A small-space algorithm for removing small connected components from a binary image

PAPER

A Small-Space Algorithm for Removing Small Connected Components from a Binary Image

Tetsuo ASANO[†], *and* **Revant KUMAR**[††],

SUMMARY Given a binary image $\mathscr{I}$ and a threshold t, the **size-thresholded binary image** $\mathscr{I}(t)$ defined by $\mathscr{I}$ and t is the binary image after removing all connected components consisting of at most t pixles. This paper presents space-efficient algorithms for computing a **size-thresholded binary image** for a binary image of n pixels, assuming that the image is stored in a read-only array with random-access. With regard to the problem, there are two cases depending on how large the threshold t is, namely, **Relatively large threshold** where $t = \Omega(\sqrt{n})$, and **Relatively small threshold** where $t = O(\sqrt{n})$. In this paper, a new algorithmic framework for the problem is presented. From an algorithmic point of view, the problem can be solved in $O(n)$ time and $O(n)$ work space. We propose new algorithms for both the above cases which compute the size-threshold binary image for any binary image of n pixels in $O(n \log n)$ time using only $O(\sqrt{n})$ work space.

key words: *algorithm, binary image, connected component, image processing, small work space.*

1. Introduction

Demand for embedded software is growing toward intelligent hardware such as scanners, digital cameras etc. One of the most important aspects and also differences between ordinary software in computers and embedded software comes from constraints on the size of local memory. For example, to design an intelligent scanner a number of algorithms should be embedded in the scanner. In most of those cases, the size of pictures is increasing while the amount of work space available for such software is severely limited. In the sense algorithms which require a restricted amount of work space and run reasonably fast are desired.

In this paper we propose space efficient algorithms for removing small connected components from a binary image. Removal of small irrelevant components, considered to be noise, improves the binary image. This find applications in Mathematical Morphology and stroke-like pattern noise. The tasks considered have straightforward solutions [3]–[12] if sufficient memory (size proportional to the size of the image) is available. Solving the same tasks with restricted memory, without severely compromising the running time, is more of a challenge.

An interesting application of our algorithms is the following. Suppose we are given a color image $\mathscr{C}$. Given a color range, say $[r_1, r_2]$ for red, $[g_1, g_2]$ for green, and $[b_1, b_2]$ for blue, we can easily compute a binary image by determining pixel value to be 1 if and only if its color value is in the specified color range. Then, we can compute connected components in the binary image. Usually meaningful information is associated with large connected components since small components are considered as noises. For example, we can measure how clear the boundaries are by following boundaries of connected components while computing color differences between inside and outside. If we do not worry about work space, we first compute the binary image using one array and then implement connected components labeling using another array. Then, it is easy to measure component sizes. Everything is done in linear time. But what happens if we do not have enough work space? Our algorithms use only $O(\sqrt{n})$ work space to compute the measures. Thus, this is a new approach to image analysis/evaluation.

Manuscript received September 20, 2012.
Manuscript revised January 1, 1.
Final manuscript received January 1, 1.

[†]School of Information Science, JAIST, Japan
[††]Department of Mathematics and Computing, Indian Institute of Technology (IIT) at Guwahati, India

DOI: 10.1587/transinf.E0.D.1

2. Basic Algorithm for Connected Components Labeling

The problem known as the connected components labeling is to assign integer labels to all white pixels in a given binary image so that two white pixels have the same label if and only if they belong to the same connected component. The first author presented a small-space algorithm for connected components labeling for a binary image of n pixels which runs in $O(n \log n)$ time and $O(\sqrt{n})$ work space [1]. The algorithm examines pixels in raster order (see Figure 1(a)). To simplify the algorithm we implicitly assume that a given image is surrounded by black pixels as shown in Figure 1(b). Since pixels are linearly ordered, if the current pixel is p, then the previous pixel is $p-1$ (implicitly assuming that we scan pixels except those on the left boundary at the 0-th column). When we encounter a boundary edge, we traverse the boundary in two opposite directions while maintaining two edges, forward edge e_f and backward edge e_b (see Figure 1(c)). If we maintain correct labels in two rows, we can obtain correct labels when boundaries go below the current rows. An idea for acceleration is a bidirectional search. In the algorithm, NextEdge(e) and PrevEdge(e) are important functions to compute the next and previous, respectively, of an edge e on a boundary. Boundaries may be different depending on which connectivity, i.e., 4-connectivity or 8-connectivity, is used. However, in any case both functions can be computed in constant time. Refer to the literature [1] for the detail.

