CONNECTED COMPONENT LABELING USING MODIFIED LINEAR QUADTREES

Xiaoning Wang and Wayne A. Davis

Department of Computing Science
The University of Alberta
Edmonton, Alberta, Canada T6G 2H1

Abstract[1]

Using the modified linear quadtree proposed in [1,9], this paper presents an $O(n \cdot N)$ algorithm for labeling connected components of a region consisting of N BLACK nodes in a 2^n by 2^n binary image. As a direct application of the algorithm, a method for computing the perimeter of a region is also described.

1. INTRODUCTION

The identification of all connected components of a region is a fundamental operation in image processing and geographic systems [4, 5]. Samet [6] presents an algorithm for labeling all connected components of a region represented by a quadtree, and shows that its average execution time is $O(T + N \cdot \log N)$, where T and N are the total number of nodes and the number of BLACK nodes in the quadtree, respectively. That algorithm outperforms the traditional method which has an execution time proportional to the number of pixels of the image [5]. Gargantini [3] also describes an entirely different algorithm using a linear quadtree [2]; however, that algorithm has limited power as it is only applicable to regions with very special configurations.

In this paper, an algorithm adopting a novel approach for labeling all connected components of a region using a Modified Linear Quadtree (MLQ) is presented. It is capable of handling regions with arbitrary configurations. Furthermore, the algorithm is of time complexity $O(n \cdot N)$, and hence compares favorably to Samet's algorithm [6]. As an application of the algorithm, this paper will show that, with the same time complexity, the perimeter of a region can also be computed.

2. DEFINITIONS AND NOTATION

This section contains some basic definitions and terminology for region representations that are fundamental for the remainder of this paper.

[1]This research was supported in part by the Natural Sciences and Engineering Research Council of Canada under Grant NSERC A7634.

Definition 1: An *image* is a 2^n by 2^n array of unit square pixels each of which can assume one of 2^k values, where n is called the *resolution parameter* of the image.

Definition 2: An image is called a *binary image* when its pixels assume either 1 or 0 values. A pixel is BLACK if it has the value of 1, otherwise it is WHITE.

Without loss of generality, only binary images will be considered in this paper since all of the algorithms can be extended to nonbinary images.

Definition 3: The *region* of a binary image is composed of all BLACK pixels, and the *background* of the region is composed of all WHITE pixels.

Definition 4: Let (i, j) represent the location of a pixel p in a given image, where i and j are the column and row positions respectively. Then p has four horizontal and vertical neighbors located at: (i-1, j), (i, j-1), (i, j+1) and (i+1, j). These pixels are called the *4-neighbors* of p, and are said to be *4-adjacent* to p.

Definition 5: For two BLACK pixels, p and q, of a region, p is said to be *connected* to q if there is a path from p to q consisting entirely of pixels of the region.

Definition 6: For any BLACK pixel p, the set of pixels connected to p is called a *connected component* of the region. If a region has only one component, then it is called "connected".

Based on the principle of recursive decomposition, an image is decomposed in the following manner to separate a region from its background[10]. If the region does not cover the entire binary array, the array will be subdivided into four equal-sized square blocks. This process will be applied recursively, until blocks are obtained that are either totally contained in the region or totally disjoint from it. The recursive decomposition of an image produces blocks that must have standard sizes (powers of 2) and positions. Similar definitions can now be formulated in terms of blocks.

Definition 7: A block is said to be BLACK if it contains only BLACK pixels, WHITE if it contains only WHITE pixels, and GREY if it contains both BLACK and WHITE pixels.

The four sides of a block are referred as to its North, East, South and West sides, or N, E, S and W for short. Let **OPSIDE**(T) be the side opposite to T, e.g., **OPSIDE**(E)=W.

Definition 8: Two blocks P and Q are said to be *4–adjacent* along the side T of P if the side T of P touches the side **OPSIDE**(T) of Q.

Definition 9: BLACK blocks P and Q are said to be *connected* if there exists a path consisting entirely of BLACK pixels from a pixel of P to a pixel of Q.

