Fast Connected Component Labeling Algorithm Using A Divide and Conquer Technique

by

Jung-Me Park[†], Carl G. Looney[‡], Hui-Chuan Chen[†]
Computer Science Dept. Computer Science Dept.
University of Alabama, Tuscaloosa[†] University of Nevada, Reno[‡]
Tuscaloosa, AL35487 Reno, NV 89557
jpark@cs.ua.edu, chen@cs.ua.edu, looney@cs.unr.edu

Abstract. *We investigate a method to speed up the $O(n^3)$ labeling algorithm of Rosenfeld and Pfaltz for segmenting binary images, which is unduly complex for large images. That algorithm searches line-by-line, top to bottom, to assign a blob label to each current pixel that is connected to a blob. A large number K of labels arises of which many are equivalent, so the equivalence must be resolved. This requires a KxK matrix to represent the connectivity and $O(K^3)$ operations for resolution, which is very large for large images. Our approach partitions the binary image into NxN rectangles and perform local equivalence resolution on each while keeping track of the global equivalence with list pointers to equivalence lists. Such divide and conquer technique greatly increases the run time speed.*

Keywords: binary images, connectivity, labeling algorithm, equivalence resolution, divide and conquer.

---

1. Introduction

Detection of connected components between pixels in binary images is a fundamental step in segmentation of an image objects and regions, or *blobs*. Each blob is assigned a unique label to separate it from other blobs. All the pixels within a blob of spatially connected 1's are assigned the same label. It can be used to establish boundaries of objects, components of regions, and to count the number of blobs in an image [1]. Its applications can be found in automatic inspection, optical character recognition, robotic vision, etc. [2].

The original algorithm was developed by Rosenfeld and Pfaltz [3] in 1966. It performs two passes through the image. In the first pass, the image is processed from left to right and top to bottom to generate labels for each pixel and all of the equivalent labels are stored in a pair of arrays. In the second pass, each label is replaced by the label assigned to its equivalence class. Several papers [4,5,6] pointed out the problems in the second pass for large images because the equivalence arrays can become unacceptably large [4]. The way in which label equivalences are resolved can have a dramatic effect upon the running time of this algorithm.

Modifications include one proposed by Haralick that does not use an equivalence array (see [4]) and a small equivalence table by Lumia, Shapiro, and Zuniga [4] that is reinitialized for each line. The latter paper makes comparison runs between these three algorithms. Another solution uses a bracket table [7] to associate equivalent groups. Its pushdown stack data structure is implemented in hardware. Our approach computes the connected components of a binary image in real time without any special hardware support. Instead it applies the power and efficiency of the divide-and-conquer technique. This new method can compute connectivity in a 1769*1168 image in about 2, rather than hundreds, of seconds.

2. Connected Components

2. 1 Basic Pixel-Connectivity.

A pixel p at coordinate (x, y) has four direct neighbors, $N_4(p)$ and four diagonal neighbors, $N_D(P)$. Eight-neighbors, $N_8(p)$ of pixel p consist of the union of $N_4(p)$ and $N_D(P)$ (see [1] for a basic description).

To establish connectivity between pixels of 1s in a binary image, three type of connectivity for pixels p and q can be considered: i) 4-connectivity – connected if q is in $N_4(P)$; ii) 8-connectivity – connected if q is in $N_8(p)$; iii) m-connectivity- connected if q is in $N_4(P)$, or if q is in $N_D(P)$ and $N_4(p) \cap N_4(q) = \varnothing$;

2.2 A Connected Component Labeling Algorithm.

The labeling algorithm is described below based on 8-connectivity.

Step 1: Initial labeling. Scan the image pixel by pixel from left to right and top to bottom. Let p denote the current pixel in the scanning process and 4-nbr denote four neighbor pixels in N, NW, NE and W direction of p. If p is 0, move on to the next scanning position. If p is 1 and all values in 4-nbrs are 0, assign a new label to p. If only one value in 4-nbrs is not 0, assign its values to p. If two or more values in 4-nbrs are not 0, assign one of the labels to p and mark labels in 4-nbrs as equivalent.

Step 2: Resolve equivalences (This is developed in the following paragraphs).

