He LF, Chao YY, Suzuki K. An algorithm for connected-component labeling, hole labeling and Euler number computing. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 28(3): 468–478 May 2013. DOI 10.1007/s11390-013-1348-y

An Algorithm for Connected-Component Labeling, Hole Labeling and Euler Number Computing

Li-Feng He[1,2] (何立风), *Senior Member*, *IEEE*, Yu-Yan Chao[1,3] (巢宇燕)
and Kenji Suzuki[4], *Senior Member*, *IEEE*

[1] *Artificial Intelligence Institute, College of Electrical and Information Engineering, Shaanxi University of Science and Technology, Xi'an 710021, China*

[2] *Faculty of Information Science and Technology, Aichi Prefectural University, Aichi 4801198, Japan*

[3] *Faculty of Environment, Information and Business, Nagoya Sangyo University, Aichi 4888711, Japan*

[4] *Department of Radiology, The University of Chicago, IL 60637, U.S.A.*

E-mail: helifeng@ist.aichi-pu.ac.jp; chao@nagoya-su.ac.jp; suzuki@uchicago.edu

Received March 7, 2012; revised January 7, 2013.

Abstract Labeling connected components and holes and computing the Euler number in a binary image are necessary for image analysis, pattern recognition, and computer (robot) vision, and are usually made independently of each other in conventional methods. This paper proposes a two-scan algorithm for labeling connected components and holes simultaneously in a binary image by use of the same data structure. With our algorithm, besides labeling, we can also easily calculate the number and the area of connected components and holes, as well as the Euler number. Our method is very simple in principle, and experimental results demonstrate that our method is much more efficient than conventional methods for various kinds of images in cases where both labeling and Euler number computing are necessary.

Keywords computer vision, connected-component labeling, Euler number, hole, pattern recognition

1 Introduction

Labeling of connected components and calculating the Euler number in a binary image are two fundamental operations in image analysis, pattern recognition, computer (robot) vision, and machine intelligence[1-2]. Labeling is necessary whenever independent objects (connected components) are to be recognized in a binary image. On the other hand, the Euler number, which is defined as the difference between the number of connected components and that of holes in a binary image, is a basic topologic property of a binary image and is used for processing cell images in medical diagnosis[3], document image processing[4], shadow detection[5], reflectance-based object recognition[6], and robot vision[7].

Many algorithms have been proposed for the two operations. For labeling, there are mainly two classes of algorithms: 1) raster-scan algorithms[8-15] and 2) label propagation algorithms[16-19]. According to experimental results for various types of images, the algorithm proposed by He, Chao, and Suzuki[15], which is an improvement of their two-scan algorithm proposed in [14], is the most efficient one, and has been used for various applications[20-21]. For convenience, we denote this algorithm as HCS algorithm.

There are also many algorithms proposed for calculating the Euler number in a binary image[22-24]. One of the most famous algorithms is based on counting certain 2×2 pixel patterns called bit-quads[25-26] and is used in the MATLAB image-processing tool box①. For convenience, we denote this algorithm as ML algorithm.

Although the Euler number and the number of connected components in a binary image are closely related, they are usually calculated separately by different algorithms in conventional methods.

This paper proposes an algorithm for labeling connected components and holes simultaneously; thus, the Euler number can also be calculated easily. Our algorithm is especially efficient when both labeling and Euler number computing are necessary.

Regular Paper

This work was supported in part by the Grant-in-Aid for Scientific Research (C) of the Ministry of Education, Science, Sports and Culture of Japan under Grant No. 23500222.

① http://www.mathworks.com/access/helpdesk/help/toolbox/images/bweuler.html, March 2012.

The rest of this paper is organized as follows: We review the HCS labeling algorithm and the ML algorithm for Euler number computing. Section 3 introduces our algorithm. We provide experimental results in Section 4, and make a discussion in Section 5. Lastly, we give our concluding remarks in Section 6.

2 Review of the HCS Algorithm and ML Algorithm

For an $N \times M$-size binary image, we use $b(x, y)$ to denote the pixel as well as its value at (x, y) in the image, where $0 \leqslant x \leqslant N-1$ and $0 \leqslant y \leqslant M-1$. We assume that the foreground pixels and background pixels in a given binary image are represented by 1 and 0, respectively. As in most labeling algorithms, we assume that all pixels on the border of an image are background pixels.

A pixel is a 4-neighbor of pixel $b(x, y)$ if it shares an edge with $b(x, y)$. The 4-neighbors of pixel $b(x, y)$, namely $P2$, $P4$, $P6$ and $P8$, are shown in Fig.1. On the other hand, a pixel is an 8-neighbor of pixel $b(x, y)$ if it shares an edge or a vertex with $b(x, y)$. The 8-neighbors of pixel $b(x, y)$, namely $P1 \sim P8$, are shown in Fig.1.

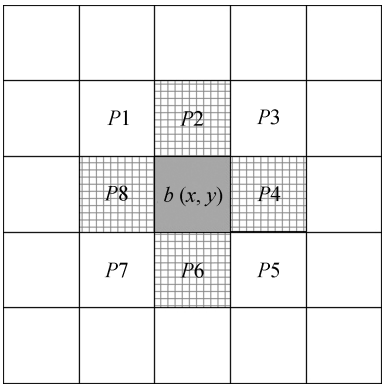


Fig.1. Neighbors of a pixel.

Two foreground (background) pixels, p and q, are said to be 8-connected (4 -connected) if there is a path of foreground pixels (background pixels) $a_1, \ldots, a_n$ where $a_1 = p$ and $a_n = q$, such that a_i and a_{i+1} are 8-neighbor (4-neighbor) for all $1 \leqslant i \leqslant n-1$. For example, in Fig.2, pixels A and B are 8-connected foreground pixels, and pixels C and D are 4-connected background pixels. Usually, we consider 8-connectivity for connected components (foreground pixels) and 4-connectivity for holes (background pixels)[②].

