LEVEL II

UNIVERSITY OF MARYLAND
COMPUTER SCIENCE CENTER

COLLEGE PARK, MARYLAND
20742

(1) LEVEL II

(14) CSC-TR-756
DAAG-53-76C-0138

(11) April 1979

(6) CONNECTED COMPONENT LABELING USING QUADTREES

(10) Hanan Samet
Computer Science Department
University of Maryland
College Park, MD 20742

(9) Technical rept.

(15) DAAG53-76-C-0138,
DARPA Order-3206

(12) 30

ABSTRACT

An algorithm is presented for labeling the connected components of an image represented by a quadtree. The algorithm proceeds by exploring all possible adjacencies for each node once and only once. Once this is done, any equivalences generated by the adjacency labeling phase are propagated. Analysis of the algorithm reveals that its worst case average execution time is bounded by a quantity proportional to the product of the log of the region's diameter and the number of blocks comprising the area spanned by the components.

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, as is the help of Kathryn Riley in preparing this paper. The author has benefited greatly from discussions with Charles R. Dyer and Azriel Rosenfeld. He also thanks Pat Young for her help with the figures.

403 018

JCB

1. Introduction

Connected component labeling is a basic operation in image processing [RK]. The standard labeling algorithms use either an array or run-length representation for the two-level ("binary") image whose components are to be labeled. In this paper we present an algorithm for labeling the connected components of 1's in a binary image that is represented by a quadtree ([Klinger,DRS,Sametl]).

We assume that the given binary image is a 2^n by 2^n array of unit square "pixels." The quadtree is an approach to image representation based on successive subdivision of the array into quadrants. In essence, we repeatedly subdivide the array into quadrants, subquadrants,..., until we obtain blocks (possibly single pixels) which consist entirely of either 1's or 0's. 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. For example, Figure 1b is a block decomposition of the region in Figure 1a while Figure 1c is the corresponding quadtree. In general, BLACK and WHITE square nodes represent nodes consisting entirely of 1's and of 0's, respectively. Circular nodes, also termed GRAY nodes, denote non-terminal nodes.

Sections 2-6 present and analyze our algorithm. Included is a formal description of the algorithm along with motivating considerations. The actual algorithm is given using a variant of ALGOL 60 [Naur].

2. Definitions and Notation

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. Give a node P and a son I, these fields are referenced as FATHER(P) and SON(P,I) respectively. At times it is useful to use the function SONTYPE(P) where SONTYPE(P) = Q iff SON(FATHER(P), Q) = P. The sixth field, named NODETYPE, describes the contents of the block of the image which the node represents--i.e., WHITE, if the block contains no 1's; BLACK, if the block contains only 1's, and GRAY, if it contains pixels of both types. Alternatively, BLACK and WHITE nodes are terminal nodes while GRAY nodes are non-terminal nodes. The seventh field, named REGION, identifies the connected component containing the block represented by the node. This field is only meaningful for BLACK nodes. It is set as a result of the connected component labeling algorithm. LABELED(P) indicates if node P has already been labeled.

Let the four sides of a node's block be called its N, E, S, and W sides. They are also termed its boundaries. The inter-relationship between a block's four quadrants and its boundaries is facilitated by use of the predicate ADJ and the function REFLECT ADJ(B,I) is true if and only if quadrant I is adjacent to boundary B of the node's block; e.g., ADJ(N,NE) is true. REFLECT(B,I) yields the quadrant which is adjacent to quadrant I

along boundary B of the block represented by I; e.g., REFLECT(W,NW)=NE, REFLECT(E,NW)=NE, REFLECT(N,NW)=SW, and REFLECT(S,NW)=SW. Figure 2 shows the relationship between the quadrants of a node and its boundaries.

Given a quadtree corresponding to a 2^n by 2^n array, we say that the root node is at level n, and that a node at level i is at a distance of n-i from the root of the tree. In other words, for a node at level i, we must ascend n-i FATHER links to reach the root of the tree. Note that the farthest node from the root of the tree is at level ≥Ø.

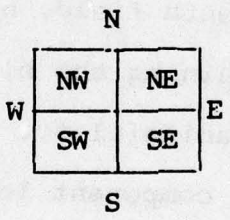

Figure 2. Relationship between a block's four quadrants and its boundaries.

3. Informal description of the algorithm

The connected component labeling algorithm has three phases. The first phase traverses the tree and explores all possible adjacencies between pairs of BLACK nodes. During this process all BLACK nodes are labeled. Should any equivalences be discovered between regions already labeled, then their component identifiers are added to a list of pairs of equivalences. Once the entire tree is traversed in this manner, the second phase processes pairs of equivalences to yield equivalence classes (e.g., [Knuth, Tarjan]). Finally, the third phase traverses the tree one more time with all members of an equivalence class being assigned the same component identifier (i.e., label).

