Connected-component labeling

From Wikipedia, the free encyclopedia

Connected-component labeling (alternatively **connected-component analysis**, **blob extraction**, **region labeling**, **blob discovery**, or **region extraction**) is an algorithmic application of graph theory, where subsets of *connected components* are uniquely *labeled* based on a given heuristic. Connected-component labeling is not to be confused with segmentation.

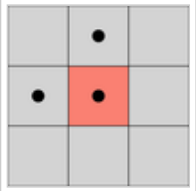
Connected-component labeling is used in computer vision to detect connected regions in binary digital images, although color images and data with higher dimensionality can also be processed.[1][2] When integrated into an image recognition system or human-computer interaction interface, connected component labeling can operate on a variety of information.[3][4] Blob extraction is generally performed on the resulting binary image from a thresholding step. Blobs may be counted, filtered, and tracked.

Blob extraction is related to but distinct from blob detection.

Contents

Overview

A graph, containing vertices and connecting edges, is constructed from relevant input data. The vertices contain information required by the comparison heuristic, while the edges indicate connected 'neighbors'. An algorithm traverses the graph, labeling the vertices based on the connectivity and relative values of their neighbors. Connectivity is determined by the medium; image graphs, for example, can be 4-connected or 8-connected.[5]



4-connectivity

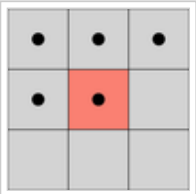
Following the labeling stage, the graph may be partitioned into subsets, after which the original information can be recovered and processed .

Algorithms

The algorithms discussed can be generalized to arbitrary dimensions, albeit with increased time and space complexity.

One component at a time

This is a fast and very simple method to implement and understand. It is based on graph traversal methods in graph theory. In short, once the first pixel of a connected component is found, all the connected pixels of that connected component are labelled before going onto the next pixel in the image. This algorithm is part of Vincent and Soille's watershed segmentation algorithm,[6] other implementations also exist.[7]


8-connectivity

In order to do that a linked list is formed that will keep the indexes of the pixels that are connected to each other, steps (2) and (3) below. The method of defining the linked list specifies the use of a depth or a breadth first search. For this particular application, there is no difference which strategy to use. The simplest kind of a last in first out queue implemented as a singly linked list will result in a depth first search strategy.

It is assumed that the input image is a binary image, with pixels being either background or foreground and that the connected components in the foreground pixels are desired. The algorithm steps can be written as:

1. Start from the first pixel in the image. Set "curlab" (short for "current label") to 1. Go to (2).
2. If this pixel is a foreground pixel and it is not already labelled, then give it the label "curlab" and add it as the first element in a queue, then go to (3). If it is a background pixel, then repeat (2) for the next pixel in the image.
3. Pop out an element from the queue, and look at its neighbours (based on any type of connectivity). If a neighbour is a foreground pixel and is not already labelled, give it the "curlab" label and add it to the queue. Repeat (3) until there are no more elements in the queue.
4. Go to (2) for the next pixel in the image and increment "curlab" by 1.

Note that the pixels are labelled before being put into the queue. The queue will only keep a pixel to check its neighbours and add them to the queue if necessary. This algorithm only needs to check the neighbours of each foreground pixel once and doesn't check the neighbours of background pixels.

Two-pass

Relatively simple to implement and understand, the two-pass algorithm[8] iterates through 2-dimensional, binary data. The algorithm makes two passes over the image: the first pass to assign temporary labels and record equivalences and the second pass to replace each temporary label by the smallest label of its equivalence class.

The input data can be modified *in situ* (which carries the risk of data corruption), or labeling information can be maintained in an additional data structure.