A *connected component* in a binary image is a maximum set of foreground pixels such that any two pixels in the set are 8-connected. On the other hand, a *hole* is a maximum set of background pixels such that they are enclosed by foreground pixels, and any two pixels in the set are 4-connected. For example, in Fig.2, there are two connected components and three holes; thus, the Euler number is $2 - 3 = -1$.

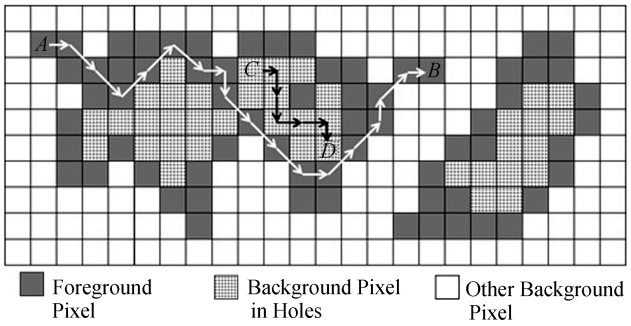


Fig.2. Binary image.

2.1 HCS Algorithm

The HCS algorithm is a two-scan labeling algorithm. Similar to other two-scan labeling algorithms, it completes labeling in two raster scans by three processes: 1) provisional label assignment, i.e., assigning a provisional label to each foreground pixel, and finding equivalent labels (i.e., provisional labels assigned to a connected component); 2) label equivalence record (i.e., using some data structures to record equivalent labels) and label-equivalence resolving (i.e., finding a representative label for all equivalent provisional labels); 3) label replacement (i.e., replacing each provisional label by its representative label). The first process is performed in the first scan, the second process is usually executed in the first scan or between the first scan and the second scan, and the third process is carried out in the second scan.

The HCS algorithm uses equivalent label sets and a representative label table to record equivalent labels and resolve the label equivalences. At any time, the smallest label in an equivalent-label set is used as the representative label for all labels in the set. For convenience, an equivalent label set with the representative label u is denoted as $S(u)$, and the representative label of a provisional label s is t, denoted as $t = T[s]$.

In the first scan, this algorithm uses the mask shown in Fig.3(a), which consists of four scanned neighbors of the current foreground pixel, to assign a provisional label to each foreground pixel, and to record and resolve label equivalences. At any moment, all equivalent provisional labels are combined in an equivalent label set with the same representative label.

[②]If we also use 8-connectivity for holes, all background pixels in Fig.2 will be 8-connected; thus, there will be no hole in Fig.2. Therefore, we should use 4-connectivity for holes.

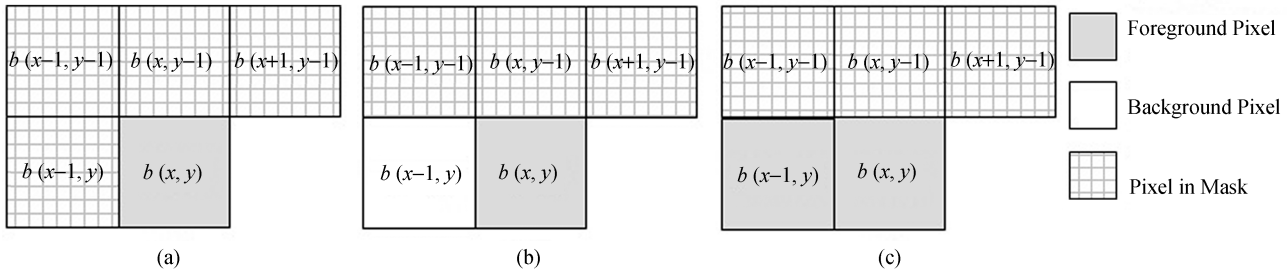


Fig.3. Mask for the 8-connected connectivity.

For the case where the current foreground pixel follows a background pixel (Fig.3(b)), if there is no label (foreground pixel) in the mask, this means that the current foreground pixel does not connect with any scanned foreground pixel, and the current foreground pixel belongs to a new connected component. The algorithm assigns a new provisional label l, which is initialized to 1, to the current foreground pixel and establishes the equivalent label set $S(l) = \{l\}$; it sets the representative label table as $T[l] = l$, and $l = l+1$ for later processing. Otherwise, i.e., if there are some foreground pixels in the mask, all of such foreground pixels and the current foreground pixel belong to the same connected component. Therefore, the current foreground pixel can be assigned any of the labels in the mask. On the other hand, for the case where the current foreground pixel follows another foreground pixel (Fig.3(c)), the current foreground pixel can be assigned the same label as that of the foreground pixel.

In any case, if there are provisional labels belonging to different equivalent label sets in the mask, all provisional labels in those sets are equivalent labels, and thus, all such sets should be combined together.

As introduced in [14], equivalent label sets and the representative table for resolving label equivalences can be implemented efficiently by use of three one-dimensional arrays, where an equivalent label set is represented as a list. The first array, R, is for the representative label table. The fact that the representative label of provisional label a is r is realized by $R[a] = r$. The second array, *next*, is for expressing the next label of a provisional label in an equivalent label set. $next[d] = e$ indicates that the label next to label d is e. Especially, $next[f] = -1$ means that there is no label next to label f, i.e., f is the last label in the equivalent label set. The third array, *last*, is used for expressing the last label of an equivalent label set. $last[p] = t$ means that the last label in the equivalent label set $S(p)$ is t.