Definition 10: For two integers I and J given by

$$I = \sum_{i=0}^{n-1}(I_i \cdot 2^i), \text{ and } J = \sum_{i=0}^{n-1}(J_i \cdot 2^i), \text{ where } I_i, J_i \in \{0,1\},$$

$$\textbf{SHUFFLE}(I,J) = \sum_{i=0}^{n-1}(I_i \cdot 2 + J_i) \cdot 4^i.$$

To represent a block obtained by the recursive decomposition method requires the following definition:

Definition 11: The *key* of a block or node with 2^s by 2^s pixels is **SHUFFLE**(I, J), where (I, J) is the location of its left bottom pixel, and s is the *resolution parameter* of the block.

It is now easy to show that the two-tuple <K,s> uniquely represents a block, where K and s are the key and resolution parameter of the block, respectively.

A **modified linear quadtree** (MLQ) is defined to be a sequence of BLACK nodes in two-tuple form sorted in ascending key order. This differs from the usual definition of a linear quadtree in that the key of the node is stored as a single integer rather than as an n-digit quaternary code, and the resolution parameter of the node is given explicitly rather than implied by the number of don't care characters in the quaternary code. This modification results in space efficiency and improved execution time [9].

In presenting the connected component labeling algorithm, each BLACK node in the MLQ is stored as a record consisting of three fields. The first two fields, termed KEY and RES, contain the key and the resolution parameter of the node, respectively. The third field, termed ID, identifies the connected component containing the node. It is set as a result of the algorithm to be presented. An array **M** is used to represent the MLQ. Therefore, **M** has the property that for any

i, j = (1,2,...,N), if i < j then **M**[i]·KEY < **M**[j]·KEY.

The predicate **UNEXPLORED**(P,T) is true if and only if the side **T** of node **P** has not been marked "explored" in the progress of the algorithm. The predicate **LABEL**(**P**) is true if and only if **P**.ID has been assigned a value.

3. AN OBSERVATION

Given a node **P** in **M**, its four adjacent or neighboring nodes can be determined in $O(n)$ steps [1,9]. Suppose **Q** is the adjacent node to **P** in the west direction. The color of **Q** can be determined as WHITE, BLACK or GREY in $O(\log N)$ time [1,9]. The BLACK or WHITE color of **Q** provides the information regarding whether **Q** is connected to **P** or not. Very little knowledge, however, of what is happening between **P** and **Q** is known when the color of **Q** is GREY. Simply, this is because there can either be no BLACK node or as many as up to 2^s BLACK nodes in **Q** adjacent to **P**, where s= P·RES, i.e., **P** is a block of 2^s by 2^s pixels.

This implies that up to 2^s further searches on **M** must occur in order to exhaust all possible adjacencies. In fact, this is precisely what Samet's algorithm does. Assuming a random image, in the sense that a node is equally likely to appear in any position and at any level in the quadtree, the neighbor finding operation using a quadtree is so efficient that the average number of nodes visited is a constant [8]. Correspondingly, the neighbor finding operation using a linear quadtree is less efficient in that the average number of nodes visited is $O(\log N)$ [2]. Therefore, a connected component labeling algorithm using a linear quadtree cannot do the same thing as Samet's algorithm does.

Gargantini's algorithm [3] imposes a special configuration on the region to avoid performing an exhaustive search. As a result, the algorithm is not able to deal with regions with arbitrary configurations. Clearly it is a crucial step, in achieving an efficient method that when **Q**, the adjacent node of **P**, turns to be GREY, of how to preclude further searching on **M** without losing any information regarding the adjacencies.

It is this observation that leads to a new method, to be described in the next section, for labeling all connected components of a region using an MLQ.

4. AN INFORMAL DESCRIPTION

The connected component labeling algorithm has three phases. An array called MAP will be used mainly by the first phase. MAP is constructed from **M** such that for any two integers, i, j = (1,2,...,N), if i < j then **M**[MAP[i]]·RES ≤ **M**[MAP[j]]·RES. In essence, the use of MAP provides the visit of the nodes in **M** in ascending size order, while traversing **M**.