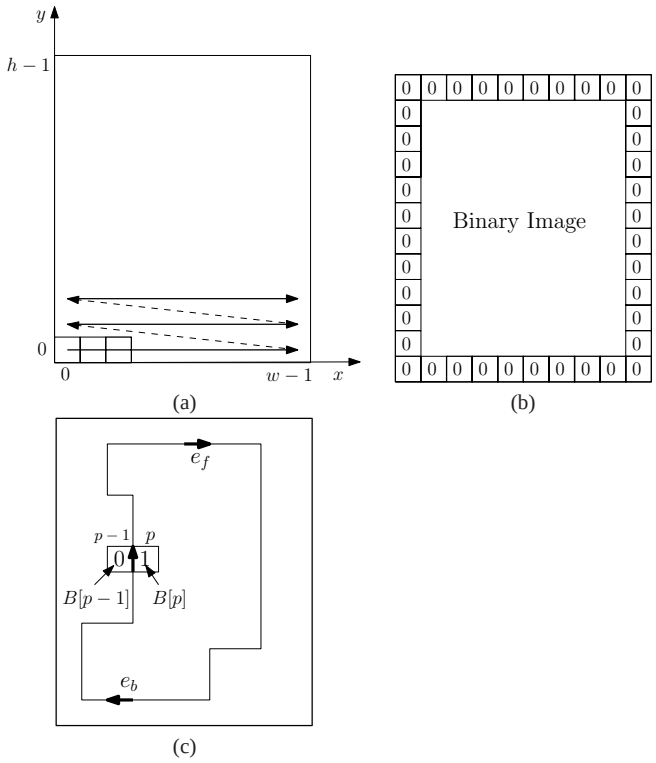


Fig. 1 Preliminary definitions: (a) raster order among all pixels, (b) surrounding black pixels, and (c) forward and backward edges on the boundary starting from the edge between two pixels p and $p-1$.

Theorem 2.1: *[1]* Given a binary image of n pixels in a read-only array, its connected components labeling can be computed and those labels can be reported in raster order in $O(n \log n)$ time using only $O(\sqrt{n})$ work space.

The algorithm is given below as **Algorithm 1**.

An easy extension of the algorithm is to report sizes of all connected components (just output them without storing them). To extend the above algorithm so that it can count the size (number of pixels) of connected components, we use an array for a compressed list of labels. As is stated above, we maintain two rows of the labeling matrix, one for the previous row and the other for the current row. Suppose we have computed labels in the current row. Then, we first convert the content of the row into the compressed list, which is a sorted list of labels together with their frequencies (number of occurrences). Let $\mathscr{L}_i$ be the compressed list of the current row (say, the i^{th} row) Let $\mathscr{L}_{i-1}$ be the compressed list of the previous row. Now, we compare the two lists. Then, there are three cases:

Continuing label: A label k is called **continuing** if it appears both in $\mathscr{L}_i$ and $\mathscr{L}_{i-1}$. In this case the label in the previous row keeps the size of the component before the current row and the label in the current row keeps that in the current row. Thus, we just add them and replace the label information in $\mathscr{L}_i$.

New label: A label k is called **new** if it is not in $\mathscr{L}_{i-1}$ but in $\mathscr{L}_i$.

Algorithm 1: Compute the connected components labeling.

Input: A binary image $B[i], i = 0, \ldots, wh-1$, linearly ordered in raster order.
Output: The labeling matrix of the input binary image in raster order.
Array: lab[2][0..$w-1$]. `// label arrays for two rows`
$L = 0$; `// the number of labels used so far`
Initialize the array lab[2][0..$w-1$], $a = 0, b = 1$
for $i = 1$ *to* $h-1$ **do**
  `// the labels in the previous row are computed in lab[b][.]`
  for $j = 1$ *to* $w-1$ **do** `// pixel in the raster order`
    $p = i \times w + j$; `// pixel at` (i, j).
    switch *the value of* $B[p-1]B[p]$ **do**
      case '00' `// empty pixel p`
        lab[a][p] $= 0$; break;
      case '11' `// next pixel of the run`
        lab[a][p] = lab[a][$p-1$]; break;
      case '01' `// boundary pixel p`
        $e_f = e_b =$ the left edge of the pixel p;
        `// two pointers in two directions`
        $d = 0$.; `// search direction: 0 for forward`
        repeat `// traverse the boundary in two directions`
          if $d = 0$ **then** $e_f = NextEdge(e_f)$ **else** $e_b = PrevEdge(e_b)$ $d = 1 - d$.
        until $e_f = e_b$ *or* e_f *or* e_b *is adjacent to a pixel with a positive label k in the row i or* $i-1$;
        if $e_f = e_b$ **then** `// new component`
          $L = L + 1, k = L$; `// new label`
        lab[a][p] $= k$; break;
      otherwise `// case '10',`
        lab[a][p] $= 0$; break;
  Output the array lab[a][0..$w-1$].
  Exchange the roles of a and b ($a = 1-a, b = 1-b$).