Phase one traverses the tree in postorder (i.e., the sons of a node are visited first). In particular, the sons are visited in the order NW, NE, SW, and SE. For each BLACK terminal node, say P, we explore the eastern and southern adjacencies. This means that all of the node's BLACK adjacent southern and eastern neighbors are visited. If they have not been previously visited, then they are labeled with the label of P. If P does not already have a label, then it is assigned the label of one of its adjacent neighbors if it has a label. If adjacent BLACK nodes have already been assigned labels that are different, then the labels are added to the list of equivalences that will be merged in the second phase.

The key to the algorithm is that phase one assures that every adjacency of two BLACK nodes will be explored once and only once. To see this, note that the traversal starts at the NW-most son, if possible, and the brothers are traversed in the order NW, NE, SW, and SE. Clearly, by the time any BLACK node is visited, its northern and western adjacencies have already been explored. Thus the northern and western adjacencies need not be reexplored. This is because each node labels all of its adjacent eastern and southern neighbors.

Note the analogy between phase one and the algorithm for computing the total perimeter of an image represented by a quadtree [Samet2]. When computing perimeter we must explore adjacencies of BLACK and WHITE nodes rather than adjacencies of BLACK and BLACK nodes. Besides the duality in the type of adjacency, there is only one other difference. The perimeter computation algorithm requires that adjacencies in four directions need to be explored whereas adjacencies in only two directions need to be explored in phase one. Four directions were necessary in the perimeter computation algorithm because for each pair of adjacent BLACK and WHITE nodes only the BLACK node causes the adjacency to be explored (WHITE nodes do not).

As an example of the application of the algorithm, consider the image given in Figure 1a. Figure 1b is the corresponding block decomposition and Figure 1c is its quadtree representation.

All of the BLACK nodes have numbers ranging between 1 and 41 while the WHITE nodes have numbers ranging between 42 and 91. The BLACK nodes have been numbered in the order in which they were labeled by phase one. The WHITE nodes have been numbered in the order in which they were visited (i.e., the argument to procedure LABEL). Thus node 1 has been labeled before nodes 2, 3, etc. Figure 3 shows the labels assigned to the five components. Phase two of the algorithm will merge the equivalence pair B≡D to form component 4 and the equivalence pairs F≡G and G≡H to form component 5.

4. Formal statement of the algorithm

The following ALGOL-like procedures specify the connected component labeling algorithm. Actually, we only present the procedures corresponding to the first and third phases of the algorithm. Phase two can be achieved by using a variant of algorithm E in [Knuth].

The main procedure is termed COMPONENT and is invoked with a pointer to the root of the quadtree representing the image. The global variable MERGES is used to accumulate all of the equivalence relations formed by adjacent BLACK ndoes. MERGES is subsequently processed by phase two to yield a set of equivalence classes--i.e., one class per component. LABEL implements phase one by traversing the tree and controlling the exploration of adjacent BLACK nodes. FIND_NEIGHBOR locates a neighboring node of greater or equal size along a specified border. If no such neighboring BLACK or WHITE node exists, then FIND_NEIGHBOR returns a pointer to a GRAY node of equal size. In such a case, procedure LABEL_ADJACENT continues the search recursively by examining all BLACK and WHITE adjacent neighbors of smaller size. Otherwise, LABEL_ADJACENT assigns a label to the adjacent neighbor if it is BLACK. The labels are assigned by procedure ASSIGN_LABEL. Procedure UPDATE corresponds to phase three and results in the traversal of the tree in order to propagate the equivalences thereby uniquely labeling each component.

```
procedure COMPONENT(QUADTREE);
/* label all of the connected components of the tree rooted at QUADTREE*/
begin
    node QUADTREE;
    pairlist MERGES;
    MERGES←empty;
    LABEL(QUADTREE);
    process equivalences specified by MERGES;
    UPDATE(QUADTREE);
end;

procedure LABEL(P);
/*assign labels to node P and its sons*/
begin
    node P,Q;
    quadrant I;
    if GRAY(P) then
        begin
            for I in {NW,NE,SW,SE} do LABEL(SON(P,I));
        end
    else if BLACK(P) then
        begin
            Q←FIND_NEIGHBOR(P,'E');
            if not NULL(Q) then LABEL_ADJACENT(Q,'NW','SW',P);
```

```
            Q←FIND_NEIGHBOR(P,'S');
            if not NULL(Q) then LABEL_ADJACENT(Q,'NW','NE',P);
            if not LABELED(P) then REGION(P) GENREGION( );
        end
    else return; /*a WHITE node */
end;


node procedure FIND_NEIGHBOR(P,S);
/* given node P, return a node which is adjacent to side S of node P*/
begin
    node P,Q;
    side S;
    if not NULL(FATHER(P)) and ADJ(S,SONTYPE(P)) then
        /*find a common ancestor*/
        Q←FIND_NEIGHBOR(FATHER(P),S)
    else Q←FATHER(P);
    /* follow reflected path back to locate the neighbor */
    return (if not NULL(Q) and GRAY(Q) then SON(Q,REFLECT(S,SONTYPE(P)))
            else Q);
end;
```

```
procedure LABEL_ADJACENT(R,Q1,Q2,P);
/* find all descendants of node R adjacent to node P--i.e., in
   quadrants Q1 and Q2 */
begin
    node P,R;
    quadrant Q1,Q2;
    if GRAY(R) then
        begin
            LABEL_ADJACENT(SON(R,Q1),Q1,Q2,P);
            LABEL_ADJACENT(SON(R,Q2),Q1,Q2,P);
        end
    else if BLACK(R) then ASSIGN_LABEL(P,R)
    else return; /* a WHITE node */
end;

procedure ASSIGN_LABEL(P,Q);
/* assign a label to nodes P and Q if they do not already have
   one.  If both have different labels, then enter them in MERGES */
begin
    node P,Q;
    if LABELED(P) and LABELED(Q) then
        begin
            if REGION(P)≠REGION(Q) then add <REGION(P),REGION(Q)>
            to MERGES;
        end
    else if LABELED(P) then REGION(Q)←REGION(P)
    else if LABELED(Q) then REGION(P)←REGION(Q)
    else REGION(P)←REGION(Q)←GENREGION( );
end;
```

```
procedure UPDATE(P);
/* propagate the equivalences in the quadtree rooted at P */
begin
    node P;
    quadrant I;
    if GRAY(P) then
        begin
            for I in {NW,NE,SW,SE} do UPDATE(SON(P),I);
        end
    else if BLACK(P) then REGION(P) LOOKUP (REGION(P),MERGES)
    else return;  /* a WHITE node* /
end;
```