When a new equivalent label set $S(l) = \{l\}$ is established, because there is only one label in the set, we know that the representative label of label l is itself; there is no label next to label l; and the last label in the set is label l. Thus, the operations for establishing a new equivalent label set $S(l) = \{l\}$ can be made by the following process:

$$\begin{aligned} &R[l] = l;\\ &next[l] = -1;\\ &last[l] = l;\\ &l = l + 1. \end{aligned}$$

On the other hand, the procedure for combining two equivalent label sets $S(u)$ and $S(v)$, namely $combine(u, v)$, can be made as follows: if u is smaller than v, the equivalent label set $S(v)$ is merged into equivalent label set $S(u)$, where 1) for each label k in $S(v)$, changing its representative label from v to u; 2) changing the next label of the last label in $S(u)$ to v; 3) changing the last label of $S(u)$ to the last label of $S(v)$. Otherwise, i.e., if u is larger than v, the equivalent label set $S(u)$ is merged into equivalent label set $S(v)$, where 1) for each label k in $S(u)$, changing its representative label from u to v; 2) changing the next label of the last label in $S(v)$ to u; 3) changing the last label of $S(v)$ to the last label of $S(u)$.

The pseudo-code for $combine(u, v)$ can be shown as follows:

```
if u is smaller than v,
    1) (∀k ∈ S(v))(R[k] = u);
    2) next[last[u]] = v;
    3) last[u] = last[v];
else if u is larger than v
    1) (∀k ∈ S(u))(R[k] = v);
    2) next[last[v]] = u;
    3) last[v] = last[u];
end of if.
```

For an $N \times M$-sized binary image, because the maximum number of provisional labels for labeling the image is $N \times M/4$, the sizes of the arrays R, *next*, and *last* should be taken as $N \times M/4$.

As soon as the first scan is finished, all equivalent labels of each connected component have been combined into a corresponding equivalent label set with a unique

representative label. In the second scan, by replacement of each provisional label by its representative label, all foreground pixels of each connected component will be assigned a unique label.

2.2 Algorithm for Calculating the Euler Number in a Binary Image

The ML algorithm for calculating the Euler number in a binary image[26] is based on counting certain 2×2 pixel patterns called bit-quads, which are shown in Fig.4, in the image. Let N_1, N_2, and N_3 be the number of patterns P_1, P_2, and P_3 in the image, respectively. Then, the formula for calculating the Euler number, namely E, for 8-connectivity is:

$$E = (N_1 - N_2 - 2N_3)/4.$$

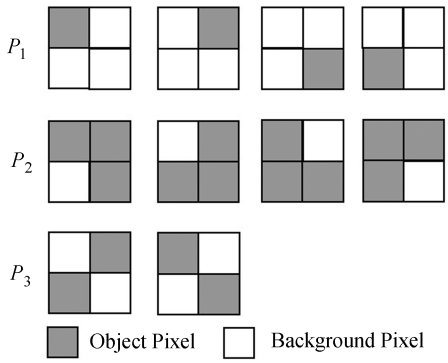


Fig.4. Bit-quads for calculating the Euler number in the ML algorithm.

3 Our Proposed Algorithm

Our method is mainly aimed at the cases where both connected component labeling and Euler number computing are necessary. By conventional methods, in such cases, we need either to run a labeling program and an Euler number computing program separately or to run a labeling program twice: one is for labeling connected components, and the other is for labeling holes. We extend the HCS algorithm to be able to label connected components and holes simultaneously in two raster-scans; thus, the Euler number can also be calculated easily.

In order to label connected components and holes simultaneously with the same data structure, we use the numbers between 0 and H as provisional labels for labeling holes and those larger than H for labeling connected components. For $N \times M$-sized binary images, the maximum number of both connected components and holes is $N \times M/4$. Therefore, the size of R, *next*, and *last* should be enlarged to $N \times M/2$, and the value of H should be set to $N \times M/4$.

3.1 Labeling Connected Components and mHoles

Except for initializing the variable l for provisional labels to $H + 1$, labeling connected components in our method is the same as that in the HCS algorithm. For labeling holes, we initialize the variable, say, l_{H}, for provisional labels for holes to 0, which is for the background pixels 4-connected to the background pixels on the border of the image. The mask for labeling holes consists of the processed two neighbor pixels, as shown in Fig.5(a).

For a background pixel $b(x, y)$ following a foreground pixel (Fig.5(b)), we check the other pixel in the mask, i.e., $b(x, y-1)$. If $b(x, y-1)$ is a background pixel, i.e., $b(x, y-1) < H$, we assign its label to $b(x, y)$. Otherwise, if $b(x, y-1)$ is a foreground pixel, we assign a new provisional label to $b(x, y)$. The pseudo-code of the procedure for this case can be given as follows:

```
if b(x, y−1) is smaller than H
    let b(x, y) = b(x, y−1);
else
    assign b(x, y) a new label;
    l_H increases 1;
end of if.
```

On the other hand, for a background pixel $b(x, y)$ following another background pixel (Fig.5(c)), we assign the label of $b(x-1, y)$ to $b(x, y)$. If the other pixel in the mask, i.e., $b(x, y-1)$, is a background pixel, then $b(x-1, y)$ and $b(x, y-1)$ belong to the same hole. If

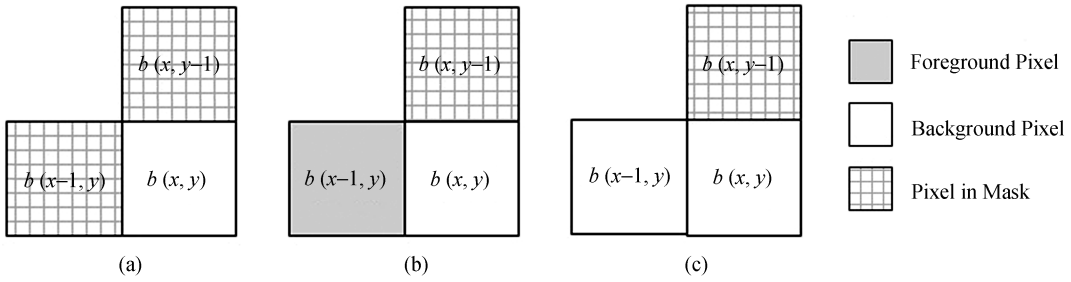


Fig.5. Mask for labeling holes.

