbor pairs whereas we only needed to consider one row or column when analyzing neighbor pairs in the *N*, *E*, *S*, and *W* directions. This is necessary because for diagonal neighbors, each row in the image has a different number of neighbor pairs with a common ancestor at a given level while this number is constant for each row or column when considering neighbor pairs in the horizontal and vertical directions.

For each node P at level i having a common ancestor at level j, the maximum number of nodes that will be visited by GTEQUAL_CORNER_NEIGHBOR is $(j-i)+(j-i) = 2 \cdot (j-i)$. Assuming that node P is equally likely to occur at any level i and at any of the $(2^{n-i}-1)^2$ positions at level i, then the average of the maximum number of nodes visited by GTEQUAL_CORNER_NEIGHBOR is

$$\frac{\sum_{i=0}^{n-1} \sum_{j=i+1}^{n} 4^{n-j} \cdot (2 \cdot (2^{n-i-(n-j)}-1)-1) \cdot 2 \cdot (j-i)}{\sum_{i=0}^{n-1} (2^{n-i}-1)^2} \tag{6}$$

Making use of the following identities in addition to those of (2) and (3) in the numerator of (6) leads to (9) (for further details, see [24]):

$$\sum_{j=0}^{n-1-i} \frac{j}{2^{2j}} = \frac{1}{9} \cdot \left(4 - \frac{3 \cdot (n-1-i)+4}{2^{2n-2-2i}}\right) \tag{7}$$

$$\sum_{j=0}^{n-1-i} \frac{1}{2^{2j}} = \frac{4}{3} \cdot \left(1 - \frac{1}{2^{2n-2i}}\right) \tag{8}$$

$$^{1}/_{9} \cdot 2^{2n+6} - (n+1) \cdot 2^{n+3} + n^2 + {}^{11}/_{3} \cdot n + {}^{8}/_{9} \tag{9}$$

The denominator of (6) can be manipulated in a similar manner to yield

$$^{1}/_{3} \cdot (2^{2n+2} - 3 \cdot 2^{n+2} + 3 \cdot n + 8) \tag{10}$$

Substituting (9) and (10) into (6) results in

$$= \frac{16}{3} - \frac{(6 \cdot n - 10) \cdot 2^{n+2} - 3 \cdot n^2 + 5 \cdot n + 40}{2^{2n+2} - 3 \cdot 2^{n+2} + 3 \cdot n + 8}$$

$$< \frac{16}{3}$$

Q.E.D.

It is also useful to obtain the number of nodes in the quadtree. Letting B and W correspond to the number of BLACK and WHITE, respectively, leaf nodes in the quadtree we have

LEMMA 4: *The number of nodes in a quadtree having B and W leaf nodes is bounded by* $^{4}/_{3} \cdot (B+W)$.

PROOF: See Lemma 1 in [18].

We can now prove our main result.

THEOREM 3: The average execution time of the QMAT computation algorithm is $O(B)$.

PROOF: From Lemmas 2 and 3 we have that for each side and corner of a BLACK node, GTEQUAL_ADJ_NEIGHBOR and GTEQUAL_CORNER_NEIGHBOR result in an average of $4 + {}^{16}/_{3} = 9^{1}/_{3}$ nodes being visited. There are four sides and corners for each BLACK node. Thus the four invocations of these procedures contribute $4B \cdot 9^{1}/_{3}$. From Lemma 4, we have that the number of nodes in the quadtree is bounded by $^{4}/_{3} \cdot (B+W)$. This quantity correlates with the work performed by procedure QMAT since each node in the quadtree is visited by the traversal. Summing up these values we have $4 \cdot B \cdot 9^{1}/_{3} + {}^{4}/_{3} \cdot (B+W) = {}^{4}/_{3} \cdot (29 \cdot B + W)$ which is of order B. *Q.E.D.*

The algorithm has an execution time complexity of the same order of magnitude as the one developed in [23] for the computation of the chessboard distance transform, i.e., $^{4}/_{3} \cdot (43 \cdot B + W)$. Observe that a direct implementation of Algorithm A of Section 3 would require, at worse, work proportional to $B \uparrow 2$ since sorting is required (a $B \cdot \log B$ operation) as well as checking every BLACK block against every other BLACK block for subsumption. Actually, it is conceivable that a $B \cdot \log B$ procedure for subsumption exists which would render Algorithm A to be $O(B \cdot \log B)$ which would still be inferior to Algorithm A.