5. Analysis

The running time of the connected component labeling algorithm is determined by the time necessary to execute its three phases. Prior to analyzing this value we first examine the spatial configurations confronted by the algorithm and how they affect its execution time. It should be clear that the greater the number of BLACK nodes, the more time is spent exploring adjacencies in phase one. Phase two is more dependent on the shape of the various components. The execution time of phase two is dominated by the number of equivalence pairs that are generated in phase one. An equivalence pair is generated whenever an adjacency of a previously labeled node is explored and it is found that the adjacent neighbor has already been assigned a different label.

The situation giving rise to the generation of an equivalence pair can be best seen by examining Figure 4. Components 1, 2, and 3 do not result in the generation of equivalence pairs because of the manner in which phase one explores adjacencies--i.e., in the eastern and southern directions thereby processing the quadrants in the order NW, NE, SW, SE. Thus we see that the nodes or blocks comprising the quadtree are processed in the order in which they are adjacent. This is not always the case for a component having the form of component 4 in Figure 4 (in this case we have the equivalence of D and E). This is especially

true when the vertical and horizontal segments are not comprised of single blocks, or have adjacent northern or western neighbors in the case of the horizontal segment. For example, in the image represented by Figure 1a we find that no equivalence pairs were generated for the components 1, 2, and 3 whereas this was not true for the components 4 and 5. In particular, we have the equivalences B≡D for component 4 and F≡G and G≡H for component 5. Note that if the block labeled 40 would have been WHITE rather than BLACK, then block 41 would have been labeled with G and no equivalence pair would have been generated.

Phase one depends on the speed of the combination of procedures LABEL_ADJACENT and FIND_NEIGHBOR. These procedures are invoked in phase one (i.e., in procedure LABEL) twice as many times as one has BLACK nodes. The actual amount of work performed by these procedures is more accurately represented by considering the number of nodes that are visited when an adjacency is being explored. Recall that we must find the neighbor, and if it is GRAY, then visit all adjacent neighbors of a smaller size. In the worst case, we are at level n-1, with a GRAY neighbor, and all adjacent neighbors at level Ø. In such a case, we must visit 2^n nodes. For example, consider Figure 5 where n=3 and we wish to visit the blocks adjacent to the block labeled A (i.e., blocks B, C, D, and E). We must visit the root of the quadtree as well as A's neighboring GRAY node and all of its NW and SW sons--i.e., a complete binary tree of height 2. In total,

$2^3 = 8$ nodes are visited. In general, let the space be partitioned into a 2^n by 2^n array. Assume a random image--i.e., a BLACK node is equally likely to appear in any position and level in a quadtree. Recalling the analogy drawn in Section 3 between phase one and the perimeter computation algorithm we have the following result:

Theorem 1: The average of the maximum number of nodes visited by LABEL_ADJACENT is n+1.

Proof: See Theorem 1 in [Samet2].

