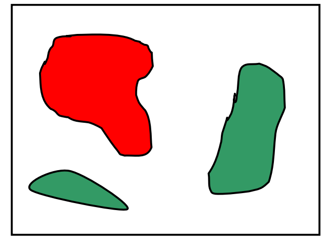
Finding the connected components in an image



A connected component is a set of connected pixels that share a specific property, *V*. Two pixels, *p* and *q*, are connected if there is a path from *p* to *q* of pixels with property *V*. A path is an ordered sequence of pixels such that any two adjacent pixels in the sequence are neighbors. An example of an image with a connected component is shown at the right. All of the pixels in the red object are connected. There are separate green objects. It is possible to find a path between any two pixels within any of the objects, but not between pixels in different objects.

We want to create an algorithm that can find the connected components in an image. We will assume that all of the pixels in the image have been labeled with values that correspond to the property of interest. In the above image there would be three values for *V*, say *V*=0 for the white background, *V*=1 for red pixels and *V*=2 for green pixels. It must be able to determine, for example, that the two green objects in the above picture are distinct.

Constructing a connected component consists of growing sets of pixels that are connected and have the same value of a property. This could be accomplished by first finding a pixel with a given property value, then looking at all its neighbors, labeling each that has the same value as being in the same component, and so on. This leads to a somewhat random stepping through the image, and is somewhat inefficient. Scanning the image in a specified order can develop a more systematic and efficient algorithm.

Image scanning and labeling

Let us first introduce some notation. The image will be represented by an array *A* that has *N* columns and *M* rows. *A*[*x*,*y*] refers to the element in column *x* and row *y*, with $x \in \{0, N-1\},\ \ y \in \{0, M-1\}$. We will assume that each pixel has a value

Let *Q* be an array that is the same size as *A*. We will use this array to hold the connected component labels.

Let *L* be the label index. We start with *L*=0 and increment *L* whenever we want to create a new connected component label. The goal is to end up with all of the pixels in each connected component having the same label and all of the distinct connected components having different labels.

In the course of running the algorithm, it is possible that some of the pixels in the same connected component will end up with different labels. These different label values will be discovered and resolved at the end of the labeling process. This will be clear as you follow the description below. We will let EQ be a vector that holds the equivalence class relations that are discovered as the algorithm is running.

We assume that all of the pixels have a value of the property *V* and that we are interested in those pixels with nonzero values. Those with value zero are assumed to be the image background. Setting the pixel values is done before starting this algorithm. The following algorithm will be described for 4-connected neighborhoods.

Step 1: Label pixel $A[0,0]$. If $A[0,0]>0$ then increment L and set $Q[0,0]=L$. This takes care of the first pixel in the image.

Step 2: Label the pixels in row $y=0$. For $x=1$ to $N-1$, check the value of $A[x,0]$. If $A[x,0]>0$ and $A[x,0]=A[x-1,0]$ then set $Q[x,0]=Q[x-1,0]$. If $A[x,0]>0$ and $A[x,0]\neq A[x-1,0]$ then increment L and set $Q[x,0]=L$. This will cause neighboring pixels in the first row that have the same value to have the same label. However, pixels in the first row that have the same value but are not connected will have different labels. This is shown in the chart below, where there are pixels colored red, green and blue. The labels are recorded in the cells.

0	0	1	1	0	0	2	2	2	3	3	0	4	4	0	5	5	5	0

Step 3: Label the rest of the rows. The first element of each row is handled a little differently than the rest of the elements, since it has no left neighbor. For $y=1$ to $M-1$ do the following. If $A[0,y]>0$ and $A[0,y]=A[0,y-1]$ then set $Q[0,y]=Q[0,y-1]$. If $A[0,y]>0$ and $A[0,y]\neq A[0,y-1]$ then increment L and set $Q[0,y]=L$. This takes care of labeling the first element in the row.

Labeling the rest of the elements in a row requires that we look at both the neighbor above and the neighbor to the left. For simplicity, let us refer to the current pixel as p, the neighbor to the left as s and the one above as t. If $A[p]=A[s]$ but $A[p]\neq A[t]$ then set $Q[p]=Q[s]$. If $A[p]\neq A[s]$ but $A[p]=A[t]$ then set $Q[s]=Q[t]$. If $A[p]\neq A[t]$ and $A[p]\neq A[s]$ then increment L and set $Q[p]=L$. If $A[p]=A[t]$, $A[p]=A[s]$ and $Q[s]=Q[t]$ then set $Q[p]=Q[t]$. This takes care of all the cases except $A[p]=A[t]$, $A[p]=A[s]$ and $Q[s]\neq Q[t]$. This means that pixels s and t have the same values but different labels. This tells us that s and t are really in the same component and should have the same label, but we did not know that until now. We therefore label pixel p with the smaller of the two labels and record the fact that the larger label value is equivalent to the smaller one. Let L_1 be the smaller value and L_2 be the larger value. Then set $Q[p]=L_1$ and $EQ[L_2]=L_1$. We will resolve these equivalencies in the next pass.

The first three rows of an image with their labels to this point are shown in the diagram below. Because of the shape of the connected components and the order in which pixels are examined, pixels in the same connected component may have different labels. The system finds that the following are equivalent: (1,2), (1,6), (3,4) and (5,7). At this point in the algorithm the equivalency table would look like EQ=[0,1,1,3,3,5,1,5]. Labels {1,2,6}, {3,4} and {5,7} are equivalent.

0	0	1	1	0	0	2	2	2	3	3	0	4	4	0	5	5	5	0
0	0	0	1	1	1	**1**	1	1	3	0	0	0	4	0	0	5	0	0
0	0	6	**1**	0	0	0	1	0	3	3	3	3	**3**	7	7	**5**	5	

Step 4: At this point the first pass has been completed. We now find the equivalency classes from the EQ array and relable the *Q* matrix.