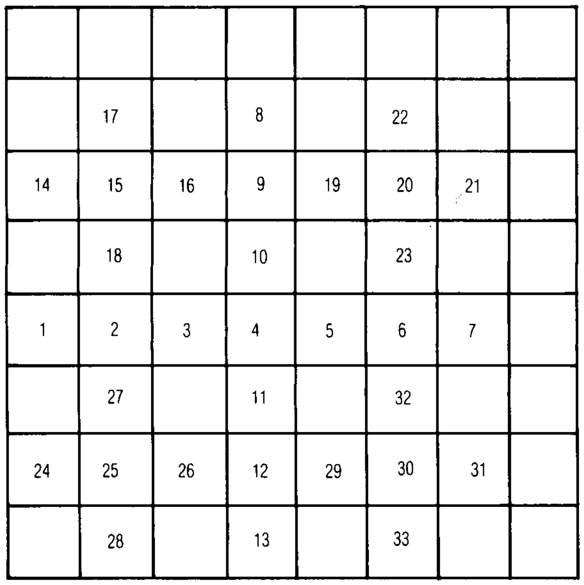


FIGURE 9. Sample grid illustrating blocks whose nodes are at level 0 and whose nearest common ancestor is at level ≤2 when attempting to locate a NE neighbor.

6. CONCLUDING REMARKS

The concept of a skeleton and medial axis transform have been adapted to images represented by quadtrees and have resulted in the definition of a new data structure termed the QMAT. The QMAT results in a partition of the image into a set of nondisjoint squares having sides whose lengths are sums of powers of 2 rather than, as is the case with quadtrees, a set of disjoint squares having sides of lengths which are powers of 2. An algorithm for the computation of the QMAT given a quadtree representation has been presented and shown to have an average execution time of $O(B)$ where B corresponds to the number of BLACK blocks comprising the objects of the image.

The algorithm and its analysis are somewhat similar to those used in the computation of the chessboard distance transform [23]. The difference is that for each BLACK node only its neighbors of greater or equal size needed to be examined and not their progeny as was the case in [23]. A new result of our analysis is that finding a corner neighbor is approximately 4/3 as complex as finding an adjacent neighbor. The algorithm in its present form could not be combined with the computation of the chessboard distance transform and done in one pass, *i.e.*, it requires a separate traversal of the tree, since computation of the QMAT relies on knowledge of the distance transform values of a node's neighbors.

The algorithm can be varied in several ways. First, in its present state, procedure QMAT overwrites the existing quadtree. It may be desirable to have an algorithm which constructs the QMAT while retaining the original quadtree. This is quite simple and can be accomplished by modifying procedure QMAT to allocate a node each time it visits one in the original quadtree. Note that our analysis assumed that all eight neighbors of a node are visited while attempting to ascertain if it is subsumed by one of its neighbors. In fact, we cease processing as soon as subsumption is found to occur. Another observation is that when overlap exists (e.g., in Figure 1b, block 10 is the *NW* neighbor of block 12 and is also its northern neighbor) we need not invoke GTEQUAL_ADJ_NEIGHBOR or GTEQUAL_CORNER_NEIGHBOR for the neighbor which overlaps the two directions. However, such a variation is of little value since the number of neighbors can be shown to range between 5 and 8.

Our view of the quadtree medial axis transform as an alternative image representation serves to reinforce the appropriateness of the chessboard distance metric for quadtrees. In particular, the analogy between squares and circles as the basis for the medial axis transform of a quadtree is noteworthy. Also, notice the similarity between the process of obtaining the QMAT and thinning [15] an image.