The speed of phase two of the algorithm depends on the method used for processing equivalence relations and on the number of pairs of equivalences and different objects of the set on which the equivalences are defined. We use a variant of an algorithm presented in [Knuth]. Its maximum execution time is proportional to the square of the number of pairs of equivalences. It is speeded up by using a modification due to D. McIlroy [Knuth] to have a maximum proportionality to the product of the number of pairs of equivalences and the log of the size of the set on which the equivalences are defined.

Recall that equivalence pairs are generated during phase one only when we are exploring the adjacencies of a node that is already labeled and its neighbor has also been labeled before, albeit with a different label. We now prove the following lemma:

Lemma 1: Phase one generates a maximum of one equivalence pair for each adjacency that is explored (i.e., each call to procedure LABEL explores two adjacencies).

Proof: There are two cases depending on the direction of the adjacency.

Case (a): An adjacency in the eastern direction can yield at most one equivalence pair regardless of the size of the neighbor. This is clearly true if the neighbor is larger (e.g., blocks 38 and 35 in Figure 1b). Similarly, if the neighbor is smaller, then only the northernmost such neighbor could have been previously labeled (e.g., blocks 12 and 20 in Figure 1b) because only it could have been the southern neighbor of a previously labeled node.

Case (b): An adjacency in the southern direction can only yield an equivalence pair if the neighbor is larger. No equivalence pair may result if the neighbor is smaller. This should be clear since southern neighbors could only have been visited if they are adjacent to a western neighbor which has been visited previously.

Q.E.D.

Letting B denote the number of BLACK nodes we have the following theorem:

Theorem 2: 2B log B is an upper bound on the execution time of phase two.

Proof: By the above lemma, phase one generates a maximum of one equivalence pair for each adjacency that is explored. Recall that phase one explores two adjacencies for each BLACK node. Also the set on which the equivalence pairs are defined has a maximum number of objects equal to the number of BLACK nodes, i.e., B. Thus when the equivalence merging algorithm of [Knuth] is used, one has 2B log B as the upper bound for the execution time.

Q.E.D.

Clearly, $B \leq 3 \cdot 2^{2n-2}$ since at most 3 of every 4 sons of a node at level 1 in a complete quadtree are BLACK. At this point we show how the upper bound of Theorem 2 may be tightened. Assume a 2^n by 2^n array and $n \geq 2$. We first prove the following lemma.

Lemma 2: 2^{2n-3} is an upper bound on the number of objects in the set upon which the equivalences are generated in phase one.

Proof: Recall our discussion with respect to the types of equivalences our algorithm is least proficient at handling. Clearly, the best algorithm is one that never generates pairs of equivalences. We mentioned the existence of a worst case configuration of nodes where an equivalence pair had to be generated (see Component 1 of Figure 4). Clearly, the upper bound of Theorem 1 arises when the quadtree corresponds to a single component--i.e., once all of the pairs of equivalences are merged, a single equivalence class results. Therefore, consider a case where the

minimal instance of the worst case case is replicated--e.g., the 2^4 by 2^4 array in Figure 6. In the general case, i.e., a 2^n by 2^n array, there are 2^{n-1} objects on which equivalence pairs are generated for the first four consecutive rows and $2^{n-1}-1$ such objects for each remaining set of four consecutive rows. Thus in a 2^n by 2^n array there are less than 2^{2n-3} objects in the set upon which equivalence pairs are generated (e.g., 1 through 29 in Figure 6).

Q.E.D.

We can now prove Theorem 2' as follows:

Theorem 2': 2B(2n-3) is an upper bound on the execution time of phase two.

Proof: From Lemma 2 we have that the maximum number of objects in the set upon which the equivalences are generated in phase one is less than 2^{2n-3}. From Theorem 2 and [Knuth] the execution time of phase two is bounded by the product of 2B and the log of the maximum number of objects in the set upon which the equivalence pairs are derived--i.e., $2B \log 2^{2n-3} = 2B(2n-3)$.

Q.E.D.

The speed of phase three of the algorithm can be obtained in a straightforward way. The tree must be traversed and for each BLACK node P, REGION(P) must be set to the head of the equivalence class obtained as a result of phase two. The actual lookup operation is bounded by the log of the maximum number of objects in

the set upon which the equivalences were generated in phase one. From Lemma 2 we have the value 2n-3. An upper bound on the size of the tree is obtained by the following lemma.

Lemma 3: The upper bound on the number of nodes of the quadtree is 4Bn+1.

Proof: See Lemma 1 in [Samet2]. Q.E.D.

We now have the following theorem:

Theorem 3: The upper bound of the execution time of phase three is proportional to B(2n-3) + 4Bn+1.

Proof: Use Lemmas 2 and 3 and the time required to access the head of an equivalence class.

Q.E.D.

Using Lemma 3 we obtain an upper bound on the average worst case execution time of phase one.

