A Very Fast Algorithm for Simultaneously Performing Connected-Component Labeling and Euler Number Computing

Lifeng He, *Senior Member, IEEE*, and Yuyan Chao

***Abstract*— Labeling connected components and calculating the Euler number in a binary image are two fundamental processes for computer vision and pattern recognition. This paper presents an ingenious method for identifying a hole in a binary image in the first scan of connected-component labeling. Our algorithm can perform connected component labeling and Euler number computing simultaneously, and it can also calculate the connected component (object) number and the hole number efficiently. The additional cost for calculating the hole number is only $O(H)$, where H is the hole number in the image. Our algorithm can be implemented almost in the same way as a conventional equivalent-label-set-based connected-component labeling algorithm. We prove the correctness of our algorithm and use experimental results for various kinds of images to demonstrate the power of our algorithm.**

***Index Terms*— Computer vision, pattern recognition, image analysis, image feature, the Euler number, the hole number.**

I. Introduction

CONNECTED component labeling and Euler number computing are two indispensable fundamental processes for image analysis, pattern recognition, computer vision, and machine intelligence [1]–[3].

Labeling connected components in a binary image is assigning to all pixels of each object (connected component) a unique label. By labeling, we can differentiate different objects, and extract the features of each object, such as area, perimeter, circularity ratio, bounding box, geometric center, contour etc. Therefore, labeling is required in almost all image recognition systems such as fingerprint identification, character recognition, automated inspection, target recognition, face identification, medical image analysis, and computer-aided diagnosis.

Manuscript received June 8, 2014; revised October 14, 2014 and March 2, 2015; accepted April 13, 2015. Date of publication April 22, 2015; date of current version May 19, 2015. This work was supported in part by the National Natural Science Foundation of China under Grant 61471227, in part by the Ministry of Education, Science, Sports and Culture, Japan, through the Grant-in-Aid for Scientific Research (C) under Grant 26330200, and in part by the Key Science and Technology Program for Social Development of Shaanxi Province, China, under Grant 2014K11-02-01-13. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Dimitrios Tzovaras.

L. He is with Artificial Intelligence Institute, College of Electrical and Information Engineering, Shaanxi University of Science and Technology, Shaanxi, China, and the Graduate School of Information Science and Technology, Aichi Prefectural University, Nagakute 480-1198, Japan (e-mail: helifeng@ist.aichi-pu.ac.jp).

Y. Chao is with the Graduate School of Environment Management, Nagoya Sangyo University, Owariasahi 488-8711, Japan (e-mail: chao@nagoya-su.ac.jp).

Digital Object Identifier 10.1109/TIP.2015.2425540

Many algorithms have been proposed for connected-component labeling. There are mainly label-equivalence-based algorithms and label propagation algorithms. Label-equivalence-based labeling algorithms process an image by raster scans. In the first scan, they assign to each foreground pixel a provisional label, where all provisional labels assigned to a connected component are called *equivalent labels*. Then, in later processing, they resolve label equivalences to find a *representative label* for each set of equivalent labels. There are multi-scan algorithms [4]–[6], two-scan algorithms [7]–[14], and the one-and-a-half-scan algorithm [15]. On the other hand, label-propagation algorithms first search an unlabeled foreground pixel in the image. For such a foreground pixel, they assign it with a new label; then, in later processing, they assign the same label to all foreground pixels that are connected to the pixel. Although these algorithms usually employ the raster scan to search an unlabeled foreground pixel for labeling, they all access pixels of an image in an irregular way, depending on the shapes of connected components in the image. There are run-based algorithms [16]–[18] and contour-tracing algorithms [19], [20].

Because topologic properties remain invariant under translation, rotation, scaling, and any arbitrary rubber-sheet transformation, they are widely used for image retrieval, object recognition and shape matching [1], [21]. Among others, the Euler number of a binary image, which is defined as the number of objects (connected components) in the image minus the number of holes in those objects, is an important topologic feature of a binary image, and has been used for many practical applications [22]–[26].

There are also many algorithms proposed for calculating the Euler number of a binary image. A famous algorithm is based on counting certain 2×2 pixel patterns called bit-quads in the image [27], [28]. This algorithm has been used in the MATLAB image-processing tool box [29]. Dyer proposed an Euler-number computing algorithm for binary images represented by quadtrees [30]. Samet and Tamminen improved this algorithm by using a new staircase type of data structure to represent the processed blocks [31]. Chen and Yen proposed a parallel localized algorithm by using square graphs to calculate the Euler number of a given binary image on a square grid [22]. Chiavetta and Gesu used connectivity graph representation of a binary image for computing the Euler number [32]. Juan et al. developed a

method for computing the Euler number of a binary image from a skeleton of the image [33]. Moreover, some methods were presented for calculating the Euler number for the run-length representation of a binary image [34]–[36], and were improved for VLSI implementations [2], [37].

In many image analysis systems, especially in object recognition systems such as computer (robot) vision, both connected-component labeling and Euler number computing are possibly necessary simultaneously. Although the Euler number of a binary image is closely related to the connected component number and the hole number of the image, as introduced above, it is usually computed by a special algorithm irrelative to the object number and the hole number. Moreover, although there is almost no study on hole number computing and its applications, it is not doubt that the hole number of a binary image is important for image analysis and pattern recognition. By the Euler number of a binary image, we can only know the difference of the connected component number and the hole number, but neither the of object (connected component) number nor the hole number in the image. If we also know the hole number, we can further know the object number; thus we can further distinguish objects with the same Euler number.

Recently, He *et al.* proposed an algorithm for computing the connected component number, the hole number, and the Euler number by labeling connected components and holes simultaneously [38]. When both connected component labeling and Euler number computing are necessary, it is more efficient than conventional algorithms.

This paper proposes a very fast algorithm for performing connected-component labeling and Euler number computing simultaneously. In our algorithm, holes can be identified in the first scan of connected component labeling. The additional computing cost for calculating the hole number is $O(H)$, where H is the number of holes in the image. By our algorithm, the connected component number, the hole number and the Euler number of a binary image can be calculated efficiently. The experimental results for various kinds of images demonstrated that our algorithm is much more efficient than conventional algorithms for performing connected-component labeling and Euler number computing simultaneously.

The rest of this paper is organized as follows: in the next section, we review some related preliminaries. Section III introduces our method for computing the hole number and the Euler number, and gives the proof of the correctness of our method. We present the experimental results in Section IV to show the efficiency of our method, and make a discussion in Section V. Lastly, we give our concluding remarks in Section VI.

II. Preliminaries

For an $N \times M$-sized binary image, where $0 \leq x \leq N-1$ and $0 \leq y \leq M-1$, the pixel at the coordinate (x, y) in the image is denoted as $b(x, y)$. When it is clear from the context, we also use $b(x, y)$ to denote its value. The same as in Ref. [38], we assume that the value of foreground pixels is 1 and that of background pixels is 0, and all pixels on the border of an image are background pixels. Moreover, we consider eight-connectivity for connected components, and thus, four-connectivity for holes.

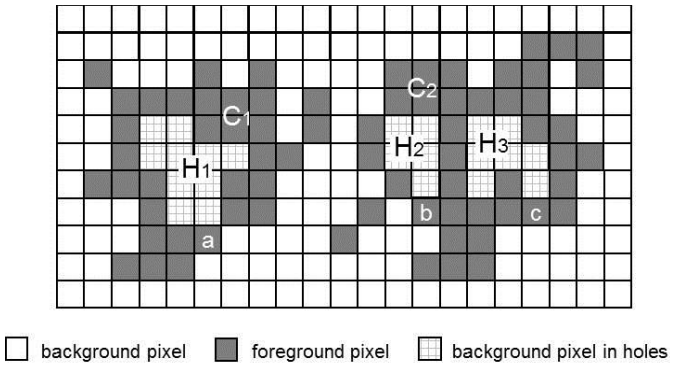

Fig. 1. An example of connected components and holes.