the provisional labels of $b(x-1, y)$ and $b(x, y-1)$ belong to different equivalent label sets, we should combine the two sets. This task can also be completed by use of $combine(u, v)$ introduced above. The pseudo-code of the procedure for this case can be given as follows:

```
b(x, y) = b(x − 1, y);
if b(x, y − 1) is smaller than H
    let b(x − 1, y) belong to S(u) and
    b(x, y − 1) to S(v), call combine(u, v);
end of if.
```

The second scan for labeling connected components and labeling holes can be finished by a single scan as follows:

```
for each pixel b(x, y)
    replace its value with R[b(x, y)];
end of for.
```

3.2 Calculating the Number of Connected Components

According to the HCS algorithm, after the first scan, all equivalent provisional labels (which belong to the same connected component) will be combined in an equivalent label set. Therefore, the number of connected components in a binary image is equal to that of the equivalent label sets. Because in each equivalent label set $S(r)$, only the label r's representative label is itself, we can calculate the number of the equivalent label sets, i.e., the number of connected components, by counting the times that $R[i]$ is equal to i between $R[H+1]$ and $R[L-1]$, where L is the terminal value of the variable for provisional labels for labeling connected components (i.e., l in the pseudo-code described above). The pseudo-code for calculating the number of connected components, namely N_{CC}, can be shown as follows:

```
initialize N_CC to 0;
for i from H + 1 to L − 1
    if R[i] is equal to i, N_CC increases 1;
end of for.
```

3.3 Calculating the Number of Holes

Similar to the number of connected components, the number of holes is equal to that of equivalent label sets established during labeling holes. Therefore, we can calculate the number of holes, namely N_{H}, by counting the times of $R[i] == i$ between $R[1]$[③] and $R[L_{\mathrm{H}}-1]$, where L_{H} is the terminal value of the variable for provisional labels for holes (i.e., l_{H} in the pseudo-code described above).

```
initialize N_H to 0;
for i from 1 to L_{H−1}
    if R[i] is equal to i, N_H increases 1;
end of for.
```

3.4 Calculating the Euler Number

With the number of connected components N_{CC} and that of holes N_{H} in a binary image in hand, according to the definition of the Euler number, the Euler number in the binary image, say, N_{E}, can be calculated simply by the following formula:

$$N_{\mathrm{E}} = N_{\mathrm{CC}} - N_{\mathrm{H}}.$$

4 Experimental Results

All algorithms used for our comparison were implemented in the C language on a PC-based workstation (Intel Pentium D 930 3.0 GHz + 3.0 GHz CPUs, 2 GB memory, Mandriva Linux OS), and compiled by the GNU C complier (version 4.2.3) with the option −O3. The source codes of our algorithm can be downloaded from http://www.aichi-pu.ac.jp/ist/∼helifeng/.

All experimental results presented in this section were obtained by averaging of the execution time for 5 000 runs by use of one CPU core. The results of the number of connected components, that of holes, and that of the Euler number are exactly the same for all runs of each image.

Images used for testing are composed of four types: artificial images, natural images, texture images, and medical images. Artificial images contain specialized patterns (stair-like, spiral-like, saw-tooth-like, checker-board-like, and honey comb-like connected components)[12] and noise images. Forty-one noise images of each of five sizes (32×32, 64×64, 128×128, 256×256, and 512×512 pixels) were used for testing (a total of 205 images). For each size, the 41 noise images were generated by thresholding of the images containing uniform random noise with 41 different threshold values from 0 to 1 000 in steps of 25. Because connected components in such noise images have complicated geometric shapes and complex connectivity, severe evaluations of algorithms can be performed with these images. Some of noise images with various densities are shown in Fig.6.

On the other hand, 50 natural images, including landscape, aerial, fingerprint, portrait, still-life, snapshot, and text images, obtained from the Standard Image Database (SIDBA) developed by the University of Tokyo[④] and the image database of the University of

[③] Notice that the representative label 0 is used for labeling the block of background pixels connected to background pixels on the image border, which is not a hole, we should not count the case of $R[0] == 0$.

[④] http://sampl.ece.ohio-state.edu/data/stills/sidba/index.htm, June 2010.

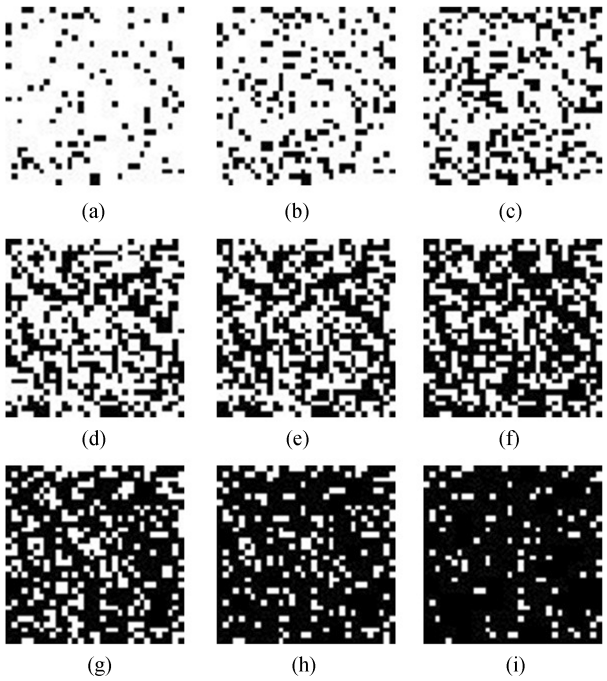


Fig.6. Samples of noise images with various densities. (a) 0.1. (b) 0.2. (c) 0.3. (d) 0.4. (e) 0.5. (f) 0.6. (g) 0.7. (h) 0.8. (i) 0.9.

Southern California⑤ were used for realistic testing of related algorithms. In addition, seven texture images, which were downloaded from the Columbia-Utrecht Reflectance and Texture Database⑥, and 25 medical images obtained from a medical image database of The University of Chicago were used for testing. All of these images were 512×512 pixels in size, and they were transformed into binary images by means of Otsu's threshold selection method[27].