Theorem 4: The upper bound on the average worst case execution time of phase one is 6Bn+2B+1.

Proof: From Theorem 1 we have that for each adjacency involving a BLACK node, phase one results in an average worst case of n+1 nodes being visited. There are two adjacencies for each BLACK node. Also from Lemma 3 we have that the tree traversal component of phase one visits at most 4Bn+1 nodes. Therefore, we have 2B(n+1) + 4Bn+1 = 6Bn+2B+1 nodes being visited.

Q.E.D.

At this point we come to the main result:

Theorem 5: The average worst case execution time of phases one, two, and three has an upper bound proportional to the product of the number of BLACK nodes and the log of the diameter of the image.

Proof: The log of the diameter of the image is n. Summing up the contributions of phases one, two, and three as indicated by Theorems 4, 2', and 3, we have the result $6Bn+2B+1 + 3B(2n-3) + 4Bn+1 = 16Bn-7B+2$.

Q.E.D.

6. Concluding remarks

An algorithm has been presented for labeling the connected components of a binary image. The analysis was somewhat facilitated by the analogy with the perimeter computation algorithm represented by a quadtree. The algorithm's running time has been shown to have an average worst case time complexity proportional to the product of the log of the image's diameter and the number of the terminal nodes describing the area spanned by the components. This is of the same order of magnitude as the complexity of the perimeter computation algorithm in [Samet2] although it should be clear that the latter is smaller. Note that Theorems 2 and 2' yielded different upper bounds on the execution speed of phase two. We chose to use the result of Theorem 2' because it had a direct relationship to the log of the region's diameter.

In general, the performance of the algorithm is quite good because in actuality very few equivalence pairs are generated. Some variant of the worst case in terms of configurations leading to the generation of an equivalence pair will occur no matter which order of traversing the adjacencies is adopted. It should be clear that phase two can be combined with phase one by performing the merge dictated by the equivalence immediately in procedure ASSIGN_LABEL rather than using the list MERGES and executing phase two. We chose the previous approach in order to simplify the presentation of the analysis. Also note that Lemma 1

insures that the upper bound of the execution time of phase two is not affected by generating the same equivalence pairs more than once (e.g., in Figure 7 the equivalence A≡B is generated once by blocks 6 and 2 and once by blocks 9 and 2.

Note finally that if the algorithm is applied to both the BLACK and the WHITE nodes, one can compute the number of holes, and hence the genus, of the image. Genus can, of course, also be computed by counting the number of occurrences of various local patterns in the image.

FIG. 1. AN IMAGE, ITS MAXIMAL BLOCKS, AND THE CORRESPONDING QUADTREE. BLOCKS IN THE IMAGE ARE SHADED.

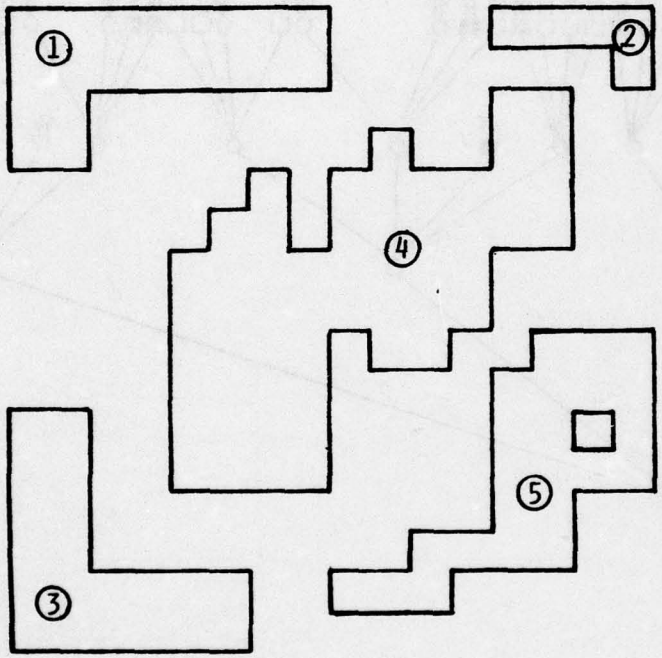

FIG. 1a. SAMPLE IMAGE.