The advantage of the QMAT is in its compactness (e.g., recall Figures 1 and 5) and in its decreased sensitivity to shift (recall Figures 6 and 7). In the worst case, the QMAT is identical to the quadtree. The medial axis transform is often used as an alternative to a border representation because of its amenability to the determination of whether or not a given point lies within a particular region [15]. This is not a problem when the quadtree representation is used. However, in the case of the QMAT this is slightly more complex since a WHITE block does not necessarily imply that the entire space spanned by the block is WHITE. In such a case the WHITE node's surrounding neighbors will have to be examined.

In Section 2, we saw that there are two ways of defining a quadtree skeleton with a small difference in the QMAT although the QMAT was shown to require the same number of nodes in either case. Using Property (3) resulted in a simpler QMAT construction algorithm while using Property (3′) results in obtaining a quadtree skeleton of less than or equal size. Since we are primarily interested in storage compactness in the form of a tree, the difference was not important. However, we also wish to be able to reconstruct the quadtree from its QMAT. In such a case, the reconstruction process is conceivably faster given a quadtree skeleton definition that incorporates Property (3′) since the quadtree skeleton is less than or equal in size (e.g., 2 nodes versus 3 nodes in Figures 4b and 4c respectively). In essence, to reconstruct a quadtree from its QMAT, we must traverse the QMAT, and for each BLACK node t_i add all elements of $S(t_i)$ to the quadtree. This can be done by using the neighbor techniques employed in the construction of a quadtree from its run length [17] and boundary code [20] representations. See [25] for an algorithm to reconstruct a quadtree from its QMAT with a QMAT defined using Property (3).

Fruitful subjects for future research include the investigation of algorithms for set operations such as intersection and union as well as connectivity and perimeter using the QMAT representation. A more thorough study of the relationship between the amount of space occupied by a quadtree and its QMAT would also be welcome.

Acknowledgments. I would like to thank Kathryn Riley, Janet Salzman, and Pat Young for help with the preparation of the manuscript. I have benefited greatly from discussions with Charles R. Dyer, Shmuel Peleg, and Azriel Rosenfeld.

REFERENCES

1. Blum, H. A transformation for extracting new descriptors of shape. In Wathen-Dunn, W., ed., *Models for the Perception of Speech and Visual Form*, M.I.T. Press, Cambridge, Massachusetts, 1967, pp. 362–380.
2. Duda, R.O. and Hart, P.E. *Pattern Classification and Scene Analysis*, Wiley-Interscience, New York, 1973.
3. Dyer, C.R. Computing the Euler number of an image from its quadtree, *Comput. Gr. Image Process.* *13*, 3 (1981), 270–276.
4. Dyer, C.R. Rosenfeld, A., and Samet, H. Region representation: Boundary codes from quadtrees, *Commun. ACM 23*, 3 (March 1980), 171–179.
5. Hunter, G.M. Efficient computation and data structures for graphics, Ph.D. dissertation, Department of Electrical Engineering and Computer Science, Princeton University, Princeton, New Jersey, 1978.
6. Hunter, G.M. and Steiglitz, K. Operations on images using quadtrees, *IEEE Trans. Pattern Anal. Machine Intell. 1*, 2 (1979), 145–153.
7. Hunter, G.M., and Steiglitz, K. Linear transformation of pictures represented by quadtrees, *Comput. Gr. Image Process. 10*, 3 (1979), 289–296.
8. Jackins, C.L. and Tanimoto, S.L. Octrees and their use in representing three-dimensional objects, *Comput. Gr. Image Process. 14*, 3 (1980), 249–270.
9. Klinger, A. Patterns and search statistics. In *Optimizing Methods in Statistics*, J. S. Rustagi, ed., Academic Press, New York, 1971.
10. Klinger, A. and Dyer, C.R. Experiments in picture representation using regular decomposition, *Comput. Gr. Image Process. 5*, 1 (1976), 68–105.
11. Meagher, D. Geometric modeling using octree encoding, *Comput. Gr. Image Process. 19*, 2 (1982), 129–147.
12. Naur, P. (ed.), Revised report on the algorithmic language ALGOL 60, *Commun. ACM*, (May 1960), 299–314.
13. Newman, W.M. and Sproull, R.F. *Principles of Interactive Computer Graphics*, 2nd ed., McGraw-Hill, New York, 1971.
14. Pfaltz, J.L. and Rosenfeld, A. Computer representation of planar regions by their skeletons, *Commun. ACM 10*, 2 (February 1967), 119–122.
15. Rosenfeld, A. and Kak, A.C. *Digital Picture Processing*, Academic Press, New York, 1976.
16. Rutovitz, D. Data structures for operations on digital images. In *Pictorial Pattern Recognition*, G. C. Cheng et al., eds., Thompson Book Co., Washington, DC, 1968, 105–133.
17. Samet, H. Region representation: Quadtrees from boundary codes, *Commun. ACM, 23*, 2 (March 1980), 163–170.