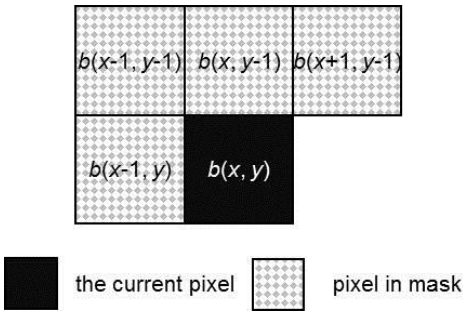

Fig. 2. Mask for the current pixel.

For two foreground (background) pixels, say, p and q, if there is a path $a_1, \ldots, a_n$ of foreground (background) pixels such that $a_1 = p$, $a_n = q$, and a_i and a_{i+1} are eight-connected (four-connected) neighbor for all $1 \leq i \leq n-1$, the two pixels are said to be *eight-connected* (*four-connected*). All eight-connected foreground pixels in a binary image consists of a *connected component*, and all four-connected background pixels such that all of them are enclosed by foreground pixels forms a *hole*. For example, there are two connected components, C_1 and C_2, and three holes, H_1, H_2 and H_3, in the image shown in Fig. 1, and thus, the Euler number of the image is $2 - 3 = -1$.

A. Connected-Component Labeling by Use of Equivalent Label Sets

In an equivalent-label-based labeling algorithm, we need to resolve label equivalences, i.e., find a representative label for provisional labels assigned to the same connected component. He *et al* proposed an efficient strategy for resolving label equivalences by use of *Equivalent Label Sets* [9]. For convenience, we call this strategy *ELS-strategy*.

By the ELS-strategy, at any point in the first scan, all provisional labels assigned to a connected component found so far are combined in a set, called an *equivalent label set*, where the smallest label is used as the *representative label* of the set as well as all provisional labels in the set. For convenience, we use $S(t)$ to denote the equivalent label set with t as the representative label.

As introduced in [10] and [38], the ELS-strategy can be implemented by use of three one-dimensional arrays, where an equivalent label set is represented as a list. The first array, R, is used for recording the representative label of each provisional label. The representative label of a provisional label t is represented by $R[t]$. The second array, *Next*, is used to denote the next label of a label in a list. $Next[k] = w$ means that the label next to k is w. Especially, $Next[s] = -1$ indicates that s is the last label in the list. The last array, *Last*, is used to indicate the last label of a list. $Last[u] = p$ means that the last label in the list $S(u)$ is p.

When an object pixel is assigned a new label l, a new equivalent label set can be established by executing the following procedure *newset*(l) as follows:

```
Procedure newset(l)
    R[l] ← l
    Next[l] ← −1
    Last[l] ← l
End Procedure
```

On the other hand, if two provisional labels m and n are found to be equivalent label, then all provisional labels in $S(u)$ and $S(v)$ are equivalent labels, where $u = R[m]$ and $v = R[n]$; thus, the two equivalent label sets should be combined. The pseudo code for combining $S(u)$ and $S(v)$, similar in Ref. [38], denoted as *combine*(u, v), is shown as follows:

```
Procedure combine(u, v)
    If u < v   %%%S(u)←S(u)∪S(v)
        k ← v
        While k ≠ −1
            R[k] ← u
            k ← Next[k]
        End While
        Next[Last[u]] ← v
        Last[u] ← Last[v]
    Else If u > v   %%%S(v)←S(u)∪S(v)
        k ← u
        While k ≠ −1
            R[k] ← v
            k ← Next[k]
        End While
        Next[Last[v]] ← u
        Last[v] ← Last[u]
    End If
End Procedure
```

The ELS-strategy has been used in many labeling algorithms. First of all, it is used in a run-based labeling algorithm [10], where a run is a block of contiguous object pixels in a row. Because all pixels in a run certainly belong to the same connected component, thus, they can be assigned to the same label. Moreover, run data are recorded for resolving label equivalences among runs. This algorithm is efficient for images with large average length of runs, and is improved in Ref. [15] by considering a run as a super pixel. Instead assigning each object pixel a provisional label, the improved algorithm assigns each run a provisional label. Moreover, it only relabels object pixels in the second scan by use of recorded run data.

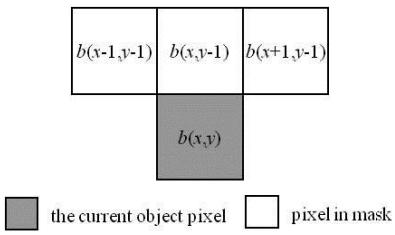

Fig. 3. Mask used in the HCS algorithm.

The ELS-strategy is also used in a pixel-based labeling algorithm [11]. This algorithm uses the mask shown in Fig. 2 for processing each of object pixels. In order to reduce the number of operations, it proposed an optimal order, i.e., $b(x, y-1) \rightarrow b(x-1, y) \rightarrow b(x+1, y-1) \rightarrow b(x-1, y-1)$, for checking pixels in the mask. This algorithm is improved in Ref. [12] by processing object pixels following another object pixel and those following a background pixel in a different way to avoid checking $b(x-1, y)$, thus, leads to reduce the number of pixel accesses. It is further improved in Ref. [14] by scanning image lines alternate lines and processing pixels two by two to reduce the number of pixel accesses. Moreover, the information obtained during processing the current two object pixels are used for processing the next two object pixels to reduce the number of pixel accesses further.

Moreover, the ELS-strategy is used in a block-based labeling algorithm [13], which resolves connectivity among 2×2 blocks. Because all foreground pixels in a 2×2 block are certainly 8-connected, they belong to the same connected component and thus will be assigned the same label. Therefore, instead of assigning to each foreground pixel a provisional label, this algorithm assigns to each foreground-pixel-contained block a provisional label. In other words, it considers a block as a super pixel. In the second scan, it assigns to all foreground pixels in each block the representative label of the block.

Because the algorithm proposed in Ref. [12] by He, Chao, and Suzuki is still one of the fastest and simplest connected component labeling algorithm to be implemented in our current CPUs, we introduce the algorithm in detail. For convenience, we call this labeling algorithm *HCS* algorithm.

In the first scan, the HCS algorithm does nothing for background pixels and uses the mask shown in Fig. 3 to process object pixels. For an object pixel $b(x, y)$ following a back ground pixel (one of Fig. 4(1) - (8)), it checks whether there is a label in the mask. If there is no label (Fig. 4(1)), then assign the pixel a new label l, and establish equivalent label set $S(l)$. Otherwise, assign any label in the mask to $b(x, y)$. Moreover, if there are two object pixels become connected by $b(x, y)$ (Fig. 4 (6)), combine the two corresponding equivalent label sets.

On the other hand, for an object pixel $b(x, y)$ following another object pixel (one of Fig. 4(9) - (16)), it assigns the label assigned $b(x-1, y)$ to $b(x, y)$. Moreover, if there are two object pixels become connected by $b(x, y)$ (Fig. 4 (13) or (14)), combine the two corresponding equivalent label sets.

Notice that u is equal to v means that $S(u)$ and $S(v)$ is the same equivalent label set; thus, nothing needs to be done. In this way, as soon as the first scan finishes, each of foreground pixels will be assigned a provisional label, and all provisional labels assigned to the same connected component will be combined in an equivalent label set with the same representative label. Then, in the second scan, by replacing the provisional label of each foreground pixel by its representative label, we can complete labeling.