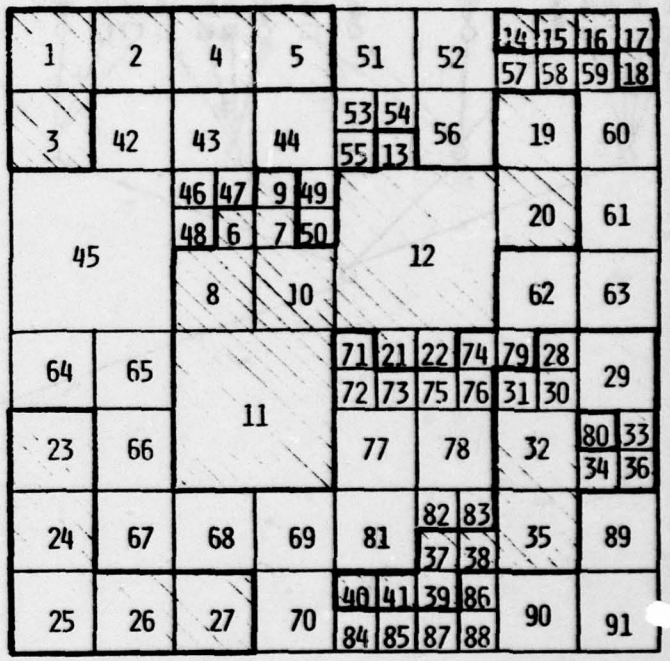

FIG. 1b. BLOCK DECOMPOSITION OF THE IMAGE IN (a).

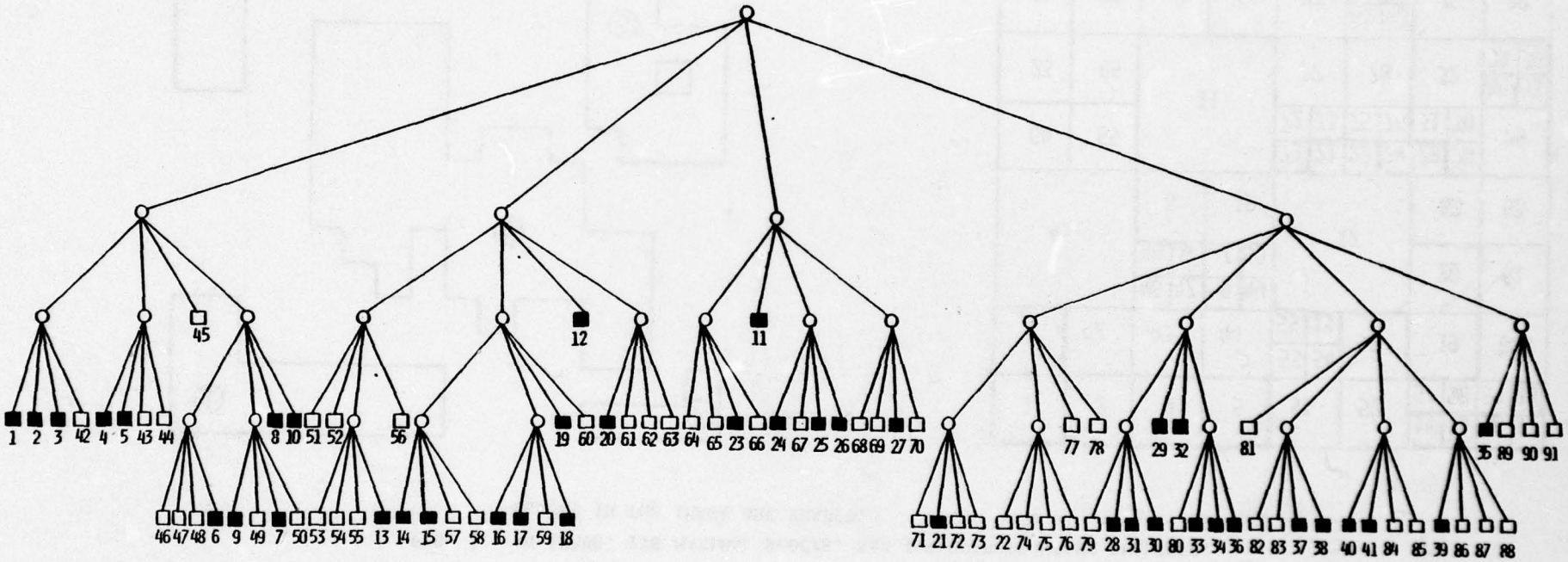

FIG. 1 c. QUADTREE REPRESENTATION OF THE BLOCKS IN (b).

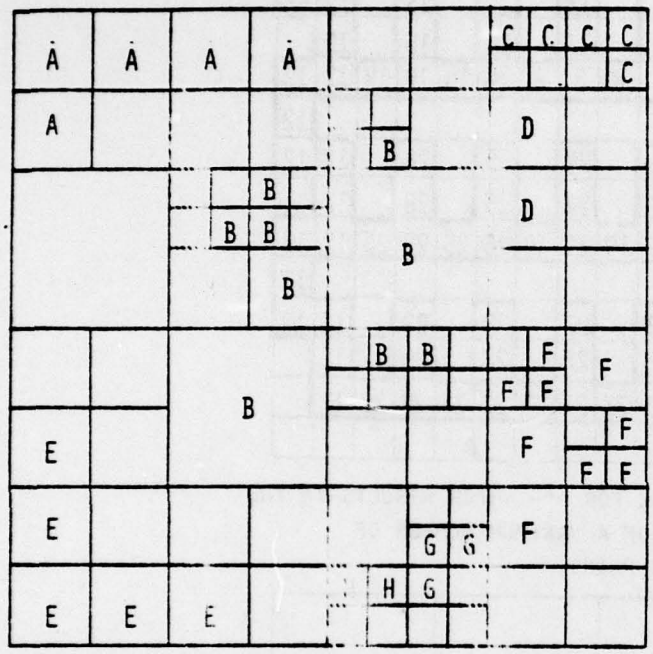

FIG. 3. RESULT OF THE APPLICATION OF PHASE ONE TO FIG. 1b.