We compared our algorithm with the ML algorithm in the following two ways: 1) only calculating the Euler number; 2) labeling and calculating the Euler number. Notice that, in case 1, the second scan in our algorithm is unnecessary; in case 2, the comparative algorithm consists of the HCS algorithm (used for labeling) and the ML algorithm (used for calculating the Euler number). For convenience, we use Ours1 and Ours2 to denote our algorithm in case 1 and case 2, and ML1 and ML2 to denote the comparative algorithm in case 1 and case 2, respectively.

4.1 Execution Time Versus the Size of an Image

We used all noise images to test the linearity of the execution time versus image size. As shown in Fig.7, both the maximum execution time and the average execution time of all of the four algorithms have the ideal linear characteristics versus the number of pixels in an image. We can find that, if we calculate only the Euler number, the ML1 algorithm is better than the Our1 algorithm for the maximum execution time, but the two algorithms are almost the same for the average execution time. When both labeling and Euler number computing are necessary, Ours2 is much better than ML2.

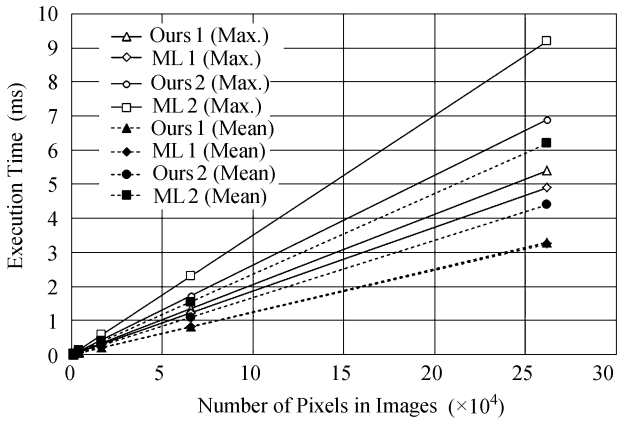


Fig.7. Execution time versus number of pixels in images.

4.2 Execution Time Versus the Density of an Image

Noise images with a size of 512×512 pixels were used for testing the execution time versus the density of the foreground pixels in an image. The results are shown in Fig.8. We can find that, in the cases where only the Euler number is calculated, Ours1 is better

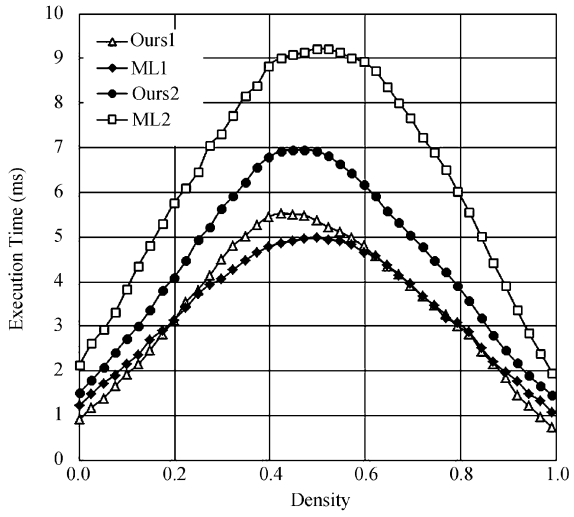


Fig.8. Execution time versus number of density in images.

⑤http://sipi.usc.edu/database/, September 2012.

⑥http://www1.cs.columbia.edu/CAVE/software/curet/, September 2012.

than ML1 for images with low and high densities, but is worse for images with middle densities. For the cases where both labeling and Euler number computing are necessary, Ours2 is much better than ML2 for all densities.

4.3 Comparisons in Terms of Maximum, Mean, and Minimum Execution Time

Natural images, medical images, texture images, and artificial images with specialized shape patterns were used for this test. The results of the comparisons are shown in Table 1, where the artificial* images are those artificial images introduced above except for the stair-like image. From Table 1, except for the stair-like image, for all kinds of images, both Ours1 and Ours2 are much more efficient than ML1 and ML2, respectively. In fact, except the stair-like image and a text image (Fig.9(c)), for each image used in this test, Ours1 is more efficient than ML1, and except for the stair-like image, Ours2 is much more efficient than ML2. The execution time (ms), the number of connected components N_{CC}, the number of holes N_H, and the Euler number N_E for the selected six images are illustrated in Fig.9, where the foreground pixels are displayed in black.

Table 1. Maximum, Mean, and Minimum Execution Time (ms) on Various Types of Images

Image Type		Ours1	ML1	Ours2	ML2
Natural	Max.	2.48	2.65	3.46	4.88
	Mean	1.35	1.64	2.12	2.86
	Min.	0.79	1.19	1.59	2.08
Medical	Max.	1.52	1.83	2.23	3.26
	Mean	1.17	1.47	1.89	2.60
	Min.	0.84	1.29	1.73	2.36
Textural	Max.	2.10	2.21	2.85	4.14
	Mean	1.41	1.64	2.17	3.08
	Min.	0.67	1.05	1.48	2.01
Artificial*	Max.	0.84	1.16	1.60	2.14
	Mean	0.47	0.59	0.80	1.14
	Min.	0.26	0.29	0.33	0.60
Stair-like		2.23	1.07	3.13	2.62

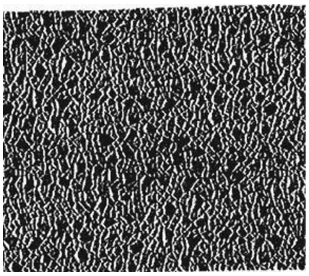
ML1: 1.72 Ours1: 1.46
ML2: 3.19 Ours2: 2.24
N_{CC}: 172 N_H: 391
N_E: −219

ML1: 1.50 Ours1: 1.14
ML2: 2.74 Ours2: 1.94
N_{CC}: 271 N_H: 447
N_E: −176

ML1: 1.57 Ours1: 1.92
ML2: 3.23 Ours2: 2.81
N_{CC}: 2 290 N_H: 1 354
N_E: 936

ML1: 1.39 Ours1: 0.90
ML2: 2.33 Ours2: 1.72
N_{CC}: 49 N_H: 54
N_E: −5

ML1: 1.62 Ours1: 1.51
ML2: 2.90 Ours2: 2.07
N_{CC}: 442 N_H: 917
N_E: −475



ML1: 2.16 Ours1: 1.98
ML2: 3.98 Ours2: 2.70
N_{CC}: 35 N_H: 2 272
N_E: −2 237

Fig.9. Execution time (ms), the number of connected components N_{CC}, the number of holes N_H, the Euler number N_E for the selected six images. (a) Fingerprint image. (b) Portrait image. (c) Text image. (d) Snap shot image. (e) Medical image. (f) Texture image.