Connectivity checks are carried out by checking neighbor pixels' labels (neighbor elements whose labels are not assigned yet are ignored), or say, the North-East, the North, the North-West and the West of the current pixel (assuming 8-connectivity). 4-connectivity uses only North and West neighbors of the current pixel. The following conditions are checked to determine the value of the label to be assigned to the current pixel (4-connectivity is assumed)

Conditions to check:

1. Does the pixel to the left (West) have the same value as the current pixel?
    1. **Yes** – We are in the same region. Assign the same label to the current pixel
    2. **No** – Check next condition
2. Do both pixels to the North and West of the current pixel have the same value as the current pixel but not the same label?
    1. **Yes** – We know that the North and West pixels belong to the same region and must be merged. Assign the current pixel the minimum of the North and West labels, and record their equivalence relationship
    2. **No** – Check next condition
3. Does the pixel to the left (West) have a different value and the one to the North the same value as the current pixel?
    1. **Yes** – Assign the label of the North pixel to the current pixel
    2. **No** – Check next condition
4. Do the pixel's North and West neighbors have different pixel values than current pixel?
    1. **Yes** – Create a new label id and assign it to the current pixel

The algorithm continues this way, and creates new region labels whenever necessary. The key to a fast algorithm, however, is how this merging is done. This algorithm uses the union-find data structure which provides excellent performance for keeping track of equivalence relationships.[9] Union-find essentially stores labels which correspond to the same blob in a

disjoint-set data structure, making it easy to remember the equivalence of two labels by the use of an interface method E.g.: findSet(l). findSet(l) returns the minimum label value that is equivalent to the function argument 'l'.

Once the initial labeling and equivalence recording is completed, the second pass merely replaces each pixel label with its equivalent disjoint-set representative element.

A faster-scanning algorithm for connected-region extraction is presented below.[10]

On the first pass:

1. Iterate through each element of the data by column, then by row (Raster Scanning)
2. If the element is not the background
    1. Get the neighboring elements of the current element
    2. If there are no neighbors, uniquely label the current element and continue
    3. Otherwise, find the neighbor with the smallest label and assign it to the current element
    4. Store the equivalence between neighboring labels

On the second pass:

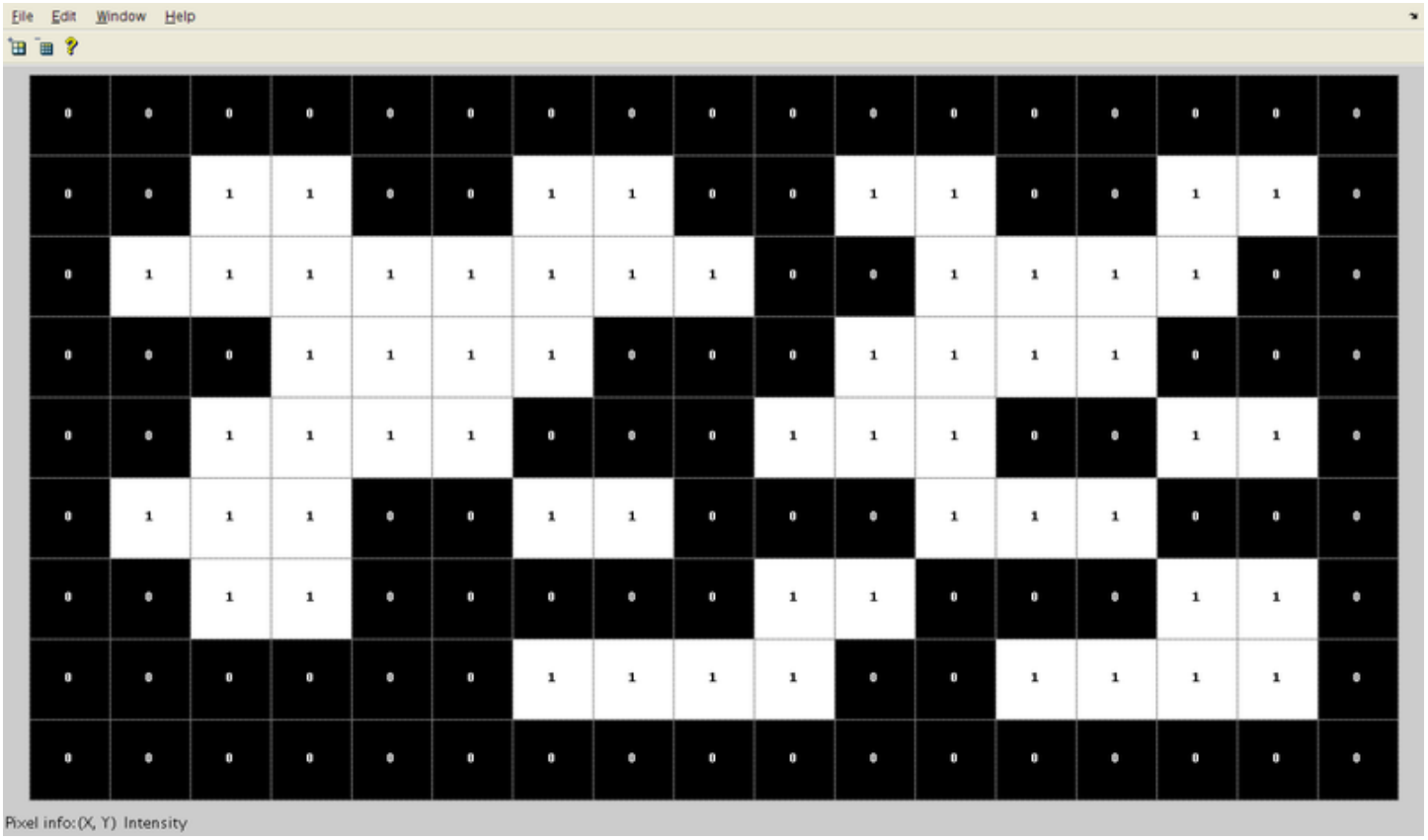
1. Iterate through each element of the data by column, then by row
2. If the element is not the background
    1. Relabel the element with the lowest equivalent label