The pseudo code of the HCS algorithm can be shown as follows, where l is the variable for provisional labels.

```
l←0
For y from 1 to M−1   %% first scan
  For x from 1 to N−1
    If b(x, y) > 0
      If b(x, y−1) > 0
        b(x, y) ← b(x, y−1)
      Else If b(x+1, y−1)>0
        b(x, y) ← b(x+1, y−1)
        If b(x−1, y−1) > 0
          combine(R[b(x−1, y−1)], R[b(x+1, y−1)])
        End If
      Else If b(x−1, y−1) > 0
        b(x, y) ← b(x−1, y−1)
      Else
        l ← l + 1
        b(x, y) ← l
        newset(l)
      End If
      x ← x + 1
      While b(x, y) > 0
        b(x, y) ←b(x−1, y)
        If b(x, y−1) < 1 and b(x+1, y−1) > 0
          combine(R[b(x−1, y)], R[b(x+1, y−1)])
        End If
        x ← x + 1
      End While
    End If
    x ← x + 1
  End For
  y ← y + 1
End For
R[0] ← 0
For y from 1 to M−1  %% second scan
  For x from 1 to N−1
    b(x, y) = R[b(x, y)]
    x ← x + 1
  End For
  y ← y + 1
End For
```

Moreover, according to the above analysis, when the first scan finishes, the number of equivalent label sets will be equal to the number of connected components in the image. For each equivalent label set $S(r)$, only the representative label r satisfies the condition $R[r] = r$. Therefore, we can calculate the number of connected components, C, as follows:

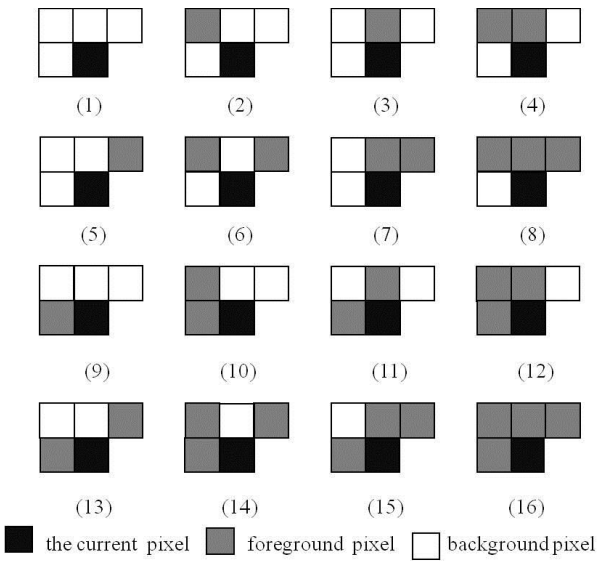

Fig. 4. The 16 cases in the mask for the current foreground pixel.

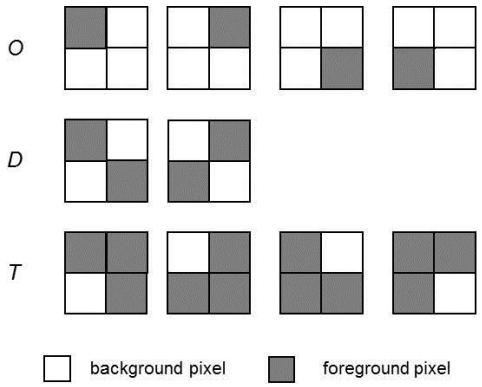

Fig. 5. Bit-quads for Euler number computing.

```
C←0
For i from 1 to l
  If R[i] = i
    C ← C + 1
  End If
End For
```

B. Euler Number Computing by Counting Bit-Quads

Gray proposed a bit-quad-based method for calculating the Euler number in a binary image [27], [28]. Let N_O, N_D and N_T be the numbers of patterns O (i.e., one-object-pixel bit quads), D (i.e., diagonal-object-pixel bit quads), and T (i.e., three-object-pixel bit quads) shown in Fig. 5, respectively. Then, by this method, the Euler number, E, for eight-connectivity can be calculated by the following formula:

$$E = (N_O - N_T - 2N_D)/4.$$

For convenience, the same as in Ref. [38], we denote this algorithm as *ML algorithm*. A great advantage of the ML algorithm is fully parallelized. The pseudo code for an efficient implementation is shown as follows, where similar in the algorithm proposed in [14], the information obtained during processing the current bit-quad is used for

processing the next bit-quad in order to reduce the number of pixel access, and C_1, C_2, C_3 and C_4 are the bit-quads $\begin{bmatrix} 0 & b(x, y-1) \\ 0 & b(x, y) \end{bmatrix}$, $\begin{bmatrix} 0 & b(x, y-1) \\ 1 & b(x, y) \end{bmatrix}$, $\begin{bmatrix} 1 & b(x, y-1) \\ 0 & b(x, y) \end{bmatrix}$ and $\begin{bmatrix} 1 & b(x, y-1) \\ 1 & b(x, y) \end{bmatrix}$, respectively.

```
Define Increment&Check(x)
    x ← x + 1
    If x ≥ M
        goto Endx
    End If
End Define
x ← 0
y ← 1
While y < N
    While x < M
        C1:
        Increment&Check(x)
        If b(x, y) > 0
            If b(x, y–1) = 0   //
                N_O ← N_O + 1
                C2:
                Increment&Check(x)
                If b(x, y) > 0
                    If b(x, y–1) > 0
                        N_T ← N_T + 1
                        goto C4
                    Else
                        goto C2
                    End If
                Else    %% b(x, y) = 0
                    If b(x, y–1) > 0
                        N_D ← N_D + 1
                        goto C3
                    Else
                        N_O ← N_O + 1
                        goto C1
                    End If
                End If
            Else    %%b(x, y–1) > 0
                C4:
                Increment&Check(x)
                If b(x, y) > 0
                    If b(x, y–1) = 0
                        N_T ← N_T + 1
                        goto C2
                    Else
                        goto C4
                    End If
                Else    %% b(x, y) = 0
                    If b(x, y–1) > 0
                        N_T ← N_T + 1
                        goto C3
                    Else
                        goto C1
                    End If
                End If
            End If
        Else    %% b(x, y) = 0
            If b(x, y–1) > 0   //
                N_O ← N_O + 1
                C3:
                Increment&Check(x)
                If b(x, y) > 0
                    If b(x, y–1) = 0
                        N_D ← N_D + 1
                        goto C2
                    Else
                        N_T ← N_T + 1
                        goto C4
                    End If
                Else    %% b(x, y) = 0
                    If b(x, y–1) = 0
                        N_O ← N_O + 1
                        goto C1
                    Else
                        goto C3
                    End If
                End If
            Else    %% b(x, y–1) = 0
                goto C1
            End If
        End If
    End While
    Endx:
    y ← y + 1
End While
E = (N_O − N_T –2N_D)/4
```

C. Calculating the Euler Number by Labeling Connected Components and Holes

Recently, He *et al* proposed an algorithm for calculating the Euler number of a binary image by use of the ELS-strategy to label connected components and holes simultaneously with the same data structures [38]. This algorithm uses provisional labels from 1 to W for labeling holes and provisional labels larger than W for labeling connected components. By this algorithm, in the first scan, foreground pixels are processed in the same way as in the HCS algorithm, and background pixels are processed in a similar way but 4-connectivity is considered. Similar as in the HCS algorithm, when the first scan finishes, all equivalent labels assigned to a connected component or a hole will be combined in an equivalent label set with the same representative label. Thus, the number of the connected components, C, and that of the hole, H, in the image can be calculated according to the numbers of the equivalent label sets for connected components and holes, respectively. Then, the Euler number of the image can be calculated. For convenience, we denote this algorithm as the *HCS_E algorithm*. More details about the HCS_E algorithm can be found in Ref. [38].