5 Discussion

5.1 General Analysis of Ours1, Ours2, ML1 and ML2

Both the HCS labeling algorithm and the ML algorithm are sequential processing ones: they process pixels one by one in the raster scan order. The major operation of both the algorithms for processing a pixel is checking the values of pixels.

According to the analysis results shown in [15], for processing a foreground pixel in the first scan, the average number of times for checking pixels by the HCS

algorithm, namely N_{HCS1f}, is 2.75 and that for processing a background pixel, namely N_{HCS1b}, is 1. Thus, the average number of times for checking pixels by the HCS algorithm in the first scan, namely N_{HCS1}, is $(N_{\mathrm{HCS1f}} + N_{\mathrm{HCS1b}})/2 = (2.75 + 1)/2 = 1.875$[⑦].

In our proposed algorithms, for processing a foreground pixel in the first scan, the average number of times for checking pixels, namely N_{Ours1f}, is exactly the same with the HCS algorithm, thus, $N_{\mathrm{Ours1f}} = N_{\mathrm{HCS1f}} = 2.75$. For processing a background pixel to labeling holes, from the explanation in Subsection 3.1, the average number of times for checking pixels, namely N_{Ours1b}, is 2. Moreover, if we equate the checking operation and the rewriting operation, the average number of times for checking pixels in the second scan in the HCS algorithm and our proposed algorithm, namely N_2, is 1.

On the other hand, according to the ML algorithm introduced in Subsection 2.2, the average number of times for checking pixels for processing any pixel by the ML algorithm, namely N_{ML}, is 4.

Thus, the average number of times for checking pixels for processing a pixel by Ours1, namely N_{Ours1}, is $(N_{\mathrm{Ours1f}} + N_{\mathrm{Ours1b}})/2 = (2.75 + 2)/2 = 2.375$; that by Ours2, namely N_{Ours2}, is $N_{\mathrm{Ours1}} + 1 = 3.375$; that by ML1, namely, N_{ML1}, is 4; that by ML2, namely N_{ML2}, is $N_{\mathrm{ML1}} + N_{\mathrm{HCS1}} + N_2 = 4 + 1.875 + 1 = 6.875$.

Although the average number of times for checking pixels for processing a pixel by Ours1, i.e., $N_{\mathrm{Ours1}} = 2.375$, is a little smaller than that by ML1, which is 4, Ours1 also needs to initialize data structures and to resolve label equivalences, therefore, Our1 is hardly more efficient than ML1.

However, in the cases where we not only calculate the Euler number, but also label connected components, the average number of times for checking pixels for processing a pixel by Ours2 is $N_{\mathrm{Ours2}} = 3.375$, which is much smaller than that by ML2, i.e., $N_{\mathrm{ML2}} = 6.875$. That is to say, Ours2 is much more efficient than ML2.

5.2 Analysis of Experimental Results

We first analyze the reason that Ours1 and Ours2 are not so efficient as ML1 and ML2 for the stair-like images used in our test. As shown in Fig.10(a), the stair-like image consists of many oblique foreground-pixel lines and background-pixel lines. By our algorithms, for each such line, many provisional labels will be assigned to the pixels of the line, which will finally be combined into the same equivalent label set. In fact, for this image, the number of the provisional labels assigned by our algorithms for foreground-pixel lines in this image is 32 577, and the number of the connected components is 255, and the provisional labels for background-pixel lines and the number of holes are 65 536 and 0, respectively. That is, we need to initialize 32 577 equivalent label sets and merge them to 255 equivalent label set for foreground-pixel lines, and initialize 65 536 equivalent label sets and merge them to one equivalent label sets for background-pixel lines. Initializing so many equivalent label sets and combining so many equivalent label sets together require time-consuming work.

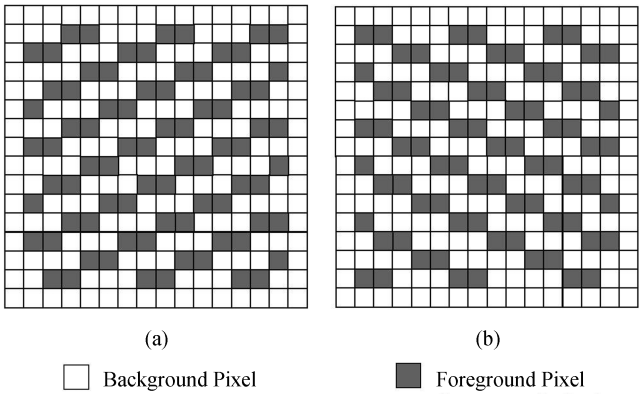


Fig.10. Stair-like images.

On the other hand, the 2×2 pixel patterns in these image are regular and are favor types for checking in our implementation of the ML algorithm. If we change the line direction as in Fig.10(b), the results will vary considerably. By our algorithms, only 305 provisional labels were generated for 305 foreground lines, and only one provisional label was generated for background pixels. Therefore, the work for initializing equivalent label sets is reduced greatly and we do not need to combine any equivalent label set. Thus, our algorithms are very efficient for this image. The results on the two images are shown in Table 2.