The equivalent relations are expressed as a binary matrix. For example, if label 1 is equivalent to 2, label 3 is equivalent to 4, label 4 is equivalent to 5, and label 1 is equivalent to 6 then the matrix L is that shown in Figure 1 a). Equivalence relations satisfy reflexivity, symmetry and transitive [1]. To add reflexivity in matrix L, all main diagonals are set to 1. To obtain transitive closure the Floyd-Warshall (F-W) algorithm [1] is used.

```
for j  = 1  to  n
 for i = 1 to n
     if   L[i,j] = 1 then
             for k = 1 to n
                  L[i,k] = L[i, k] OR L[j,k];
```

After applying reflexivity and the F-W algorithm, the matrix L is that shown in 1 b). This algorithm can be performed in $O(n^3)$ ***OR*** operations. After calculating the transitive closure, each label value is recalculated to resolve equivalences. The image is scanned again and each label is replaced by the label assigned to its equivalence class.

a)

	1	2	3	4	5	6
1		1				1
2	1					
3				1		
4			1		1	
5				1		
6	1					

b)

	1	2	3	4	5	6
1	1	1				1
2	1	1				1
3			1	1	1	
4			1	1	1	
5			1	1	1	
6	1	1				1

Figure 1. Equivalence relations in terms of binary matrix. a) Matrix before applying the F-W algorithm. b) Matrix after applying reflexivity and the F-W algorithm.

3. A Fast Connected Component Labeling Algorithm

The main idea in this algorithm is to divide the image into NxM small regions (we use NxN here for simplicity). The large equivalence array is the main bottleneck in the original algorithm, but NxN small equivalence arrays can be found in greatly reduced time. Figure 2 shows that an image divided into 3x3 small regions for labeling independently as described in Section 2.2. Then we connect each region with its neighbor regions to generate the actual label within the entire image. We use NxN pointers Label_List[i] to point to arrays that maintain the *global labels* with respect to the entire image. Label_List[i] points to the array for Region [i] where each array element is the global label within the entire image and the index for each array element is the local label within Region[i]. Memory allocation for each array pointed to by Label_List[i] can be done dynamically according to the maximum local label in Region[i]. Figure 3 depicts these lists. The example of Figure 4 shows that local label 1, 2 and 6 are equivalent and their global label within the entire image is 8; local label 3, 4, and 5 are equivalent and their global label is 9. The Total_Index equals 7 at the end of Region[i-1], which is kept in the list at index 0.

Our *fast labeling algorithm* (based on 8-connectivity) is described below. The other connectivity differs only in its neighboring checking

The *Fast Labeling Algorithm*.

Step 1: Divide the given image into NxN small regions and set Total_Index = 0

Step 2: For each region i = 1 to NxN
- i) apply ***Step 1*** of the original algorithm in Section 2.2;
- ii) allocate memory for the array pointed to by Label_List[i] as maximum no. of labels for Region[i];
- iii) use F-W algorithm in Section 2.2 to resolve the equivalences within Region[i].
- iv) for j=1 to size of an array for Region[i] do
  Label_List[i][j] = Total_Index + lbl
  // lbl is a label to its equivalence class after equiv. resolution (see Figure 4).
- v) Total_index = Total_index + maximum{lbl}
- vi) if (i > 1) then call *Merge(i)*;
  // to update labels in bordering area between regions

Step3: For each region i = 1 to NxN do
scan image in Region[i] from left to right, top to bottom and replace all local label value k with Label_List[i][k];

The Merge(i) Function (resolve equivalences of pixels in bordering area between regions).

Step 1: select first pixel p in Region[i];
If (label (p) >0) then
for each pixel q in N_8 (p) intersects other regions
// see figure 5 a)
if(label(q) > 0) then
call Resolve_Equivalence(p,q,i);

Step 2: for each pixel p in the first column in Region[i]
if (label (p) > 0) then
for each pixel q in N_8 (p) intersects Region [i-1]
// see Figure 5 –b)
if (label (q) is > 0) then
call Resolve_Equivalence(p,q,i);

Step 3: for each pixel p in the first row in Region[i]
if label(p) > 0 then
for each pixel q in N_8 (p) intersects Region [i-N]
// see Figure 5-c)
if (label (q) is > 0) then
call Resolve_Equivalence(p,q,i);

The Resolve_Equivalence(p,q,i) Function.