Dead label: A label k is called **dead** if it is in $\mathscr{L}_{i-1}$ but not in $\mathscr{L}_i$. This means the component of the label k finishes at the previous row and does not extend to the row above it. In this case the component dies. Whenever we find such a dying label or component, we report its label together with its component size.

Example

Suppose the row i and row $i-1$ are labeled as follows:

```
000333005507770000333333000000000
003330055500006660033330022220000
```

In this example, the labels 3 and 5 are continuing and the label 7 is new. The labels 6 and 2 are dead since they do not appear in the second row.

```
Compressed list at row i:
   (3(9), 5(2), 7(3))
Compressed list at row i-1:
   (2(39), 3(126), 5(31), 6(3))
```

In this example we suppose that we already know that the component of the label 2 has already 39 pixels before the current row. Since the label 2 is dead, we report "Connected component of label 2 which has 39 pixels." For the continuing labels we add the two counts. For example the label 3 has appeared 126 times before the current row and 9 times at the current row, and thus the total count will be 135. Thus, the compressed list for the current row should be modified as follows.

```
Compressed list at row i:
    (3(135), 5(33), 7(3))
Compressed list at row i-1:
    (2(39), 3(126), 5(31), 6(3))
```

All these operations are done in $O(\sqrt{n}\log n)$ time. Work space we need is bounded by $O(\sqrt{n})$ since each row can contain at most $\sqrt{n}/2$ different labels (and thus as many connected components).

Theorem 2.2: Given a binary image of n pixels in a read-only array, we can report sizes of all connected components (without storing them) in $O(n\log n)$ time and $O(\sqrt{n})$ work space.

A connected component is characterized by a unique external boundary and possibly more than one internal boundary for a hole. Every boundary consists of edges between adjacent pixels. Then, each boundary has a unique edge that is the leftmost one among those lowest vertical edges on the boundary. Such an edge is called a **canonical edge** of the boundary. Especially, the canonical edge of the external boundary of a connected component C is called the canonical edge of C. The canonical edge plays an important role in our algorithms.

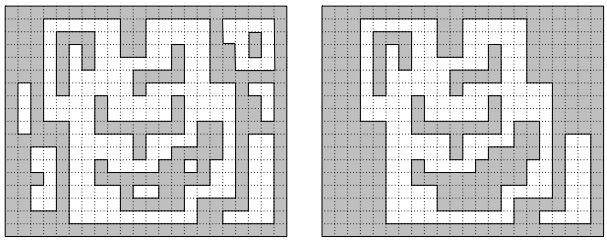
Corollary 2.3: Given a binary image of n pixels in a read-only array, we can report canonical edges and smallest bounding rectangles of all connected components (without storing them) in $O(n\log n)$ time and $O(\sqrt{n})$ work space.

Proof To report canonical edges of connected components we just keep canonical edges whenever we find new connected components. To report smallest bounding rectangles we should maintain the smallest and largest coordinates of component pixels. □

3. Binary Image with Small Components Removed

Let $\mathscr{I}$ be a binary image of n pixels and t be a positive integer. Then, $\mathscr{I}(t)$ is the binary image obtained by removing all small connected components from $\mathscr{I}$. More formally, a pixel p in $\mathscr{I}(t)$ is white if and only if it is white in $\mathscr{I}$ and it belongs to a connected component of size $> t$ in $\mathscr{I}$. The binary image $\mathscr{I}(t)$ is called a **size-thresholded binary image** of $\mathscr{I}$.

See Figure 2. Eight connected components are included in the input binary image shown in (a). The image in (b) shows the size-thresholded image after removing connected components of sizes at most 12.



(a) (b)

Fig. 2 Removing small connected components. (a) An input binary image, and (b) the binary image after removing all connected components of sizes ≤ 12.

Problem: Given a binary image $\mathscr{I}$ of n pixels in a read-only array and a threshold t, output its size-thresholded binary image $\mathscr{I}(t)$ in raster order.