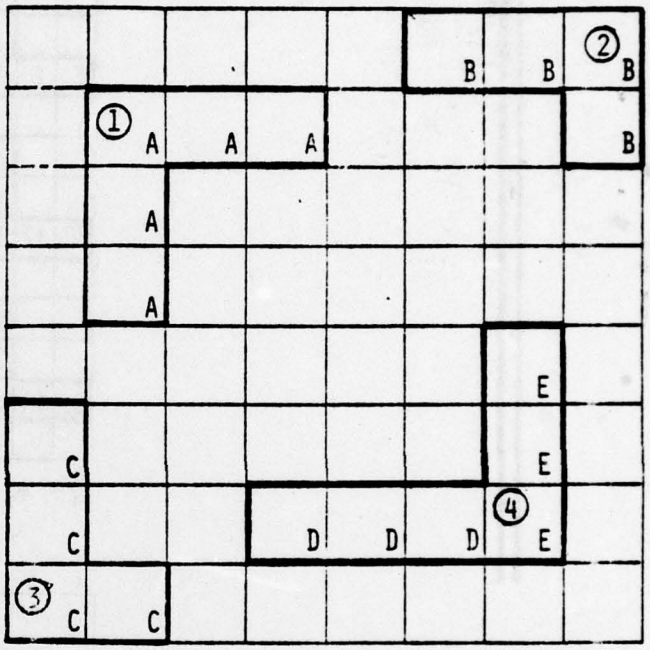

FIG. 4. SAMPLE COMPONENT SHAPES AND THE LABELS THAT THEY GENERATE.

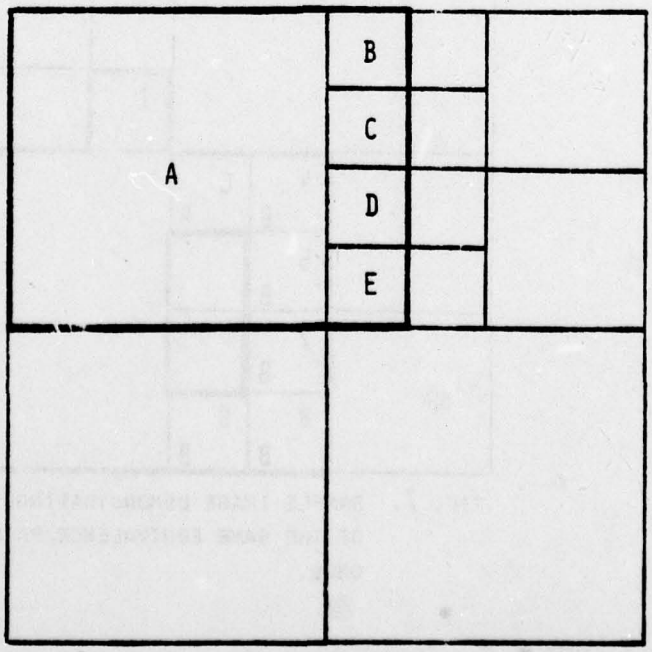

FIG. 5. SAMPLE IMAGE DEMONSTRATING THE MAXIMUM NUMBER OF NODES THAT MUST BE VISITED FOR AN ADJACENCY.

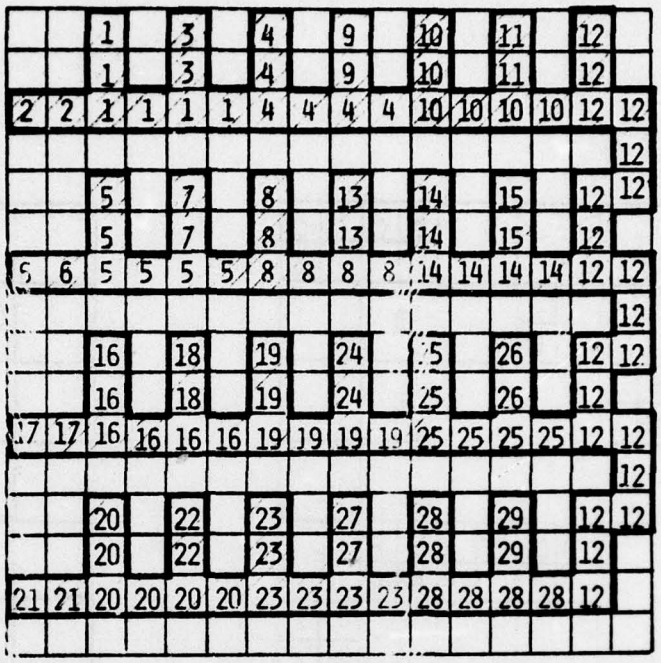

FIG. 6. SAMPLE IMAGE FOR n=5 WHICH RESULTS IN THE GENERATION OF A MAXIMUM NUMBER OF EQUIVALENCE PAIRS.