Table 2. Execution Time (ms) for Stair-Like Images

Image	Ours1	ML1	Ours2	ML2
Fig.10(a)	2.23	1.07	3.13	2.62
Fig.10(b)	1.04	1.05	1.56	2.15

In all natural images, textual images, and medical images, the only image that Ours1 is not as efficient as ML1 is the text image shown in Fig.9(c). The reason is similar to that in the case of the stair-like images. The result indicates that Ours1 is not efficient for text images for Euler number computing. However, when labeling is also necessary, Ours2 is still more efficient than ML2. Moreover, ML1 can only calculate the Euler number, whereas Ours1, except for the Euler number,

[⑦] We assume that the probability that a pixel is a background pixel or a foreground pixel is the same.

can also obtain the number of connected components and the number of holes simultaneously, which is valuable in many cases.

5.3 Generating Consecutive Labels for Connected Components and Holes

In many cases, we need consecutive labels for connected components and holes. We use positive consecutive numbers for connected components and negative consecutive numbers for holes. We need only to execute the following simple program after the first scan and before the second scan:

```
k is initialized to −1;
for i from 1 to L_H − 1
    if R[i] is equal to i
        set R[i] to k;
        k decreases 1;
    end of if
end of for
m is initialized to 1;
for i from H + 1 to L − 1
    if R[i] is equal to i
        set R[i] to m;
        m increases 1;
    end of if
end of for.
```

When the above processing terminates, $|k|-1$ is the number of holes and $m-1$ is that of connected components.

By Ours2 with the above processing for generating consecutive labels, the result for the image shown in Fig.2 is shown in Fig.11.

5.4 Calculating Areas (Number of Pixels) of Connected Components and Holes

After generating consecutive labels, we can modify the second scan to calculate the area (i.e., the number of pixels) of connected components and that of holes. If we use $SC[i]$ to denote the number of the connected component with the representative label i, $SH[j]$ to denote the number of the hole with the representative label $-j$, the modified pseudo-code for the second scan with calculation of the numbers of connected components and holes can be shown as follows, where the size of the array SC and the array SH should be the maximum possible number of the connected components and holes in a binary image, respectively; thus, both are $N \times M/4$, and all of the elements of SC and SH are initialized to 0:

```
for y from 0 to M − 1
    for x from 0 to N − 1
        set b(x, y) to R[b(x, y)];
        if b(x, y) is bigger than 0
            SC[b(x, y)] increases 1;
        else
            SH[|b(x, y)|] increases 1;
        end of if
    end of for
end of for.
```

Notice that $SH[0]$ indicates the number of background pixels except for those belonging to holes.

5.5 Maximum Execution Time of Our Algorithm

According to [14], for an $N \times M$-sized image, the order of the HCS algorithm for labeling connected components is $O(N \times M)$. Because we use the same method

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	1	0	1	1	1	1	0	1	1	1	0	0	0	0	0	0	0	0	2	2	0	0
0	0	1	1	1	1	-1	1	1	-2	-2	-2	1	1	0	0	0	0	0	2	2	2	0	0
0	0	0	1	1	-1	-1	-1	1	-2	-2	1	-2	1	0	0	0	0	0	2	-3	-3	2	0
0	0	1	-1	-1	-1	-1	-1	-1	1	-2	-2	-2	1	0	0	0	0	2	-3	-3	-3	2	0
0	0	1	-1	1	-1	-1	-1	1	0	1	-2	-2	1	0	0	0	2	2	-3	-3	2	0	0
0	0	1	1	0	1	-1	1	0	0	0	1	1	1	0	0	2	-3	-3	-3	-3	2	0	0
0	0	0	0	0	0	1	1	0	0	0	1	1	0	0	0	2	2	-3	-3	2	0	0	0
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	2	2	2	2	2	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Foreground Pixel Background Pixel in Holes Other Background Pixel

Fig.11. Output image of the input image shown in Fig.2 by Ours2 with consecutive-label processing.

and data structure for labeling holes, the order for labeling holes should also be $O(N \times M)$. Moreover, the execution time for calculating the number of connected components is propositional to the number of provisional labels assigned to connected components, whose order is also $O(N \times M)$. In the same way, the order for execution time for calculating the number of holes is also $O(N \times M)$. Therefore, the order of our algorithm should also be $O(N \times M)$, just as demonstrated in the experimental results on the noise images of different sizes shown in Fig.7.

6 Conclusions

Although the Euler number in a binary image is closely related to the number of connected components in the image, no algorithm was proposed for calculating the Euler number by a labeling algorithm alone for pixel-based images. This paper supplies such a gap. Our algorithm can label connected components and holes in a binary image simultaneously in two raster scans by use of the same data structures, and then the number of connected components, the number of holes, and the Euler number can easily be calculated by use of the data in the data structure after labeling. The experimental results demonstrated that, for almost all real images, our algorithm is more efficient than the algorithm used in the MATLAB image tool for calculating the Euler number in a binary image, and is much more efficient than conventional methods if both labeling and the Euler number are necessary.

For future work, we plan to implement our algorithm in hardware[28-29] to accelerate labeling speed, to extend it to include three-dimensional images[30-31], and to develop algorithms for parallel architectures[32-33].

References

[1] Gonzalez R C, Woods R E. Digital Image Processing (3rd edition). Addison-Wesley, 1992.

[2] Ronsen C, Denjiver P A. Connected Components in Binary Images: The Detection Problem. New York, USA: John Wiley & Sons. Inc., 1984.

[3] Hashizume A, Suzuki R, Yokouchi H *et al.* An algorithm of automated RBC classification and its evaluation. *Bio. Medical Engineering*, 1990, 28(1): 25-32. (In Japanese)

[4] Srihari S N. Document image understanding. In *Proc. ACM Fall Joint Computer Conference*, November 1986, pp.87-95.

[5] Rosin P L, Ellis T. Image difference threshold strategies and shadow detection. In *Proc. British Machine Vision Conference*, September 1995, pp.347-356.