There are two cases depending on how large the threshold t is.

Case 1. Relatively large threshold: $t = \Omega(\sqrt{n})$

In the case the number of connected components of size $> t$ must be bounded by $O(\sqrt{n})$ since the total number of pixels is n. Thus, the following algorithm is possible:

Algorithm 0:

(1) Apply the algorithm for connected components labeling using $O(\sqrt{n})$ space. In the algorithm we maintain a set of labels of those connected components of sizes $> t$ in a simple linear array D of size $O(\sqrt{n})$. More exactly, we also maintain the sizes of connected components. Whenever a component dies (that is, it does not extend to the row examined), we check its size and if it is greater than t then we insert the label into D. After completing the connected components labeling, we sort those labels in the array.

(2) Implement the connected components labeling algorithm again. At each white pixel in raster order, however, we compute its label and see whether it is included in the array D. We output 1 if its label is in D and 0 otherwise.

Lemma 3.1: Algorithm 0 above computes a size-thresholded binary image for any binary image of n pixels and a threshold $t = \Omega(\sqrt{n})$ in $O(n \log n)$ time using $O(\sqrt{n})$ work space.

Proof In the algorithm we only watch a set of connected components intersecting the current row. Since there are only $O(\sqrt{n})$ columns, the number of such components is bounded by $O(\sqrt{n})$. We also maintain a set of labels with their associated sizes. We may have $O(n)$ different labels, but we just maintain large components of size $t = \Omega(\sqrt{n})$. Thus, the work space we need is $O(\sqrt{n})$. □

Case 2. Relatively small threshold: $t = O(\sqrt{n})$

We modify the algorithm for connected components labeling. In the algorithm we scan pixels in raster order, row by row, from left to right. We maintain pixel labels in two rows. Let $lab[a][x]$ be a label of a pixel at the xth column in the current row, and $lab[b][x]$ be a label in the previous row. In addition to the label information, we also maintain output values. Let $val[a][[x]$ and $val[b][x]$ be output binary values at those pixels. Suppose we are now looking at a pixel p at the xth column in the current row. We first determine whether the vertical edge e_s immediately to the left of p is canonical edge of an external boundary. The first condition we check is whether the pixel just below p is black. If the pixel is white, then the boundary must extend further below and thus e_s is not a canonical edge. The second condition is that we find no vertical edge e preceding e_s in the raster order when we traverse the boundary. The edge e precedes e_s in the raster order when e lies in the previous row (or even lower row) or lies in the current row but to the left of e_s. The total time we need for the second condition is bounded by $O(n \log n)$ if we use bidirectional search.

When we find an external canonical edge e_s just left to a pixel p, we create a new label L and put the label at the position. What about an output there? We have to know how large the connected component starting at the pixel p. For the purpose we use a traditional folklore algorithm known as a wave propagation method. We start from a set S which consists of the pixel p. Then, we expand the set S by incorporating any white pixel adjacent to those white pixels already incorporated into the set S. One of disadvantages of the method is its space complexity. Of course, we need a data structure storing all pixels in one connected component, which may contain $O(n)$ pixels. Fortunately, we can stop the expansion as soon as the $t+1$ pixels are put into the set since it means that the component is large. If we implement the wave propagation method using a binary search tree, then the wave propagation is done in $O(t \log t)$ time using $O(t) = O(\sqrt{n})$ space. Moreover, no pixel belongs to more than one connected component. This implies that sets of pixels stored during the wave propagation step are disjoint. Therefore, the total time we need for the step is bounded by $O(n \log t) = O(n \log n)$.

On the other hand, when the edge e_s to the left of the current pixel is not an external canonical edge, we must have found a vertical edge e preceding e_s in the raster order if we follow the boundary from e_s. Due to our assumption we can assume that the labels have been correctly computed. Thus, if we find such a vertical edge e on the same boundary in the previous row or in the same row but to the left of e, we get a correct label by looking at a label of the pixel associated with e and also its corresponding output value. If the output value is 1, it means that the corresponding component is large enough, and thus we can output 1. Otherwise, the component of the label is small, and we need to output 0. In this way we can correctly compute the label and output value at the current pixel.