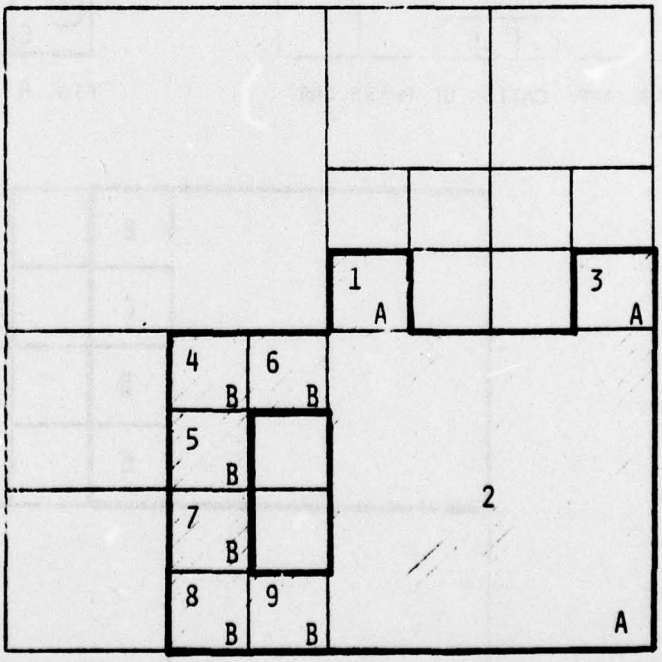

FIG. 7. SAMPLE IMAGE DEMONSTRATING THE GENERATION OF THE SAME EQUIVALENCE PAIR MORE THAN ONCE.

7. References

[DRS] C. R. Dyer, A. Rosenfeld, and H. Samet, Region representation: boundary codes from quadtrees, TR-732, Computer Science Center and Computer Science Department, University of Maryland, College Park, Maryland, February 1979.

[Klinger] A. Klinger and C. R. Dyer, Experiments in picture representation using regular decomposition, Computer Graphics and Image Processing 5, 1976, 68-105.

[Knuth] D. E. Knuth, The Art of Computer Programming, Vol. 1: Fundamental algorithms, second edition, Addison-Wesley, Reading, Mass., 1973, 353-355, 360, 572.

[Naur] P. Naur (Ed.), Revised report on the algorithmic language ALGOL60, Communications of the ACM 3, 1960, 299-314.

[RK] A. Rosenfeld and A. C. Kak, Digital Picture Processing, Academic Press, New York, 1976, Section 8.1.

[Samet1] H. Samet, Region representation: quadtrees from boundary codes, TR-741, Computer Science Department, University of Maryland, College Park, Maryland, March 1979.

[Samet2] H. Samet, Computing perimeters of images represented by quadtrees, TR-755, Computer Science Department, University of Maryland, College Park, Maryland, April 1979.

[Tarjan] R. E. Tarjan, On the efficiency of a good but not linear set union algorithm, TR 72-148, Computer Science Department, Cornell University, Ithaca, New York, November 1972.

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) CONNECTED COMPONENT LABELING USING QUADTREES		5. TYPE OF REPORT & PERIOD COVERED Technical
		6. PERFORMING ORG. REPORT NUMBER TR-756
7. AUTHOR(s) Hanan Samet		8. CONTRACT OR GRANT NUMBER(s) DAAG-53-76C-0138
9. PERFORMING ORGANIZATION NAME AND ADDRESS Computer Science Department University of Maryland College Park, MD 20742		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS U.S. Army Night Vision Laboratory Fort Belvoir, VA 22060		12. REPORT DATE April 1979 13. NUMBER OF PAGES 28
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Image processing
Pattern recognition
Region analysis
Connected components
Quadtrees

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

An algorithm is presented for labeling the connected components of an image represented by a quadtree. The algorithm proceeds by exploring all possible adjacencies for each node once and only once. Once this is done, any equivalences generated by the adjacency labeling phase are propagated. Analysis of the algorithm reveals that its worst case average execution time is bounded by a quantity proportional to the product of the log of the region's diameter and the number of blocks comprising the area connected by the components.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)