18. Samet, H. Computing perimeters of images represented by quadtrees, *IEEE Trans. Pattern Anal. Machine Intell. 3*, 6 1981, 683-687.
19. Samet, H. Connected component labeling using quadtrees, *J. ACM 28*, (July 1981), 487-501.
20. Samet, H. An algorithm for converting rasters to quadtrees, *IEEE Trans. Pattern Anal. Machine Intell. 3*, (1981), 93-95.
21. Samet, H. Region representation: quadtrees from binary arrays, *Comput. Gr. Image Process.* 13, 1980, 88-93.
22. Samet, H. Algorithms for the conversion of quadtrees to rasters, Computer Science TR 979, University of Maryland, College Park, Maryland, November 1980.
23. Samet, H. Distance transform for images represented by quadtrees, *IEEE Trans. Pattern Anal. Machine Intell. 4*, 3 (1982), 298-303.
24. Samet, H. A quadtree medial axis transform, Computer Science TR-803, University of Maryland, College Park, Maryland, August 1979.
25. Samet, H. Reconstruction of quadtrees from quadtree medial axis transforms, Computer Science TR-1224, University of Maryland College Park, Maryland, October 1982.
26. Shneier, M. Path-length distances for quadtrees, *Inf. Sci. 23*, 1 1981, 49-67.
27. Shneier, M. Calculations of geometric properties using quadtrees, *Comp. Gr. Image Process. 16*, 3 (1981), 296-302.
28. Sutherland, I.E., Sproull, R.F., and Schumaker, R.A. A characterization of ten hidden surface algorithms, *ACM Comput. Surv. 6*, 1 (1974), 1-55.
29. Warnock, J.E. A hidden surface algorithm for computer generated halftone pictures, Computer Science Department, TR 4-15, University of Utah, June 1969.
30. Yau, M. and Srihari, S.N. Recursive generation of hierarchical data structures for multidimensional digital images, *Proceedings IEEE PRIP 81*, Dallas, Texas, 1981, 42-44.

CR Categories and Subject Descriptors: E.1 [**Data Structures**]: Trees; I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling—*geometric algorithms, object representations*; I.4.2 [**Image Processing**]: Compression—*exact coding*; I.4.7 [**Image Processing**]: Feature Measurement—*size and shape*, I.2.10 [**Artificial Intelligence**]: Vision and Scene Understanding—*representations, data structures, and transforms, shape, motion*

General Terms: Algorithms

Additional Key Words and Phrases: quadtrees, skeletons, size, shape, medial axis transforms

Received 7/81; revised 5/82; accepted 9/82

Abstracts from Other ACM Publications

ACM Transactions on Mathematical Software September Issue

A Quantitative Evaluation of the Feasibility of, and Suitable Hardware Architectures for, an Adaptive, Parallel Finite-Element System
Pamela Zave and George E. Cole, Jr.