Function `WavePropagation(`p`,`t`)`
Input: A binary image of n pixels, a starting white pixel p, and a threshold t.
Output: 1 if the component containing p has more than t pixels, and 0 otherwise.
Array: $Q[1..t+1]$ an array and a binary search tree T of size $t+1$.
begin
  $Q = \{p\}$. `// a queue (a simple array)`
  $T = \{p\}$. `// a binary search tree with pixel numbers as keys.`
  $c = 1$. `// a counter`
  while Q *is not empty and* $c \leq t$ **do**
    Pop a pixel q out of Q.
    for *each* p *in the neighborhood of* q **do**
      if $p \notin T$ **then**
        Insert p into T and Q, and
        increment c.
  if $c > t$ **then** return 1 **else** return 0.
end

Theorem 3.2: The algorithm 2 computes a size-thresholded binary image for an arbitrary binary image of n pixels and any positive threshold t in $O(n \log n)$ time using $O(\sqrt{n})$ work space.

The algorithm 2 uses two arrays, one for labels and the other for output values. We can do it using only one of them. The idea for the improvement is to put the label '0' for a new connected component if it is not large enough. We compute labels for a row and then output binary values. We output 1 at a pixel p in the current row if and only if the label at p is positive. This simplified algorithm is given as Algorithm 3 below.

4. Conclusions and Open Problems

This paper extended the previous algorithm [1] for connected components labeling so that it can compute a binary

Algorithm 2: Compute the size-thresholded binary image.

Input: A binary image $B[i], i = 0, \ldots, wh-1$ and a threshold $t > 0$ on a component size, where $t = O(\sqrt{n})$.
Output: The size-thresholded binary image in which a pixel is white if and only if it is white in the input image and belongs to a connected component of size $> t$.

```
Array: lab[2][0..w-1].      // label arrays for two rows
val[2][0..w-1].     // output arrays for two rows
L = 0;     // the number of labels used so far
Initialize the array lab[2][.], a = 0, b = 1
Initialize the array val[2][.] = 0, a = 0, b = 1
for i = 1 to h-1 do
    // the labels in the previous row are computed in lab[b][.]
    // the output values in the previous row are computed in val[b][.]
    for j = 1 to w-1 do     // pixel in the raster order
        p = i × w + j;              // pixel at (i, j).
        switch the value of B[p-1]B[p] do
            case '00'               // empty pixel p
                lab[a][p] = 0; val[a][p] = 0; break;
            case '11'     // next pixel of the run
                lab[a][p] = lab[a][p-1];
                val[a][p] = val[a][p-1]; break;
            case '01'           // boundary pixel p
                e_f = e_b = the left edge of the pixel p. ;
                // two pointers in two directions
                d = 0. ;  // search direction:  0 for forward
                repeat  // traverse the boundary in two directions
                    if d = 0 then e_f = NextEdge(e_f) else
                    e_b = PrevEdge(e_b). d = 1 - d.
                until e_f = e_b or e_f or e_b is adjacent to a
                pixel with a positive label k and value v in
                the row i or i - 1;
                if e_f = e_b then     // new component
                    L = L + 1, k = L ;    // new label
                    if WavePropagation(p, t) = 1 then
                        val[a][p] = 1;
                    else
                        val[a][p] = 0;
                lab[a][p] = k;
                val[a][p] = v; break;
            otherwise                   // case '10',
                lab[a][p] = 0;
                val[a][p] = 0;
    Output the array val[a][.].
    Exchange the roles of a and b (a = 1 - a, b = 1 - b).
```

Algorithm 3: Compute the size-thresholded binary image using one local array.