Here, the **background** is a classification, specific to the data, used to distinguish salient elements from the **foreground**. If the background variable is omitted, then the two-pass algorithm will treat the background as another region.

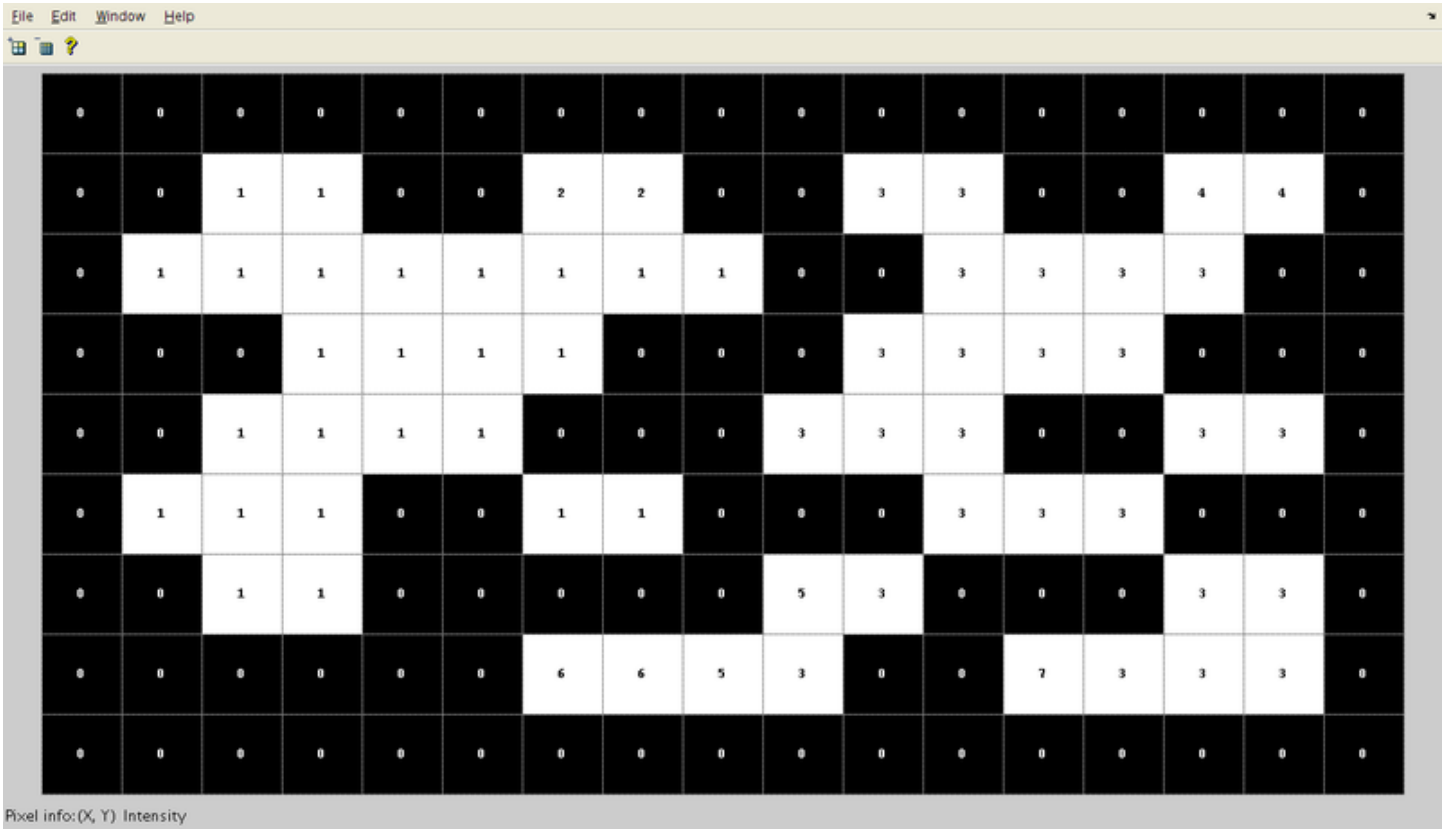
Graphical example of two-pass algorithm

1. The array from which connected regions are to be extracted is given below (8-connectivity based).

We first assign different binary values to elements in the graph. Attention should be paid that the "0~1" values written on the center of the elements in the following graph are elements' values. While, the "1,2,...,7" values in the next two graphs are the elements' labels. The two concepts should not be confused.