The first phase explores all possible adjacencies between any pair of BLACK nodes in **M** and generates

equivalence pairs. The second phase merges all the equivalence pairs generated during phase one into equivalence classes. Finally, the third phase assigns the same identifier (i.e., the label) to those BLACK nodes that belong to the same equivalence class to reflect a connected component.

In particular, phase one traverses **M** in ascending size order. For each BLACK node **P** in **M** being visited, and T in {N,E,S,W}, if the side T of **P** has not been previously marked, then the adjacency between node **P** and the BLACK node of greater or equal size along the side T of **P** needs be explored. If such a BLACK node indeed exists in **M**, say **Q**, then the side **OPSIDE**(T) of **Q** is marked "explored" and is assigned the same label as that of **P** to indicate that both **P** and **Q** belong to the same component. Depending on the configuration of the region under consideration, **Q** may already have been assigned a label different from that of **P**, in which case, an equivalence pair consisting of the two labels is generated. This equivalence pair will be used in the later stages of the algorithm to update the labels of **P** and **Q** so that eventually they will be assigned the same label. If the side T of **P** has already been marked "explored", then the exploration of the adjacency to the side T of **P** is no longer needed.

The consequence of this technique is not only to save one search on **M**, but rather to save the necessity of exhausting all possible adjacencies along the side T of **P**. The reason for this is as follows. The side T of **P** can be marked "explored" only at the time when that side of **P** was found to be connected to a BLACK node that was being visited by the algorithm. The size of this BLACK node cannot be bigger than **P** for otherwise it would not be visited before **P**. As a matter of fact, there could be as many such BLACK nodes as the size of **P** in **M**. Regardless how many BLACK nodes of this nature exist, the "explored" status of the side T of **P**, while **P** is being visited, simply indicates that the exploration of the adjacencies across the side T has been previously done.

The distinct feature of this algorithm is that phase one guarantees that, at most, one exploration of an adjacency along each side of every BLACK node in **M** is sufficient to discover all possible adjacencies between any pair of BLACK nodes. To see this, remember that phase one visits the nodes in **M** in ascending size order. Consider, for example, the image in Fig. 1, where the resolution parameter n is 3. By the time BLACK node **A** is visited, its eastern adjacency needs not be re-explored, since BLACK nodes **E**, **D**, **C** and **B** have already been visited before **A**, and the adjacencies were discovered at that time. Now, however, its northern adjacency must be explored, since that side of **A** cannot be marked "explored" although **F** was visited before **A**. As A's northern neighbor of equal size is found to be GREY, the algorithm immediately concludes that there does not exist a BLACK node adjacent to the northern side of **A**, for otherwise the northern side of **A** would have been marked "explored". Therefore, no further search is necessary.

Phase two will merge the equivalence pairs generated during phase one into equivalence classes in such a way that each equivalence class contains all labels assigned to those BLACK nodes that form a connected component.

Finally, phase three updates the labels assigned to the BLACK nodes during phase one using the equivalence classes generated by phase two. Upon completion of phase three, all BLACK nodes of each connected component will have unique labels.

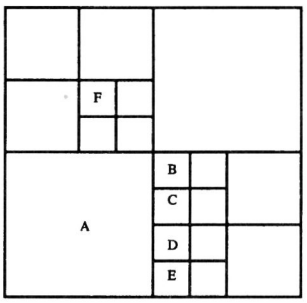

Fig. 1. An Adjacency Configuration.

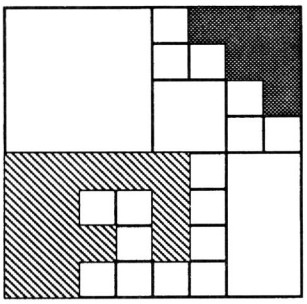

Fig. 2. A Region With 2 Components.