III. Proposed Algorithm

As introduced in the above section, the number of holes can be calculated by labeling holes. In fact, we can find the number of holes more efficiently by only labeling connected

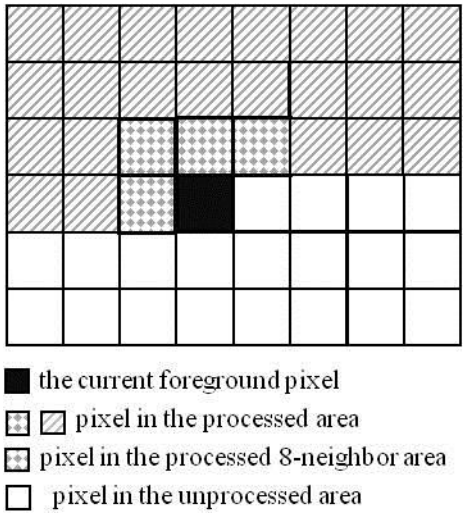

Fig. 6. Processed area in raster scan.

components (i.e., without labeling holes). As mentioned in Introduction, a hole is enclosed by foreground pixels of a connected component. When we observe an image in raster scan, we can find that for each hole, there is a foreground pixel such that it finishes the closure of foreground pixels for the hole. For example, in the binary image shown in Fig. 1, the pixels a, b, and c are such a pixel for holes H_1, H_2, and H_3, respectively. Such pixels are called as *closing pixels*, whose definition is given as follows.

Definition 1: Let $b(x, y)$ be the current foreground pixel when processing a binary image in raster scan. If two independent branches of a connected component in the processed area are connected by $b(x, y)$, $b(x, y)$ is said to be a closing pixel.

It is intuitive that in a binary image, each closing point corresponding to a hole, and vice versa. Therefore, the number of the holes in a binary image is equal to the number of closing pixels in the image. We show that closing pixels can be found in the first scan of the HCS algorithm.

Lemma 1: When processing $b(x, y)$ in the first scan by the HCS algorithm, if and only if the procedure *combine*(u, v) is called in the case where u and v are equal, $b(x, y)$ is a closing pixel.

Proof: Let $b(x, y)$ be the current pixel being processed in the HCS algorithm. We prove Lemma 1 by induction on the number of pixels that have been processed. Thus, we only need to consider the processed area together with $b(x, y)$, as shown in Fig. 6. Moreover, notice that as introduced in Section II B, for processing a foreground pixel, only when there are two independent foreground pixel blocks in the mask, i.e., when the configuration of the mask is one of Fig. 5 (6), (13), (14), the HCS algorithm will call the procedure *combine*(u, v), where u and v are the representative labels of the two independent foreground pixel blocks, respectively.

Basic Case: $b(x, y)$ is the only pixel in the processed area. Obviously, no matter whether $b(x, y)$ is a foreground pixel or a background pixel, it should not be a closing pixel. On the other hand, for processing this pixel, the HCS algorithm will not call *combine*(u, v). Therefore, Lemma 1 is true in this case.

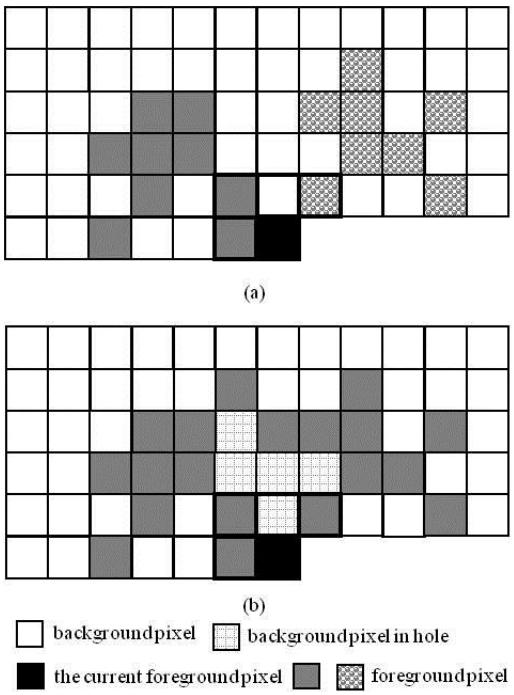

Fig. 7. Examples of two foreground pixel blocks in the mask in the processed 8-neigbhor area that belong to: (a) two different connected components, (b) the same connected component.

Induction: Suppose that Lemma 1 is true after the nth pixel is added into the image. We show that it is also true after the $(n+1)$th pixel $b(x, y)$ is processed. We consider the following four cases:

(1) $b(x, y)$ is a background pixel. Obviously, it is not a closing pixel. On the other hand, by the HCS algorithm, no operation will be performed for background pixels, thus, *combine*(u, v) will not be called. Therefore, Lemma 1 is true in this case.

(2) $b(x, y)$ is a foreground pixel in the cases where there is no foreground pixel in the processed 8-neighbor area (Fig. 4 (1)). It is obvious that $b(x, y)$ does not connect any processed foreground pixel, $b(x, y)$ is not a closing pixel. On the other hand, in this case, as mentioned above, for processing $b(x, y)$, the HCS algorithm will not call *combine*(u, v). Therefore, Lemma 1 is true in this case.

(3) $b(x, y)$ is a foreground pixel in the cases where there is only one foreground pixel block in the processed 8-neighbor area (Fig. 4 (2)-(5), (7)-(12), (15), (16)). We can see that $b(x, y)$ does not connect two independent foreground pixel blocks in the processed area, $b(x, y)$ is not a closing pixel. On the other hand, as mentioned above, in such a case, *combine*(u, v) will not be called by the HCS algorithm. Therefore, Lemma 1 is true in these cases.

(4) $b(x, y)$ is a foreground pixel in the cases where there are two independent foreground pixel blocks in the processed 8-neighbor area (Fig. 4 (6), (13), (14)). As mentioned above, in such a case, for processing $b(x, y)$, the HCS algorithm will call *combine*(u, v), where u and v are the representative labels of the two independent foreground pixel blocks, respectively. If the two independent foreground pixel blocks belong to different connected components in the processed area, for

example, as shown in Fig. 7 (a), $b(x, y)$ will not be a closing pixel. By the HCS algorithm, u and v will be different; thus, *combine*(u, v) will be called such that u and v are not equal. Otherwise, if the two independent foreground pixel blocks belong to the same connected component in the processed area, for example, as shown in Fig. 7 (b), they are independent branches of a connected component. Thus, $b(x, y)$ will be a closing pixel. On the other hand, as introduced in Section II B, by the HCS algorithm, u and v will be equal. Thus, *combine*(u, v) will be called such that u and v are equal. Therefore, Lemma 1 is also true in these cases.

Therefore, Lemma 1 is true after the $(n+1)$th pixel is added. Thus, Lemma 1 is proved. □

Theorem 1: The number of holes in a binary image is equal to the number of the times that *combine*(u, v) is called such that u and v are equal in the HCS labeling algorithm.

According to Lemma 1, the proof of Theorem 1 is trivial.

By Theorem 1, the hole number of a binary image, H, can be calculated in the HCS algorithm by modifying *combine*(u, v) to *combine*$^*(u, v)$ as follows, where H is initialized to 0:

```
Procedure combine*(u, v)
    If u<v    %%S(u)←S(u)∪S(v)
        k ← v
        While k ≠ −1
            R[k] ← u
            k ← Next[k]
        End While
        Next[Last[u]] ← v
        Last[u] ← Last[v]
    Else If u > v    %%S(v)←S(u)∪S(v)
        k ← u
        While k ≠ −1
            R[k] ← v
            k ← Next[k]
        End While
        Next[Last[v]] ← u
        Last[v] ← Last[u]
    Else %% u and v are equal
        H ← H + 1
    End If
End Procedure
```

In this way, the pseudo code of our proposed algorithm are exactly the same as that of the HCS algorithm except replacing *combine*(u, v) by *combine*$^*(u, v)$. When the first scan of our proposed algorithm finishes, we can obtain the hole number H. Then after computing the connected component number C, we can calculate the Euler number. Notice that by executing the second scan, we can complete connected component labeling.