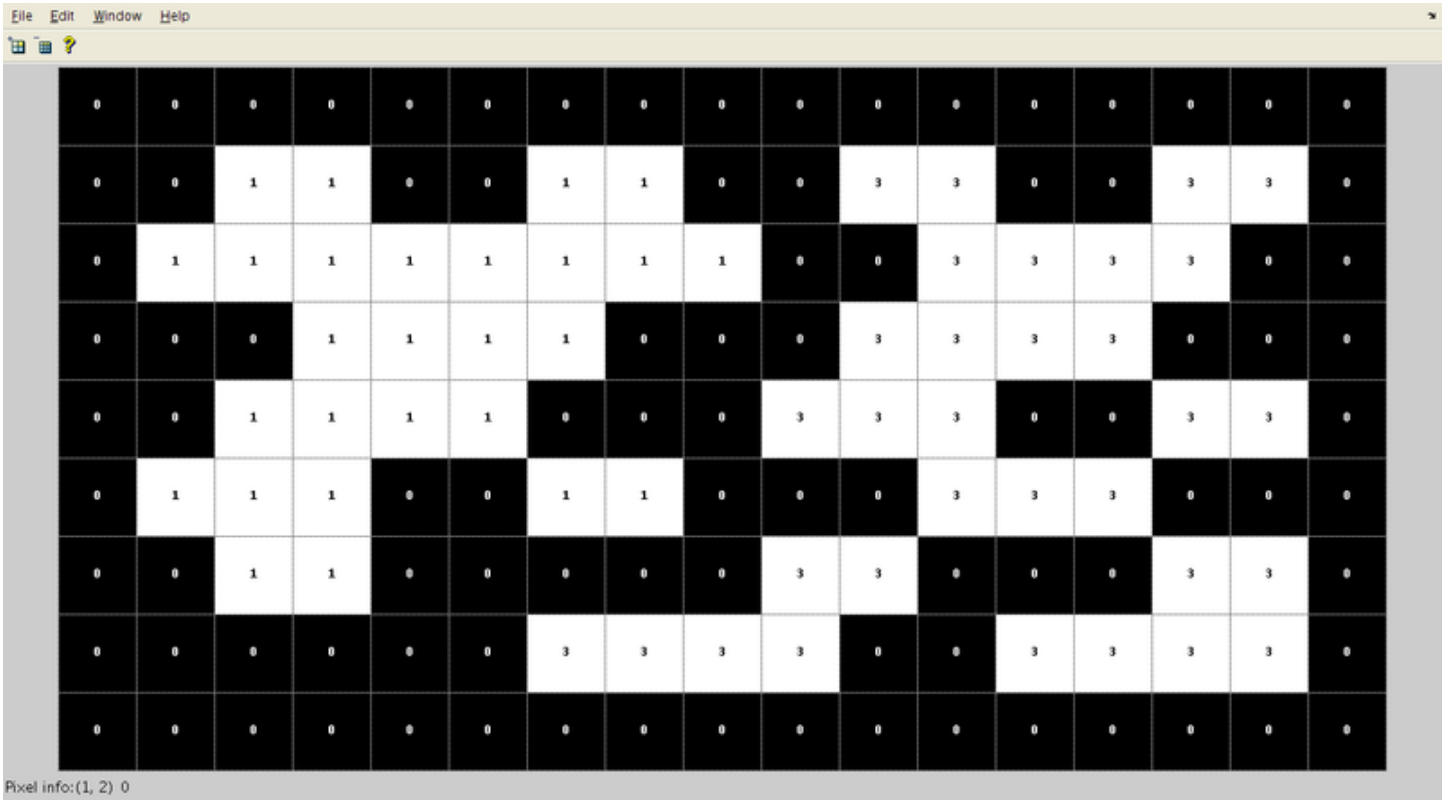
2. After the first pass, the following labels are generated. A total of 7 labels are generated in accordance with the conditions highlighted above.