5. THE FORMAL ALGORITHM

The connected component labeling algorithm will now be specified by the following procedures. Actually, only those procedures that correspond to phases one and three will be presented. Phase two can be achieved by using the well known **UNION-FIND** algorithm [11]. The main procedure is named **LABEL-CC**, and invoked with an array **M** and an integer N corresponding to the number of BLACK nodes in **M**. Steps 1 and 2 construct the MAP and initialize a list called E-list which will contain the equivalence pairs as they are generated. Procedure **PROPAGATE** implements phase one. It visits the nodes in **M** in ascending size order through MAP, explores the adjacencies between pairs of BLACK nodes by invoking **EXPLORE**, assigns labels produced by **ID-GENERATOR**, and accumulates equivalence pairs in the E-list. Procedure **EQ-NEIGHBOR** used by **EXPLORE** computes the key of **M**[j]'s equal-sized neighbor in the direction specified by the parameter side. The unspecified procedure **SEARCH**(**M**, **P**) works as follows: if **P** is a BLACK node then **SEARCH** returns an integer value k such that **P** is either equal to or contained in **M**[k]. However, if **P** is WHITE or GREY then **SEARCH** simply returns a zero. Unique labels are generated by procedure **ID-GENERATOR**, and assigned to BLACK nodes by procedure **ASSIGN-**

LABEL. Procedure **UPDATE** implements phase three by uniquely labeling each component while scanning **M**.

```
Procedure LABEL-CC(M, N)
  begin
  1 construct MAP;
  2 E-list:={ϕ};
  3 PROPAGATE(M, N);
  4 generate equivalence classes from E-list;
  5 UPDATE(M, N);
  end;

Procedure PROPAGATE(M, N)
  begin
    for i:=1 to N do
      begin
        j:=MAP[i];
        for side in {N,E,S,W} do
          if UNEXPLORED(M[j], side)
            then EXPLORE(M[j], side);
        if not LABEL(M[j]) then
          M[j].ID:= ID-GENERATOR;
      end;
  end;

Procedure EXPLORE(M, j, side)
  begin
    neighbor:= EQ-NEIGHBOR(M[j], side);
    k:= SEARCH(M, neighbor);
    if k > 0 then
      begin
        mark OPSIDE(side) of M[k] "explored";
        ASSIGN-LABEL(M[j], M[k]);
      end;
  end;

Procedure ASSIGN-LABEL(node,adj)
  begin
   if LABEL(node) and LABEL(adj)
    then if node.ID ≠ adj.ID
     then add (node.ID,adj.ID) to E-list;
    else if LABEL(node)
     then adj·ID:=node.ID
     else if LABEL(adj)
      then node.ID:=adj.ID
      else node.ID:=adj.ID:= ID-GENERATOR;
  end;

  Procedure UPDATE(M, N)
    begin
      for i:=1 to N do
        M[i].ID:= FIND(M[i]);
    end;
```

Example: As an example of the application of the algorithm, consider the region given in Fig. 2 whose block decomposition is given in Fig. 3. The BLACK nodes have been numbered in the order in which they were visited by phase one. Thus node 1 has been visited before nodes 2, 3, etc. The labels assigned to the two components by the first phase of the algorithm are shown in Fig. 4. A short explanation about Fig. 4 is necessary at this point. When node 7 is visited, neither node 7 nor node 11, its eastern neighbor, has been labeled yet, thus label d is generated and assigned to both. When node 8 is visited, it has no label, but its northern neighbor, node 11, has already been assigned the label d, and thus node 8 is assigned the label d as well.

Fig. 4 illustrates the status of the image at the conclusion of the first phase of the algorithm. It has four different labels: a, b, c and d, with a equivalent to b, and b equivalent to c. The equivalence pair (a, b) was generated when node 9 was visited and its northern adjacency was explored. In essence, node 9 was labeled with a when node 1's western adjacency was explored, whereas node10 was labeled with b when node 2's western adjacency was explored. Similarly, the equivalence pair (b,c) was generated when node 5 was visited.

Applying the second phase of the algorithm to the generated equivalence pairs results in the following two equivalence classes: {a,b,c} and {d}.

Fig. 5 shows the labels updated by the third phase of the algorithm.

Theorem 1: The time complexity of the connected component labeling algorithm is $O(n \cdot N)$.