IV. Experimental Results

We mainly compared our algorithm with the ML algorithm and the HCS_E algorithm proposed in [38]. Our test consisted of two parts: (1) we compared the three algorithms for only calculating the Euler number; (2) we compared the HCS_E algorithm and our algorithm for performing both labeling connected components and calculating the Euler number. Notice that, in the case (1), the second scan in the HCS_E algorithm and our algorithm is unnecessary. For convenience, we use the Ours1 algorithm and the Ours2 algorithm to denote our algorithm, the HCS_E1 algorithm and the HCS_E2 algorithm for the HCS_E algorithm in the cases (1) and (2), respectively.

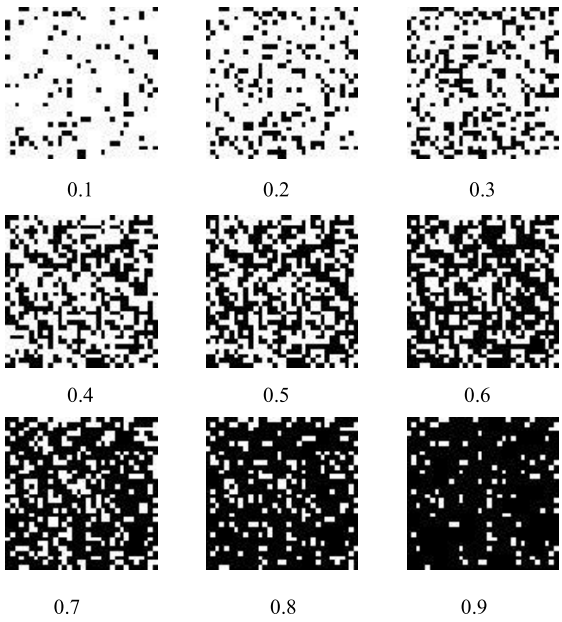

Fig. 8. Noise images with various densities.

All algorithms used in our test were implemented in C language and compiled by the GNU C complier (version 4.2.3) with the option −O3 by use of one core. The computer used for test is a PC-based workstation (Intel Pentium D 930 3.0GHz + 3.0GHz CPUs, 2GB Memory, Mandriva Linux OS). All executing times presented in this section were obtained by averaging of the execution time for 5000 runs. The connected component number, the hole number and/or the Euler number for any image calculated by any algorithm are/is exactly the same. In our test, two kinds of images were used:

(1) Artificial image set. There are specialized-pattern (stair-like, spiral-like, saw-tooth-like, checker-board-like, and honeycomb-like pattern) images [4] and noise images. The same as in Ref. [38], noise images with five sizes (32×32, 64×64, 128×128, 256×256, and 512×512 pixels) are used for test. For each size, we threshold a same size image containing uniform noise with random values from 0 to 1000 with 41 different threshold values from 0 to 1000 in steps of 25. Thus, 41 noise images whose densities (i.e., the percentages of foreground pixels) are from 0.05% to 99.22% will be generated. Because foreground pixels in such noise images have complex connectivity, we can make severe evaluations of algorithms on these images. Nine noise images with densities 0.1, 0.2, …, and 0.9, respectively, are shown in Fig. 8.

(2) The realistic image set. 50 natural images including landscape, aerial, fingerprint, portrait, still-life, snapshot, and text images are obtained from the Standard Image Database (SIDBA) developed by the University of Tokyo [39]

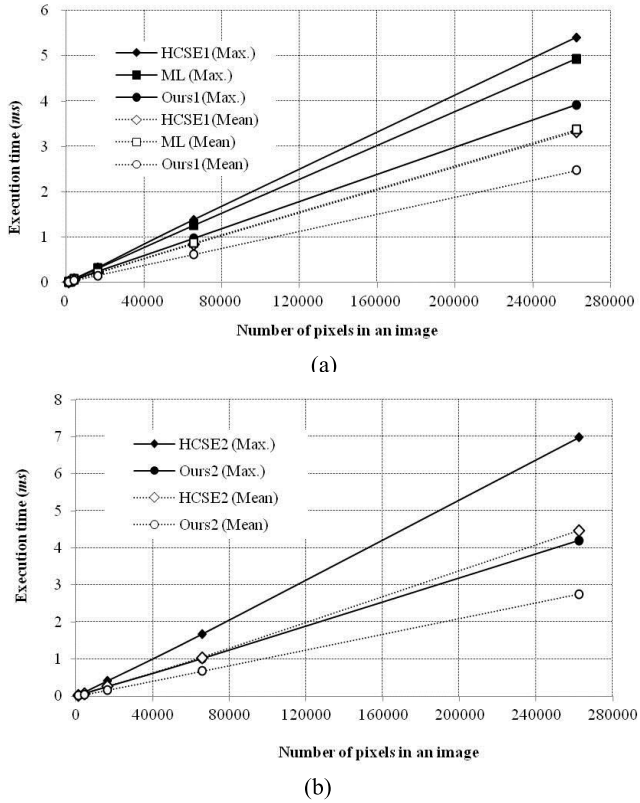

Fig. 9. Execution time versus the number of pixels in an image: (a) for Euler number computing, (b) for both labeling connected components and calculating the Euler number.

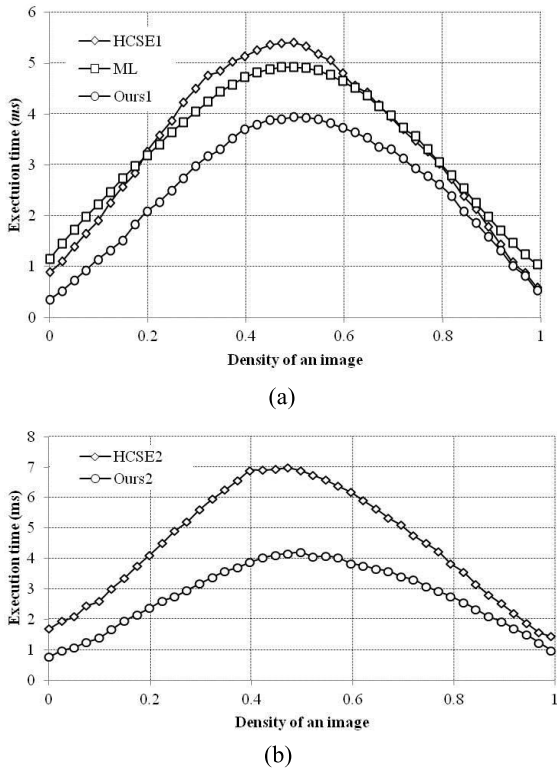

Fig. 10. Execution time versus the number of pixels in an image: (a) for Euler number computing, (b) for both labeling connected components and calculating the Euler number.

and the image database of the University of Southern California [40], respectively. In addition, seven texture images were downloaded from the Columbia-Utrecht Reflectance and Texture Database [41], and 25 medical images were obtained from a medical image database of The University of Chicago. All of these images were 512×512 pixels in size, and were binarizied by means of Otsu's method [42].

A. Execution Time Versus the Size of an Image

Because noise images with different size have good similarities, we use all noise images to test the linearity of the execution time of various algorithms versus image sizes. The results for only calculating the Euler number are shown in Fig. 9 (a), and those for performing both labeling connected components and calculating the Euler number are shown in Fig. 9 (b).