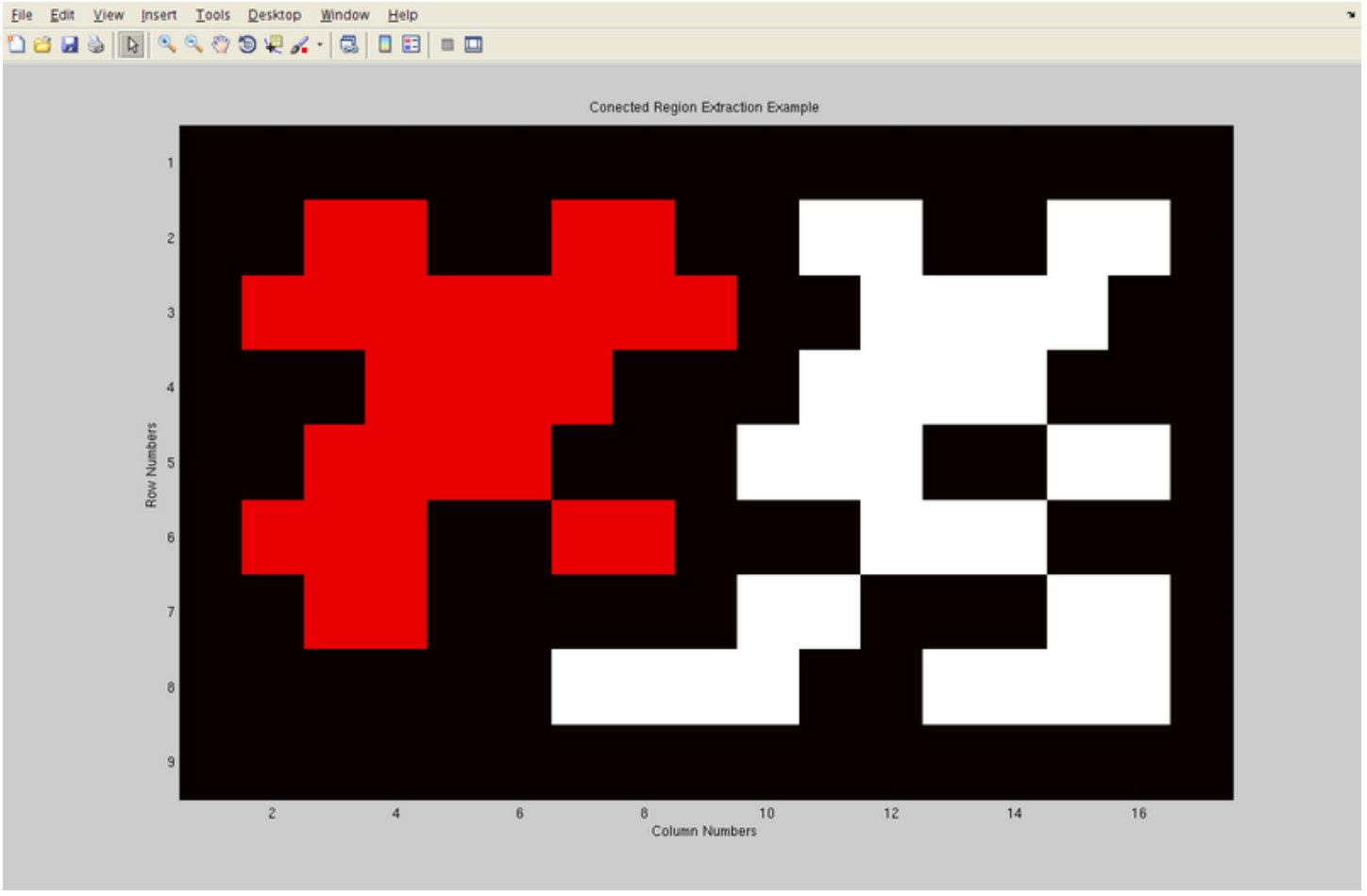
The label equivalence relationships generated are,

Set ID	Equivalent Labels
1	1,2
2	1,2
3	3,4,5,6,7
4	3,4,5,6,7
5	3,4,5,6,7
6	3,4,5,6,7
7	3,4,5,6,7

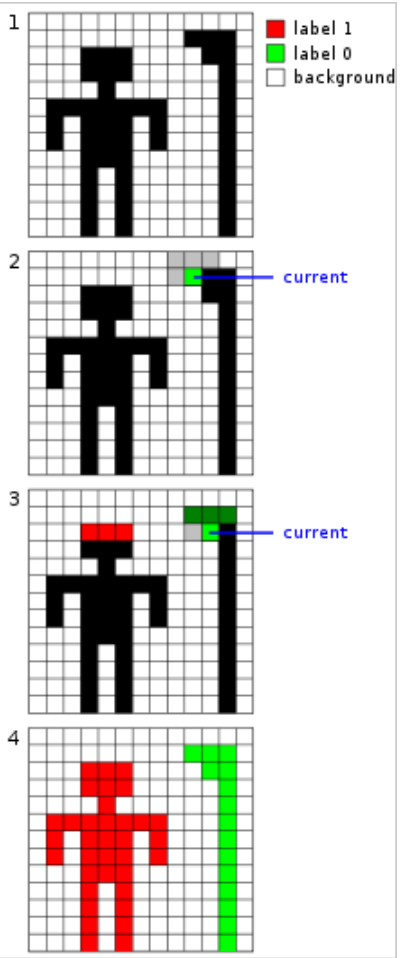
3. Array generated after the merging of labels is carried out. Here, the label value that was the smallest for a given region "floods" throughout the connected region and gives two distinct labels, and hence two distinct labels.