Proof: Constructing the MAP requires time $O(N \cdot \log N)$. Phase one calls procedure **EXPLORE** N times, and procedure **EXPLORE** requires time $O(n+\log N)$, where n and log N originates from the invoking of procedure **EQ–NEIGHBOR** and **SEARCH**, respectively. Therefore phase one takes time $O(n \cdot N + N \cdot \log N)$. Phase two requires time $O(N \cdot \log N)$ [9]. Phase three requires time $O(N)$. Since $\log N < 2n$, the time complexity of the algorithm is therefore $O(n \cdot N)$.

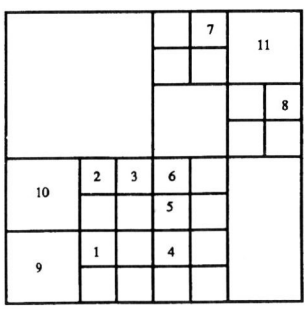

Fig. 3. Decomposition of Fig. 2.

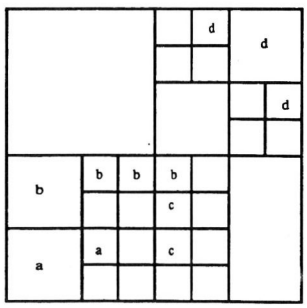

Fig. 4. Results of Phase 1.

6. COMPUTING THE PERIMETER

Perimeter computation is another basic operation in image processing. Algorithms computing the perimeter of a region in a binary image represented

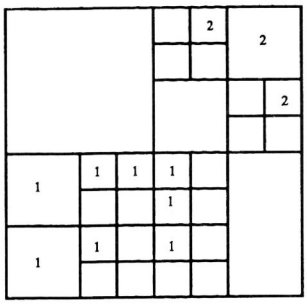

Fig. 5. Labels Resulting From Phase 5.

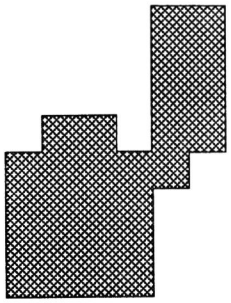

Fig. 6. A Connected Region.

either by an array or by a chain code are contained in [5]. An algorithm for computing the perimeter of a region encoded as a quadtree has also been developed by Samet [7].

The following perimeter computation algorithm traverses the MLQ in ascending size order. For each node **P** in the MLQ being visited, the length of each of its four sides is first included in the value of the perimeter. Then the neighbor nodes of **P** which have not been previously visited need to be considered. For each adjacent node **Q** that is BLACK, twice the length of the common side is deducted from the value of the perimeter. This reflects the fact that the segment between **P** and **Q** does not belong to the boundary of the region. The factor 2 occurs because the adjacency between two BLACK nodes is explored once and only once due to the traversal strategy used.

For example, given the BLACK node **D** in Fig. 7, the common segment between **D** and its southern neighbor A is explored by the time **D** is visited, but the same common segment is not considered when **A** is visited. Therefore the length of this segment **DA** has to be deducted in advance when **D** is visited.

The following procedure **PERIMETER** specifies the algorithm.

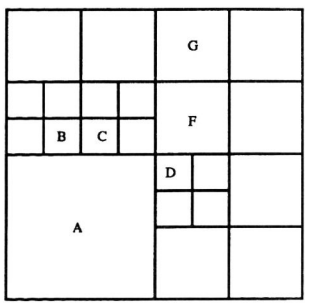

Fig. 7. Decomposition of Fig. 6.

```
Procedure PERIMETER(M, N)
 begin
  construct MAP;
  perimeter:=0;
  for i:=1 to N
   begin
    j:=MAP[i];
    segment:=2** M[j].RES;
    perimeter:= perimeter + 4 * segment;
    for side in {N, E, S, W} do
     if UNEXPLORED( M[j], side) then
      begin
       neighbor:= EQ-NEIGHBOR(M[j], side);
       k:= SEARCH(M,neighbor);
       if k > 0 then
       begin
        perimeter:=perimeter - 2 * segment;
        mark OPSIDE(side) of M[k] "explored";
       end;
      end;
   end;
  return(perimeter);
 end;
```

The key to this algorithm is that each node in the MLQ is visited once and, at most, its four neighbors need be explored. Such an advantage is achieved by traversing the MLQ in ascending size order. Otherwise, in the worst case, when the node being visited is of size 2^{n-1} by 2^{n-1}, 2^{n-1} nodes need be searched as in Samet's algorithm [7].