Step1: Index1 = Label_List[region no. of q][label(q)];
Index2 = Label_list[i][label(p)] ;
if(Index1 not equal to Index2) then
do Step 2.

Step2: Small_Lbl = min{index1,index2};
Large_Lbl = max{index1, index2};
for k=1 to i do
for j=1 to size of an array for Region[k].
if (Label_List[k][j] > Larege_Lbl) then
Label_List[k][j] = Label_List[k][j] –1;
else if (Label_List[k][j] = Large_Lbl) then
Label_List[k][j] = Small_Lbl;
Total_Index = Total_Index –1;

4. Experiment Results

We tested the new algorithm with image size of 2008K (e.g., 1760*1168 pixels). The results of the processing time with the different sizes of N are listed in Table1. Figure 6 shows the CPU time as a function of the different sizes for N. There is no division in the image when N=1 and the algorithm is the same as the original one developed by Rosenfeld and Pfaltz. With a Pentium 500 MHZ, 128 MB RAM computer, it is not feasible to calculate the connected component in real time when N=1, 2 or 3. Our new algorithm computes connected components within 2 seconds for an image size of 2008k with N = 25. Table 2 shows the comparisons with other algorithms with different size images. According to Lumia, Shapiro, and Zuniga[4] an image size of 973K can be calculated in 78 seconds with their method. With the fast connected component labeling algorithm the image size of 973k can be calculated within 0.82 seconds.

Region[1]	Region[2]	Region[3]
Region[4]	Region[5]	Region[6]
Region[6]	Region[7]	Region[8]

Figure 2. Division of original image into 3*3 regions.

The next step is to determine how we can choose the proper size of N? Our experimental results suggest that N is optimal when the regions have 30*30 to 60*60 pixels. Small size of sub-images always guarantees small size of equivalence arrays. The *fast connected components algorithm* presented in this paper is superior to any other serial algorithms that we have found and makes the computations without any special hardware support.

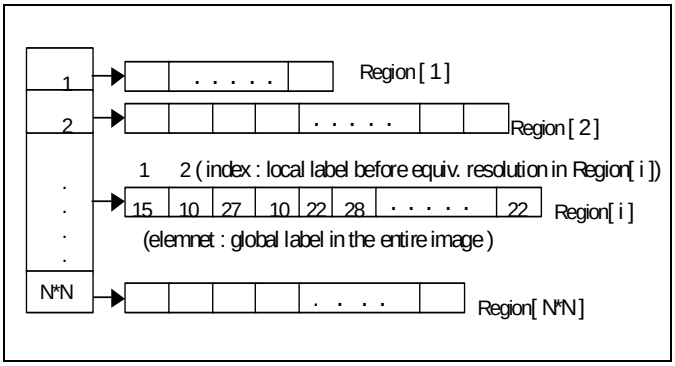

Figure3. The Label_List structure.

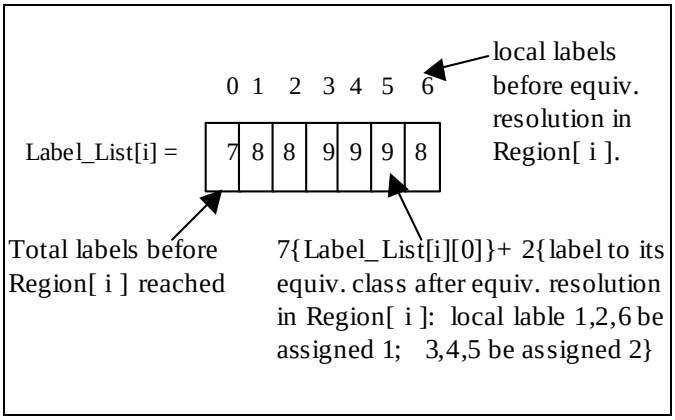

Figure 4. The Label_List[i] example.