As shown in Fig. 8, we can see that, either for calculating the Euler number alone or performing Euler number computing and connected-component labeling simultaneously, both the maximum execution times and the average execution times for various algorithms have the ideal linear characteristics versus image sizes. In either case, the execution time of our algorithms is much smaller than that of either of the other two algorithms.

B. Execution Time Versus the Density of an Image

We used 512 × 512-sized noise images for testing the execution time versus the density of an image. The experimental results for only calculating the Euler number are shown in Fig. 10 (a), and those for both labeling connected components and calculating the Euler number are shown in Fig. 10 (b).

As we can see, when we only calculate the Euler number, the performance of the HCS_E1 algorithm is better than that of the ML algorithm for low-density and high-density images, but worst for middle-densities images. On the other hand, for all images of all densities, the Ours1 algorithm is the most efficient. When performing connected component labeling and Euler number computing simultaneously, the performance of the Ours2 algorithm is much better than the HCS_E2 algorithm for images of all densities.

C. Comparisons in Terms of the Maximum, Mean, and Minimum Execution Times

All images in the realistic image set and the specialized-shape-pattern artificial images were used for this test. The results for only calculating the Euler number are shown in Table 1, and those for performing both labeling connected components and calculating the Euler number are shown in Table 2. We can find that, either for only calculating the Euler number or for performing both Euler number computing and connected-component labeling, our algorithm is much better than the others in comparison. In fact, for all images used in our test, the Ours1 algorithm is faster than either the HCS_E1 algorithm or the ML algorithm, and the Ours2 algorithm is much faster than the HCS_E2 algorithm. The execution time (*ms*), the density D, the number of connected components C, the number of holes H, and the Euler number E for the six images selected in Ref. [38] are illus-

TABLE I
MAXIMUM, MEAN, AND MINIMUM EXECUTION TIMES (ms) ON VARIOUS TYPES OF IMAGES FOR ONLY CALCULATING THE EULER NUMBER

Image Type		HCS_E1	ML	Ours1
Natural	Max.	2.48	2.65	1.79
	Mean	1.35	1.72	0.94
	Min.	0.79	1.18	0.27
Medical	Max.	1.52	1.82	1.01
	Mean	1.17	1.47	0.76
	Min.	0.84	1.29	0.61
Textural	Max.	2.10	2.21	1.49
	Mean	1.41	1.64	1.00
	Min.	0.67	1.05	0.58
Artificial	Max.	2.23	1.16	1.02
	Mean	1.83	1.14	0.95
	Min.	1.04	1.12	0.80

TABLE II
MAXIMUM, MEAN, AND MINIMUM EXECUTION TIMES (ms) ON VARIOUS TYPES OF IMAGES FOR PERFORMING BOTH LABELING CONNECTED COMPONENTS AND CALCULATING THE EULER NUMBER

Image Type		HCS_E2	Ours2
Natural	Max.	3.46	2.34
	Mean	2.12	1.37
	Min.	1.59	0.81
Medical	Max.	2.23	1.37
	Mean	1.87	1.12
	Min.	1.73	0.99
Textural	Max.	2.85	1.92
	Mean	2.17	1.43
	Min.	1.48	0.89
Artificial	Max.	3.13	1.40
	Mean	2.53	1.29
	Min.	1.32	1.08

trated in Fig. 11, where the foreground pixels are displayed in black.

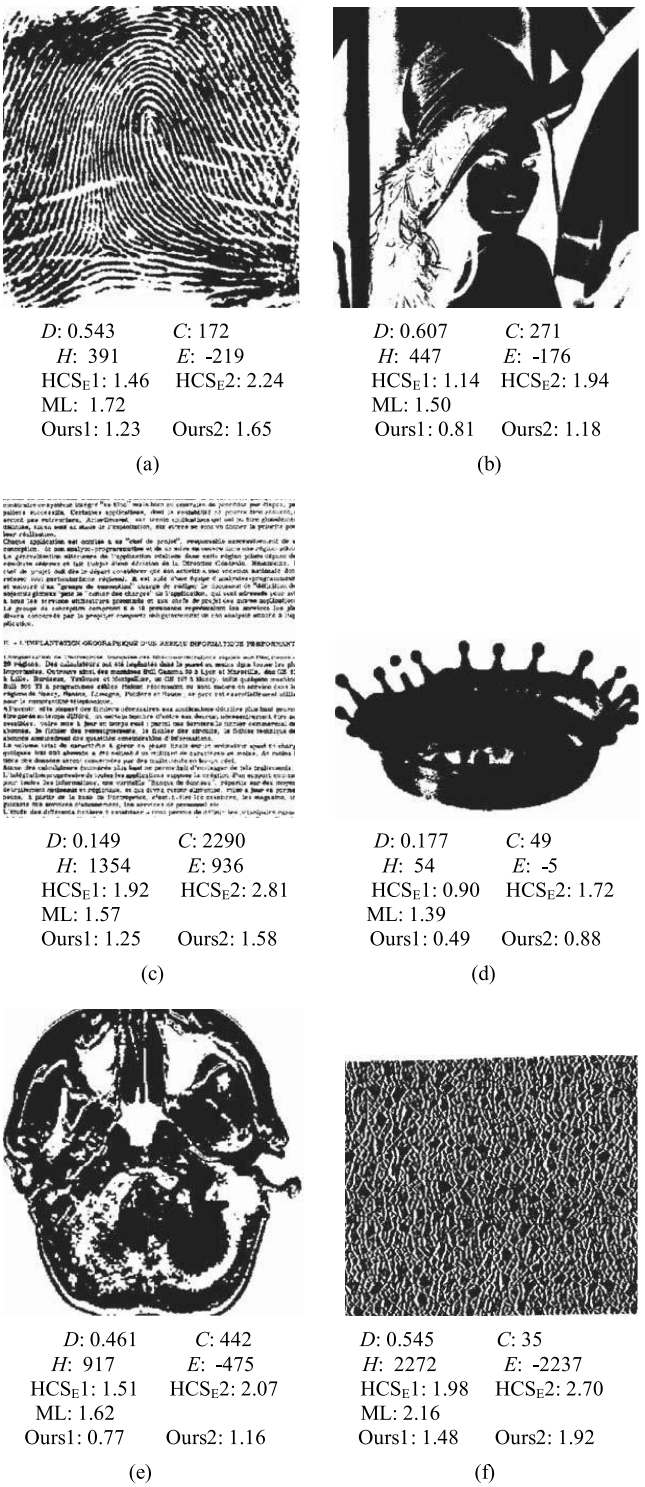

Fig. 11. Execution time (ms), the density, the number of connected components C, the number of holes H, the Euler number E for the selected six images: (a) a fingerprint image; (b) a portrait image; (c) a text image; (d) a snapshot image; (e) a medical image; (f) a texture image.

V. DISCUSSION

A. The Additional Computation Cost of Our Algorithm for Calculating the Connected Component Number, the Hole Number and the Euler Number Over the HCS Algorithm

For calculating the connected component number, the hole number and the Euler number, our algorithm needs to do the following three additional work than the HCS labeling algorithm: (1) calculating the connected component number; (2) calculating the hole number; (3) calculating the Euler number.