4. Final result in color to clearly see two different regions that have been found in the array.



The **pseudocode** is as follows:



Sample graphical output from running the two-pass algorithm on a binary image. The first image is unprocessed, while the last one has been recolored with label information. Darker hues indicate the neighbors of the pixel being processed.

```
algorithm TwoPass(data)
   linked = []
   labels = structure with dimensions of data, initialized with the value of Background

   First pass

   for row in data:
       for column in row:
           if data[row][column] is not Background

               neighbors = connected elements with the current element's value

               if neighbors is empty
                   linked[NextLabel] = set containing NextLabel
                   labels[row][column] = NextLabel
                   NextLabel += 1

               else

                   Find the smallest label

                   L = neighbors labels
                   labels[row][column] = min(L)
```

```
                for label in L
                    linked[label] = union(linked[label], L)

    Second pass

    for row in data
        for column in row
            if data[row][column] is not Background
                labels[row][column] = find(labels[row][column])

    return labels
```

The *find* and *union* algorithms are implemented as described in union find.

Sequential algorithm

Create a region counter

Scan the image (in the following example, it is assumed that scanning is done from left to right and from top to bottom):

- For every pixel check the *north* and *west* pixel (when considering 4-connectivity) or the *northeast*, *north*, *northwest*, and *west* pixel for 8-connectivity for a given region criterion (i.e. intensity value of 1 in binary image, or similar intensity to connected pixels in gray-scale image).
- If none of the neighbors fit the criterion then assign to region value of the region counter. Increment region counter.
- If only one neighbor fits the criterion assign pixel to that region.
- If multiple neighbors match and are all members of the same region, assign pixel to their region.
- If multiple neighbors match and are members of different regions, assign pixel to one of the regions (it doesn't matter which one). Indicate that all of these regions are equivalent.
- Scan image again, assigning all equivalent regions the same region value.

Others

Some of the steps present in the two-pass algorithm can be merged for efficiency, allowing for a single sweep through the image. Multi-pass algorithms also exist, some of which run in linear time relative to the number of image pixels.[11]

In the early 1990s, there was considerable interest in parallelizing connected-component algorithms in image analysis applications, due to the bottleneck of sequentially processing each pixel.[12]

The interest to the algorithm arises again with an extensive use of CUDA.

One-pass version

A one pass (also referred to as single pass) version of the connected-component-labeling algorithm is given as follows. The algorithm identifies and marks the connected components in a single pass. The run time of the algorithm depends on the size of the image and the number of connected components (which create an overhead). The run time is comparable to the two pass algorithm if there are a lot of small objects distributed over the entire image such that they cover a significant number of pixels from it. Otherwise the algorithm runs fairly fast.

Algorithm:

1. Connected-component matrix is initialized to size of image matrix.
2. A mark is initialized and incremented for every detected object in the image.
3. A counter is initialized to count the number of objects.
4. A row-major scan is started for the entire image.
5. If an object pixel is detected, then following steps are repeated while (Index !=0)
    1. Set the corresponding pixel to 0 in Image.
    2. A vector (Index) is updated with all the neighboring pixels of the currently set pixels.
    3. Unique pixels are retained and repeated pixels are removed.
    4. Set the pixels indicated by Index to mark in the connected-component matrix.
6. Increment the marker for another object in the image.

The source code is as follows(4-connectivity based):

```
One-Pass(Image)
           [M, N]=size(Image);
```

```
          Connected = zeros(M,N);
          Mark = Value;
          Difference = Increment;
          Offsets = [-1; M; 1; -M];
          Index = [];
          No_of_Objects = 0;

 for i: 1:M :
     for j: 1:N:
          if(Image(i,j)==1)
               No_of_Objects = No_of_Objects +1;
               Index = [((j-1)*M + i)];
               Connected(Index)=Mark;
               while ~isempty(Index)
                    Image(Index)=0;
                    Neighbors = bsxfun(@plus, Index, Offsets');
                    Neighbors = unique(Neighbors(:));
                    Index = Neighbors(find(Image(Neighbors)));
                    Connected(Index)=Mark;
               end
               Mark = Mark + Difference;
          end
    end
end
```

Hardware architectures

The emergence of FPGAs with enough capacity to perform complex image processing tasks also led to high-performance architectures for connected component labeling.[13][14] Most of these architectures utilize the single pass variant of this algorithm, because of the limited memory resources available on an FPGA. These types of connected component labeling architectures are able to process several image pixels in parallel, thereby enabling a high throughput at low processing latency to be achieved.