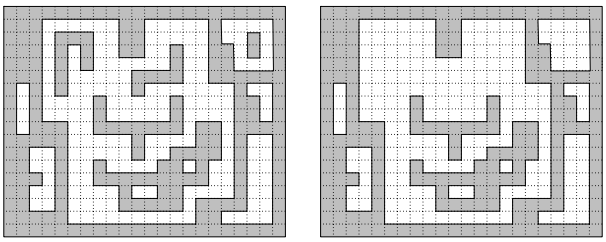
Input: A binary image $B[i], i = 0, \ldots, wh-1$ and a threshold $t > 0$ on a component size, where $t = O(\sqrt{n})$.
Output: The size-thresholded binary image in which a pixel is white if and only if it is white in the input image and belongs to a connected component of size $> t$.

```
Array: lab[2][0..w-1].      // label arrays for two rows
L = 0;     // the number of labels used so far
Initialize the array lab[2][.], a = 0, b = 1
for i = 1 to h-1 do
    // the labels in the previous row are computed in lab[b][.]
    for j = 1 to w-1 do     // pixel in the raster order
        p = i × w + j;              // pixel at (i, j).
        switch the value of B[p-1]B[p] do
            case '00'               // empty pixel p
                lab[a][p] = 0; break;
            case '11'     // next pixel of the run
                lab[a][p] = lab[a][p-1];break;
            case '01'           // boundary pixel p
                e_f = e_b = the left edge of the pixel p. ;
                // two pointers in two directions
                d = 0. ;  // search direction:  0 for forward
                repeat  // traverse the boundary in two directions
                    if d = 0 then e_f = NextEdge(e_f) else
                    e_b = PrevEdge(e_b). d = 1 - d.
                until e_f = e_b or e_f or e_b is adjacent to a
                pixel with a positive label k in the row i or
                i - 1;
                if e_f = e_b then     // new component
                    if WavePropagation(p, t) = 1 then
                        L = L + 1, k = L ;        // new label for a large component
                    else
                        k = 0; ; // a new component was found, but it is small
                lab[a][p] = k; break;
            otherwise                   // case '10',
                lab[a][p] = 0;
    for j = 1 to w-1 do     // pixel in the raster order
        if lab[a][j] > 0 then Output 1 else Output 0.
    Exchange the roles of a and b (a = 1 - a, b = 1 - b).
```

image after removing small connected components of size smaller than a predetermined threshold value. The algorithm runs in $O(n \log n)$ time for a binary image of n pixels using only $O(\sqrt{n})$ work space.

A number of open problems are left. One of the most important ones is to prove a lower bound on the time complexity assuming the work space of $O(\sqrt{n})$.

Another open problem is associated with holes. Figure 3(a) shows the same binary image shown in Figure 2(a). What happens if we remove all small **black** connected components? The resulting binary image is shown in (b) in the figure. Then, the upper right white component has more than 12 pixels since its hole disappeared. An open question is whether there is an efficient small-space algorithm for reporting the binary image after removing small black components first and then removing small white components. It is known [2] that we can remove any small component efficiently using constant extra memory if an input binary image is given as a usual read/write array. Difficulty comes from the fact that an input binary image is given on a read-only array and we are not allowed to store a binary image after removing small black components.



(a) (b)

Fig. 3 Removing small black connected components. (a) An input binary image, and (b) the binary image after removing all black connected components of sizes ≤ 12.

Acknowledgment

The part of this research was partially supported by the Ministry of Education, Science, Sports and Culture, Grant-in-Aid for Scientific Research on Priority Areas and Scientific Research (B).

References

[1] T. Asano and S. Bereg, “A New Framework for Connected Components Labeling of Binary Images,” to appear in Proc. IWCIA 2012, Austin, USA.

[2] T. Asano, “Do We Need a Stack to Erase a Component in a Binary Image?,” Fifth International Conference on FUN WITH ALGORITHMS, pp.16-27, 2010.

[3] P. Bhattacharya, “Connected component labeling for binary images on a reconfigurable mesh architectures,” J. Syst. Arch., 42(4): 309–313, 1996.

[4] F. Chang, C. J. Chen, and C. J. Lu, “A linear-time component-labeling algorithm using contour tracing technique,” Comput. Vis. Image Understand., 93: 206–220, 2004.

[5] M. B. Dillencourt, H. Samet, and M. Tamminen, “A general approach to connected-component labeling for arbitrary image representations,” J. ACM, 39(2): 253–280, Apr. 1992.

[6] T. Goto, Y. Ohta,M. Yoshida, and Y. Shirai, “High speed algorithm-for component labeling,” Trans. IEICE, J72-D-II(2): 247–255, 1989, (in Japanese).

[7] L. He, Y. Chao, and K. Suzuki, “A Run-Based Two-Scan Labeling Algorithm,” IEEE Trans. on Image Processing, 17(5): 749–756, May 2008

[8] R. Klette and A. Rosenfeld, “Digital Geometry: Geometric Methods for Digital Picture Analysis,” Elsevier, 2004.

[9] A. Rosenfeld and J. L. Pfalts, “Sequential operations in digital picture processing,” J. ACM, 13(4): 471–494, Oct. 1966.

[10] A. Rosenfeld, “Connectivity in digital pictures,” J. ACM, 17(1): 146–160, Jan. 1970.

[11] H. Samet, “Connected component labeling using quadtrees,” J. ACM, 28(3): 487–501, Jul. 1981.

[12] K. Suzuki, I.Horiba, and N. Sugie, “Linear-time connected-component labeling based on sequential local operations,” Comput. Vis. Image Understand., 89: 1–23, 2003.