According to the procedure for counting the number of connected components introduced in Section II B, the corresponding computation cost is $O(P)$, where P is the number of provisional labels assigned by the HCS algorithm. By the

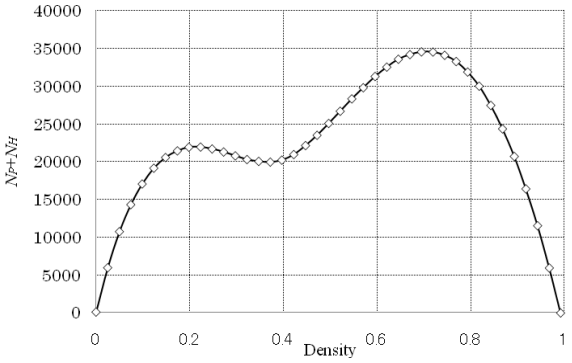

Fig. 12. $P + H$ versus the density for 512 × 512-pixel sized noise images.

procedure of *combine*$^*(u, v)$ introduced in Section III, the cost for computing the hole number is $O(H)$, where H is the number of holes. Therefore, the order of the additional computation for calculating the Euler number in our algorithm versus the HCS algorithm should be $O(P+H)$. Because, for an $N\times M$-sized binary image, the maximum number of provisional labels assigned by the HCS algorithm is $N\times M/4$ and the maximum number of holes of a binary image is $N\times M/2$, the order of the additional computation should be $O(N\times M)$ in the worst case.

In our experimental test, we found that, for processing an image, the execution time of the Ours2 algorithm is almost the same as that of the HCS algorithm. In other words, the additional costs for calculating the number of connected components, the number of holes and the Euler number are very small. For example, for any noise image, the additional execution time in our test is so small (less than 1/100 *ms*) that we could not even measure it correctly. Fig. 12 shows the value of $P + H$ versus the density for 512×512-pixel-sized noise images.

B. Comparison With the ML Algorithm

The ML algorithm can only calculate the Euler number of a binary image, but neither the connected component number nor the hole number in the image, which will be useful for image analysis and pattern recognition in many cases. In cases where connected-component labeling, the connected component number and/or the hole number are also necessary, an indispensible labeling algorithm will be needed. Thus, the whole processing time will increase more than twice.

The ML algorithm can only calculate the Euler number of a binary image, but neither the connected component number nor the hole number in the image, which will be useful for image analysis and pattern recognition in many cases. In cases where connected-component labeling, the connected component number and/or the hole number are also necessary, an indispensible labeling algorithm will be needed. Thus, the whole processing time will increase more than twice.

In comparison, except for calculating the Euler number in an image, the Ours1 algorithm can also give both the connected component number and the hole number. In the cases where the connected component number and/or the hole number are/is also necessary, for any image used in our experimental test, the execution time of the Ours2 algorithm is almost the same as that of the HCS algorithm. In other words, in such cases, the connected component number, the hole number and the Euler number can be computed in almost no cost.

However, the ML algorithm is much simpler and fully parallelized. If only Euler number computation is necessary and a parallel computer is available, the ML algorithm should be selected.

C. Comparison With the HCS_E Algorithm

Our algorithm also calculates the Euler number by use of the connected component number and the hole number. However, in contrast to the HCS_E algorithm, which calculates the hole number by labeling holes, our algorithm does that via labeling connected components: we only need to count the times that two branches of the same connected component undergo closure during the labeling. Thus, the performance of our algorithm is much better than that of the HCS_E-related algorithm for the cases of either only calculating the Euler number or together with labeling connected components.

On the other hand, the HCS_E algorithm can be extended easily to calculate various features of connected components and holes [38]. Our algorithm can also be extended to calculate features of connected components exactly in the same way as in the HCS_E algorithm, but it is not easy to extend our algorithm to calculate features of holes. Therefore, in the cases where the features of holes are needed, the HCS_E algorithm should be selected.

VI. Concluding Remarks

In this paper, we proposed a fast algorithm for performing connected component labeling and Euler number computing simultaneously. The advantages of our algorithm are: (1) its principle is simple; (2) it is easy to implement (less than 50 lines in the C language); (3) except for the Euler number, it can also calculate the connected component number and the hole number; (4) it is much more efficient than the conventional algorithms for various types of images; (5) it is suitable for hardware implementation (because it processes an image in a sequential order).

For future work, we plan to implement our algorithm in hardware [43]–[45], to extend it to include three-dimensional images [20], [46], [47], and to develop algorithms for parallel architectures [48].

Acknowledgments

The authors would like to thank the anonymous referees for their valuable comments that improved this paper greatly. They are grateful to the associate editor Dr. Dimitrios Tzovaras for his kind cooperation.

References

[1] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*. Reading, MA, USA: Addison-Wesley, 1993.
[2] A. Rosenfeld and A. C. Kak, *Digital Picture Processing*, vol. 2, 2nd ed. San Diego, CA, USA: Academic, 1982.
[3] R. Szeliski, *Computer Vision: Algorithms and Applications*. New York, NY, USA: Springer-Verlag, 2011.
[4] K. Suzuki, I. Horiba, and N. Sugie, "Linear-time connected-component labeling based on sequential local operations," *Comput. Vis. Image Understand.*, vol. 89, no. 1, pp. 1–23, Jan. 2003.
[5] A. Rosenfeld and J. L. Pfaltz, "Sequential operations in digital picture processing," *J. ACM*, vol. 13, no. 4, pp. 471–494, Oct. 1966.
[6] A. Rosenfeld, "Connectivity in digital pictures," *J. ACM*, vol. 17, no. 1, pp. 146–160, Jan. 1970.