See also

- Image analysis
- Computer vision
- Feature extraction
- Connected component (graph theory)
- Graph traversal
- Union find
- Flood fill
- Blob detection

References

1. H. Samet and M. Tamminen (1988). "Efficient Component Labeling of Images of Arbitrary Dimension Represented by Linear Bintrees". *IEEE Transactions on Pattern Analysis and Machine Intelligence* (TIEEE Trans. Pattern Anal. Mach. Intell.) **10**: 579. doi:10.1109/34.3918.
2. Michael B. Dillencourt and Hannan Samet and Markku Tamminen (1992). "A general approach to connected-component labeling for arbitrary image representations". J. ACM.
3. Weijie Chen, Maryellen L. Giger and Ulrich Bick (2006). "A Fuzzy C-Means (FCM)-Based Approach for Computerized Segmentation of Breast Lesions in Dynamic Contrast-Enhanced MR Images". *Academic radiology* (Academic Radiology) **13** (1): 63–72. doi:10.1016/j.acra.2005.08.035. PMID 16399033.
4. Kesheng Wu, Wendy Koegler, Jacqueline Chen and Arie Shoshani (2003). "Using Bitmap Index for Interactive Exploration of Large Datasets". SSDBM.
5. R. Fisher, S. Perkins, A. Walker and E. Wolfart (2003). "Connected Component Labeling".
6. Vincent, Luc; Soille, Pierre (June 1991). "Watersheds in digital spaces: an efficient algorithm based on immersion simulations". *IEEE Transactions on Pattern Analysis and Machine Intelligence* **13** (6): 583. doi:10.1109/34.87344.
7. Abubaker, A; Qahwaji, R; Ipson, S; Saleh, M (2007). "One Scan Connected Component Labeling Technique". *Signal Processing and Communications, 2007. ICSPC 2007. IEEE International Conference*: 1283. doi:10.1109/ICSPC.2007.4728561.
8. Shapiro, L., and Stockman, G. (2002). *Computer Vision* (PDF). Prentice Hall. pp. 69–73.
9. Introduction to Algorithms, [1] (http://books.google.ca/books?id=NLngYyWFl_YC&dq=introduction+to+algorithms&source=gbs_navlinks_s), pp498
10. Lifeng He; Yuyan Chao,; Suzuki, K. (1 May 2008). "A Run-Based Two-Scan Labeling Algorithm". *IEEE Transactions on Image Processing* **17** (5): 749–756. doi:10.1109/TIP.2008.919369. PMID 18390379.
11. Kenji Suzuki and Isao Horiba and Noboru Sugie (2003). "Linear-time connected-component labeling based on sequential local operations". *Computer Vision and Image Understanding* (Comput. Vis. Image Underst.) **89**: 1. doi:10.1016/S1077-3142(02)00030-9.

12. Yujie Han and Robert A. Wagner (1990). "An efficient and fast parallel-connected component algorithm". J. ACM.
13. D. G. Bailey, C. T. Johnston, N. Ma (2008). "Connected components analysis of streamed images".
14. M. J Klaiber, D. G Bailey, Y. Baroud, S. Simon (2015). "A Resource-Efficient Hardware Architecture for Connected Components Analysis". IEEE Transactions on Circuits and Systems for Video Technology.

- Horn, Berthold Klaus Paul (1986). *Robot Vision*. MIT Press. pp. 69–71. ISBN 0-262-08159-8.

External links

- Implementation in C# (http://www.codeproject.com/Articles/336915/Connected-Component-Labeling-Algorithm)
- about Extracting objects from image and Direct Connected Component Labeling Algorithm (https://drive.google.com/file/d/0B8gQ5d6E54ZDd3JiMURVU05HM2s/edit?usp=sharing)