Example: Consider the region given in Fig. 6. The corresponding block decomposition is shown in Fig. 7. The MLQ contains six BLACK nodes representing blocks **A, B, C, D, F** and **G**. Assuming n=3, the perimeter is 30. Procedure **PERIMETER** visits the BLACK nodes in the order: **B, C, D, F, G** and **A**.

The following table contains a step-by-step trace through the algorithm for this example. The symbols 'ϕ' and '-' stand for don't care and non-existance, respectively.

Theorem 2: The time complexity of the algorithm **PERIMETER** is $O(n \cdot N)$.

Proof: Similar to the proof of Theorem 1.

Note that if the region is not connected, i.e., it contains more than one connected component, then the algorithm will return the sum of the perimeters of each connected component. It is, however, not difficult to compute the perimeter of every connected component of the region simultaneously in the same time complexity with a minor modification of the algorithm, provided that all connected components have been labeled.

node	side	neighbor	segment	contribut.	perim.
B				4	4
	N	-			4
	E	C	BC	-2	2
	S	A	BA	-2	0
	W	-			0
C				4	4
	N	-			4
	E	-			4
	S	A	CA	-2	2
	W	ϕ			2
D				4	6
	N	F	DF	-2	4
	E	-			4
	S	-			4
	W	A	DA	-2	2
F				8	10
	N	G	FG	-4	6
	E	-			6
	S	ϕ			6
	W	-			6
G				8	14
	N	-			14
	E	-			14
	S	ϕ			14
	W	-			14
A				16	30
	N	ϕ			30
	E	ϕ			30
	S	-			30
	W	-			30

7. CONCLUSION

Techniques for labeling connected components and computing the perimeter of a region have been described. The algorithm for labeling connected components is superior to the one using a standard linear quadtree [3] in the sense that it is capable of handling regions with arbitrary configurations. By the same token, the perimeter computation algorithm shares the same advantage.

REFERENCES

1. Davis, W.A., and Wang, X., "A New Approach to Linear Quadtrees", *Proceedings Graphics Interface '85*, pp. 195-202, Montreal, May 1985.
2. Gargantini, I., "An efficient way to represent properties of quadtrees", *Comm. ACM*, Vol. 25, pp. 905-910, Dec. 1982.
3. Gargantini, I., "Detection of Connectivity for Regions Using Linear Quadtrees", *Comp. & Math. with Appl.*, Vol. 8, pp. 319-327, 1982.
4. Rosenfeld, A., "Connectivity in Digital Pictures", *J.ACM*, Vol. 17, pp. 146-160, 1970.
5. Rosenfeld, A. and Kak, A.C., "*Digital Picture Processing*", Academic Press, New York, 1976.
6. Samet, H., "Connected Component Labeling Using Quadtrees", *J.ACM*, Vol. 28, pp. 487-501, 1981.
7. Samet, H., "Computing Perimeters of Regions in Images Represented by Quadtrees", *IEEE Trans. Pattern Analy. & Mach. Intel.*, Vol. PAMI-3, pp. 683-687, 1981.
8. Samet, H., "Neighbor Finding Techniques for Images Represented by Quadtrees", *Comput. Graphics and Image Process.*, Vol. 18, pp. 37-57, 1982.
9. Wang, X., "Some New Approaches for Linear Quadtrees", M.Sc. Thesis, Department of Computing Science, University of Alberta, 1985.
10. Klinger, A. and Dyer, C.R., "Experiments in Picture Representation Using Regular Decomposition", *Computer Graphics and Image Processing*, Vol. 5, pp. 68-105, 1976.
11. Aho, A., Hopcroft, J. and Ullman, J.D., "*The Design and Analysis of Computer Algorithms*", Addison-Wesley, Reading, Mass., 1974.