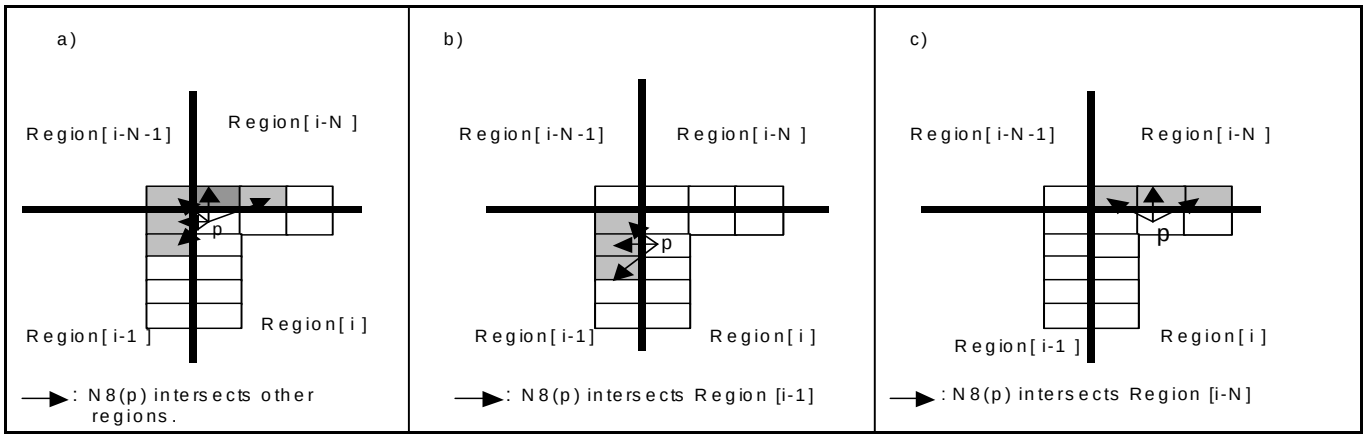

Figure 5. Merge taken place in three ways at Region[i]. a) Merge at the first pixel in Region[i] b) Merge at pixels in the first column in Region[i] and the last column in region[i-1] c) Merge at pixels in the first row in Region[i] and the last row in Region[i-N].

Table 1. CPU time for the image size 1760x1168.

N	4	5	10	15	20	25
CPU (secs.)	274.57	160.22	12.47	4.01	2.75	2.47

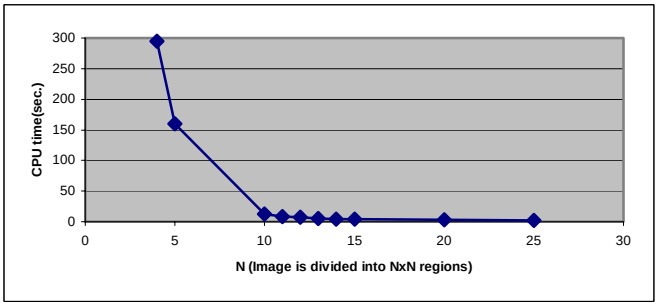

Figure 6. CPU time as a function of N.

Table 2. Comparison with other algorithms

	CPU time(sec)			
Image size	212K	321K	973k	2008k
Rosenfield[3]	1.32	113.64	------	-----
Harlick[4]*	0.55	3.24	76.57	331.75
Lumia et al.[4] *	0.22	3.79	78.43	358.39
Fast labeling	0.05 (N=10)	0.11 (N=10)	0.82 (N=20)	2.47 (N=25)

*Labels from these methods are not consecutive numbers, processing time for this step is included for fair comparison.

5. References

[1] Gonzalez, R. C., Woods, R.E., *Digital Image Processing*, Addison Wesley, 1992.

[2] Ronsen, C., Denijver, P.A., "Connected components in Binary Images:The Detection Problem," *Research Studies Press*, 1984.

[3] Rosenfeld, A., Pfaltz, J.L., "Sequential Operations in digital Processing," *JACM*, 13, 471-494,1966.

[4] Lumia, R., Shapiro, L., Zuniga, O., "A New Connected Components Algorithm for Virtual Memory Computers," *Computer Vision, Graphics, and Image Processing*, 22,1983, 287-300.

[5] Lumia R., "A New Three-dimensional connected components Algorithm," *Computer Vision, Graphics, and Image Processing*, 23, 1983, 207-217.

[6] Manohar, M, Ramapriyan, H.K., "Connected Component Labeling of Binary Image on a Mesh connected Massively Parallel Processor," *Vision, Computer Graphics and Image Processing*, 45, 133-149, 1989.

[7] Yang, X. D., "An Improved Algorithm for Labeling Connected Components in a Binary Image," *TR 89-981*, march, 1989.