An experimental implementation of a design for an adaptive, parallel finite-element system is described. The implementation was used to simulate the performance of this design on several microprocessor-based multiprocessor architectures. The real-time speedup observed was architecture-independent, but limited. Nevertheless, the approach to data segmentation and management worked well and has interesting applications.

For Correspondence: P. Zave, Bell Laboratories, Room 3D-426, Murray Hill, NJ 07974.

The Numerical Solution of Separably Stiff Systems by Precise Partitioning
David S. Watkins and Ralph W. Hansonsmith

Most codes for solving stiff systems of ordinary differential equations spend most of their time solving systems of linear equations with coefficient matrix $I - h\beta J$, where J is the Jacobian matrix of the system. The precise partitioning method solves these systems approximately by partitioning the Jacobian into stiff and nonstiff parts. The method resembles a method proposed by Enright and Kamel [*ACM Transactions on Mathematical Software* 5, 4(Dec. 1979), 374-385], but differs in that the partition of the Jacobian matrix is more precise. In this paper the method is implemented by revising a popular code. Numerical results indicate that for systems of more than about ten equations with few stiff eigenvalues, the precise partitioning method can save CPU time compared to the unmodified code.

For Correspondence: D. S. Watkins, Dept. of Pure and Applied Mathematics, Washington State University, Pullman, WA 99164-2930.

The Multifrontal Solution of Indefinite Sparse Symmetric Linear Equations
I. S. Duff and J. K. Reid

We extend the frontal method for solving linear systems of equations by permitting more than one front to occur at the same time. This enables us to develop code for general symmetric systems. We discuss the organization and implementation of a multifrontal code which uses the minimum-degree ordering and indicate how we can solve indefinite systems in a stable manner. We illustrate the performance of our code both on the IBM 3033 and on the CRAY-1.

For Correspondence: I. S. Duff, Computer Science and Systems Division, AERE Harwell, Oxford OX11 ORA, England.

Space-Efficient Implementations of Graph Search Methods
Robert E. Tarjan

Several space-efficient implementations of the two most common and useful kinds of graph search, namely, breadth-first search and depth-first search, are discussed. A straightforward implementation of each method requires n bits and $n + O(1)$ pointers of auxiliary storage, where n is the number of vertices in the graph. We devise methods that need only $2n + m$ bits, of which m are read-only, where m is the number of edges in the graph. We save space by folding the queue or stack required by the search into the graph representation; two of our methods for depth-first search are variants of the Deutsch-Schorr-Waite list-marking algorithm. Our algorithms are expressed in a version of Dijkstra's guarded command language.

For Correspondence: R. E. Tarjan, Bell Laboratories, Murray Hill, NJ 07974.

A Sparse Matrix Package—Part II: Special Cases
J. M. McNamee

A set of subroutines is described for combining pairs of sparse matrices in special cases in which one of the matrices may be regarded as full, and/or a vector, and/or an elementary matrix. Tests show that in many cases the new routines are faster than an earlier set of more general-purpose routines. Also, a new (faster) routine is given for multiplying two sparse matrices.

For Correspondence: J. M. McNamee, Dept. of Computer Science and Mathematics, Atkinson College, York University, Downsview, Ontario M3J 2R7, Canada.

HURRY: An Acceleration Algorithm for Scalar Sequences and Series
Theodore Fessler, William F. Ford and David A. Smith

We present a general acceleration algorithm for alternating and monotone scalar sequences and series. The main components of the algorithm are the subroutines WHIZ and HURRY. WHIZ is a recursive implementation of Levin's u transform, and HURRY (which calls WHIZ) estimates truncation and round-off errors to make a near-optimal stopping decision and provide a very good estimate of the accuracy of the computed answer.

We also present a test driver program that demonstrates the capabilities of HURRY when applied to a wide variety of convergent and divergent sequences and series.

For Correspondence: T. Fessler, Computer Services Division, NASA Lewis Research Center, Cleveland, OH 44135.