[7] R. Lumia, L. Shapiro, and O. Zuniga, "A new connected components algorithm for virtual memory computers," *Comput. Vis., Graph., Image Process.*, vol. 22, no. 2, pp. 287–300, May 1983.
[8] K. Wu, E. Otoo, and K. Suzuki, "Optimizing two-pass connected-component labeling algorithms," *Pattern Anal. Appl.*, vol. 12, no. 2, pp. 117–135, Jun. 2009.
[9] L. He, Y. Chao, and K. Suzuki, "A linear-time two-scan labeling algorithm," in *Proc. IEEE Int. Conf. Image Process. (ICIP)*, San Antonio, TX, USA, Sep. 2007, pp. V-241–V-244.
[10] L. He, Y. Chao, and K. Suzuki, "A run-based two-scan labeling algorithm," *IEEE Trans. Image Process.*, vol. 17, no. 5, pp. 749–756, May 2008.
[11] L. He, Y. Chao, K. Suzuki, and K. Wu, "Fast connected-component labeling," *Pattern Recognit.*, vol. 42, no. 9, pp. 1977–1987, Sep. 2009.
[12] L. He, Y. Chao, and K. Suzuki, "An efficient first-scan method for label-equivalence-based labeling algorithms," *Pattern Recognit. Lett.*, vol. 31, no. 1, pp. 28–35, Jan. 2010.
[13] C. Grana, D. Borghesani, and R. Cucchiara, "Optimized block-based connected components labeling with decision trees," *IEEE Trans. Image Process.*, vol. 19, no. 6, pp. 1596–1609, Jun. 2010.
[14] L. He, X. Zhao, Y. Chao, and K. Suzuki, "Configuration-transition-based connected-component labeling," *IEEE Trans. Image Process.*, vol. 23, no. 2, pp. 943–951, Feb. 2014.
[15] L. He, Y. Chao, and K. Suzuki, "A run-based one-and-a-half-scan connected-component labeling algorithm," *Int. J. Pattern Recognit. Artif. Intell.*, vol. 24, no. 4, pp. 557–579, Jun. 2010.
[16] D. H. Ballard, *Computer Vision*. Englewood Cliffs, NJ, USA: Prentice-Hall, 1982.
[17] K. Shoji and J. Miyamichi, "Connected component labeling in binary images by run-based contour tracing," (in Japanese), *Trans. Inst. Electron., Inf. Commun. Eng.*, vol. J83-D-II, no. 4, pp. 1131–1139, Apr. 2000.
[18] Y. Shima, T. Murakami, M. Koga, H. Yashiro, and H. Fujisawa, "A high-speed algorithm for propagation-type labeling based on block sorting of runs in binary images," in *Proc. 10th Int. Conf. Pattern Recognit.*, Jun. 1990, pp. 655–658.
[19] F. Chang, C.-J. Chen, and C.-J. Lu, "A linear-time component-labeling algorithm using contour tracing technique," *Comput. Vis. Image Understand.*, vol. 93, no. 2, pp. 206–220, Feb. 2004.
[20] Q. Hu, G. Qian, and W. L. Nowinski, "Fast connected-component labelling in three-dimensional binary images based on iterative recursion," *Comput. Vis. Image Understand.*, vol. 99, no. 3, pp. 414–434, Sep. 2005.
[21] E. C. Greanias, P. F. Meagher, R. J. Norman, and P. Essinger, "The recognition of handwritten numerals by contour analysis," *IBM J. Res. Develop.*, vol. 7, no. 1, pp. 14–21, Jan. 1963.
[22] M.-H. Chen and P.-F. Yan, "A fast algorithm to calculate the Euler number for binary images," *Pattern Recognit. Lett.*, vol. 8, no. 2, pp. 295–297, Dec. 1988.
[23] S. N. Srihari, "Document image understanding," in *Proc. ACM/IEEE Joint Fall Comput. Conf.*, Dallas, TX, USA, Nov. 1986, pp. 87–96.
[24] P. L. Rosin and T. Ellis, "Image difference threshold strategies and shadow detection," in *Proc. Brit. Mach. Vis. Conf.*, 1995, pp. 347–356.
[25] S. K. Nayar and R. M. Bolle, "Reflectance based object recognition," *Int. J. Comput. Vis.*, vol. 17, no. 3, pp. 219–240, Mar. 1996.
[26] B. K. P. Horn, *Robot Vision*. New York, NY, USA: McGraw-Hill, 1986, pp. 73–77.
[27] S. B. Gray, "Local properties of binary images in two dimensions," *IEEE Trans. Comput.*, vol. C-20, no. 5, pp. 551–561, May 1971.
[28] W. K. Pratt, *Digital Image Processing*. New York, NY, USA: Wiley, 1991, p. 633.
[29] [Online]. Available: http://www.mathworks.com/access/helpdesk/help/toolbox/images/bweuler.html, accessed May 2014.
[30] C. R. Dyer, "Computing the Euler number of an image from its quadtree," *Comput. Graph. Image Process.*, vol. 13, no. 3, pp. 270–276, Jul. 1980.
[31] H. Samet and H. Tamminen, "Computing geometric properties of images represented by linear quadtrees," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. PAMI-7, no. 2, pp. 229–240, Mar. 1985.
[32] F. Chiavetta, "Parallel computation of the Euler number via connectivity graph," *Pattern Recognit. Lett.*, vol. 14, no. 11, pp. 849–859, Nov. 1993.
[33] S. J. L. Díaz-De-León and J. H. Sossa-Azuela, "On the computation of the Euler number of a binary object," *Pattern Recognit.*, vol. 29, no. 3, pp. 471–476, Mar. 1996.
[34] C.-N. Lee and T. Poston, "Winding and Euler numbers for 2D and 3D digital images," *CVGIP, Graph. Models Image Process.*, vol. 53, no. 6, pp. 522–537, Nov. 1991.
[35] S. Di Zenzo, L. Cinque, and S. Levialdi, "Run-based algorithms for binary image analysis and processing," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. PAMI-18, no. 1, pp. 83–89, Jan. 1996.
[36] S. Dey, B. B. Bhattacharya, M. K. Kundu, and T. Acharya, "A fast algorithm for computing the Euler number of an image and its VLSI implementation," in *Proc. 13th Int. Conf. VLSI Design*, Jan. 2000, pp. 330–335.
[37] A. Bishnu, B. B. Bhattacharya, M. K. Kundu, C. A. Murthy, and T. Acharya, "A pipeline architecture for computing the Euler number of a binary image," *J. Syst. Archit.*, vol. 51, no. 8, pp. 470–487, Aug. 2005.
[38] L.-F. He, Y.-Y. Chao, and K. Suzuki, "An algorithm for connected-component labeling, hole labeling and Euler number computing," *J. Comput. Sci. Technol.*, vol. 28, no. 3, pp. 468–478, May 2013.
[39] [Online]. Available: http://sampl.ece.ohio-state.edu/data/stills/sidba/index.htm, accessed Jun. 2010.
[40] [Online]. Available: http://sipi.usc.edu/database/, accessed Sep. 2012.
[41] [Online]. Available: http://www1.cs.columbia.edu/CAVE/software/curet/index.php, accessed Sep. 2012.
[42] N. Otsu, "A threshold selection method from gray-level histograms," *IEEE Trans. Syst., Man Cybern.*, vol. 9, no. 1, pp. 62–66, Jan. 1979.
[43] C. J. Nicol, "Design of a connected component labeling chip for real time image processing," in *Proc. IEEE Asia-Pacific Conf. Circuits Syst.*, Sydney, NSW, Australia, Dec. 1992, pp. 142–147.
[44] C. J. Nicol, "A systolic approach for real time connected component labeling," *Comput. Vis. Image Understand.*, vol. 61, no. 1, pp. 17–31, Jan. 1995.
[45] X. D. Yang, "Design of fast connected components hardware," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Ann Arbor, MI, USA, Jun. 1988, pp. 937–944.
[46] J. K. Udupa, "Boundary and object labelling in three-dimensional images," *Comput. Vis., Graph., Image Process.*, vol. 51, no. 3, pp. 355–369, Sep. 1990.
[47] L. He, Y. Chao, and K. Suzuki, "Two efficient label-equivalence-based connected-component labeling algorithms for three-dimensional binary images," *IEEE Trans. Image Process.*, vol. 20, no. 8, pp. 2122–2133, Aug. 2011.
[48] K.-B. Wang, T.-L. Chia, Z. Chen, and D.-C. Lou, "Parallel execution of a connected component labeling operation on a linear array architecture," *J. Inf. Sci. Eng.*, vol. 19, no. 2, pp. 353–370, 2003.

Lifeng He (SM'12) received the B.E. degree from the Northwest Institute of Light Industry, China, in 1982, a second B.E. degree from Xi'an Jiaotong University, China, in 1986, and the M.S. and Ph.D. degrees in artificial intelligence and computer science from the Nagoya Institute of Technology, Japan, in 1994 and 1997, respectively. From 2006 to 2007, he was with the University of Chicago as a Research Associate. He is currently a Guest Professor with the Shaanxi University of Science and Technology, China, and a Professor with Aichi Prefectural University, Japan. His research interests include intelligent image processing, computer vision, automated reasoning, pattern recognition, string searching, and artificial intelligence.

He is a member of the Information Processing Society of Japan, the Institute of Electronics, Information and Communication Engineers, and the Association of Authors' Representatives. He has been serving as a referee of over 10 journals in computer science, including the IEEE TRANSACTION ON NEURAL NETWORKS, the IEEE TRANSACTION ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE, the IEEE TRANSACTION ON IMAGE PROCESSING, the IEEE TRANSACTION ON COMPUTERS, *Pattern Recognition*, *Computer Vision*, *Image Understanding*, and *Pattern Recognition Letters*.

Yuyan Chao received the B.E. degree from the Northwest Institute of Light Industry, China, in 1984, and the M.S. and Ph.D. degrees from Nagoya University, Japan, in 1997 and 2000, respectively. From 2000 to 2002, she was a Special Foreign Researcher of the Japan Society for the Promotion of Science with the Nagoya Institute of Technology. She is currently a Professor with Nagoya Sangyo University, Japan, and a Guest Professor with the Shaanxi University of Science and Technology, China. Her research interests include image processing, graphic understanding, Computer Aided Design, pattern recognition, and automated reasoning.