[6] Nayar S K, Bolle R M. Reflectance-based object recognition. *International Journal of Computer Vision*, 1996, 17 (3): 219-240.

[7] Horn B K P. Robot Vision. New York: McGraw-Hill Higher Education, 1986, pp.73-77.

[8] Lumia R, Shapiro L, Zungia O. A new connected components algorithm for virtual memory computers. *Comput. Vision, Graphics and Image Processing*, 1983, 22(2): 287-300.

[9] Rosenfeld A, Pfalts J L. Sequential operations in digital picture processing. *Journal of ACM*, 1996, 13(4): 471-494.

[10] Rosenfeld A, Kak A C. Digital Picture Processing (2nd edition), Vol. 2. San Diego, USA: Academic Press, 1982.

[11] Naoi S. High-speed labeling method using adaptive variable window size for character shape feature. In *Proc. IEEE Asian Conf. Computer Vision*, December 1995, pp.408-411.

[12] Suzuki K, Horiba I, Sugie N. Linear-time connected-component labeling based on sequential local operations. *Computer Vision and Image Understanding*, 2003, 89(1): 1-23.

[13] Wu K, Otoo E, Suzuki K. Optimizing two-pass connected-component labeling algorithms. *Pattern Analysis & Applications*, 2009, 12(2): 117-135.

[14] He L, Chao Y, Suzuki K, Wu K. Fast connected-component labeling. *Pattern Recognition*, 2009, 42(9): 1977-1987.

[15] He L, Chao Y, Suzuki K. An efficient first-scan method for label-equivalence-based labeling algorithms. *Pattern Recognition Letters*, 2010, 31(1): 28-35.

[16] Ballard D H, Brown C M. Computer Vision. Prentice-Hall, 1982.

[17] Chang F, Chen C J, Lu C J. A linear-time component-labeling algorithm using contour tracing technique. *Computer Vision and Image Understanding*, 2004, 93(2): 206-220.

[18] Hu Q, Qian G, Nowinski W L. Fast connected-component labeling in three-dimensional binary images based on iterative recursion. *Computer Vision and Image Understanding*, 2005, 99(3): 414-434.

[19] Shima Y, Murakami T, Koga M, Yashiro H, Fujisawa H. A high-speed algorithm for propagation-type labeling based on block sorting of runs in binary images. In *Proc. the 10th Int. Conf. Pattern Recognition*, June 1990, pp.655-658.

[20] Wolfe C, Nicholas Graham T C, Pape J A. Seeing through the fog: An algorithm for fast and accurate touch detection in optical tabletop surfaces. In *Proc. ACM International Conference on Interactive Tabletops and Surfaces*, November 2010, pp.73-82.

[21] Abramov A. Kulvicius T, Wörgötter F, Dellen B. Real-time image segmentation on a GPU. In *Lecture Notes in Computer Science 6310*, Keller R, Kramer D, Weiss J P (eds.), Springer-Verlag, 2011, pp.131-142.

[22] Chen M H, Yan P F. A fast algorithm to calculate the Euler number for binary image. *Pattern Recognition Letters*, 1988, 8(5): 295-297.

[23] Díaz-De-León S J L, Sossa-Azuela J H. On the computation of the Euler number of a binary object. *Pattern Recognition*, 1996, 29(3): 471-476.

[24] Di Zenzo S, Cinque L, Levialdi S. Run-based algorithms for binary image analysis and processing. *IEEE Transactions on PAMI*, 1996, 18(1): 83-89.

[25] Gray S B. Local properties of binary images in two dimensions, *IEEE Transactions on Computers*, 1971, 20(5): 551-561.

[26] Pratt W K. Digital Image Processing. New York: John Wiley & Sons, Inc., 1991, p.633.

[27] Otsu N. A threshold selection method from gray-level histograms. *IEEE Trans. Syst., Man and Cybernet*, 1979, 9(1): 62-66.

[28] Dey S, Bhattacharya B B, Kundu M K, Acharya T. A fast algorithm for computing the Euler number of an image and its VLSI implementation. In *Proc. the 13th Int. Conf. VLSI Design*, January 2000, pp.330-335.

[29] Ito Y, Nakano K. Optimized component labeling algorithm for using in medium sized FPGAs. In *Proc. the 9th Int. Conf. Parallel and Distributed Computing, Applications and Technologies*, Dec. 2008, pp.171-176.

[30] Udupa J K, Ajjanagadde V G. Boundary and object labeling in three-dimensional images. *Comput. Vision, Graphics, and Image Processing*, 1990, 51 (3): 355-369.

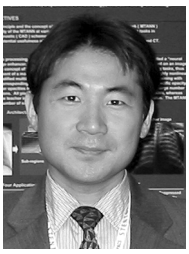
[31] He L, Chao Y, Suzuki K. Two efficient label-equivalence-based connected-component labeling algorithms for 3-dimensional binary images. *IEEE Transactions on Image Processing*, 2011, 20(8): 2122-2134.

[32] Niknam M, Thulasiraman P, Camorlinga S. A parallel algorithm for connected component labeling of gray-scale images on homogeneous multicore architectures. *Journal of Physics*: *Conference Series* 256, 2010, 012010: 1-7.

[33] Dey S, Bhattacharya B, Kundu M, Bishnu A, Acharya T. A co-processor for computing the Euler number of a binary image using divide-and-conquer strategy. *Fundamental Informaticae*, 2007, 76(1/2): 75-89.

Li-Feng He received his B.E. degree from the Northwest Institute of Light Industry, Xianyang, China, in 1982, the second B.E. degree from Xi'an Jiaotong University, China, in 1986, and the M.S. and the Ph.D. degrees in artificial intelligence and computer science from Nagoya Institute of Technology, Japan, in 1994 and 1997, respectively. From 1998 to 2011, he was an associate professor at the Aichi Prefectural University, Japan. From 2006 to 2007, he worked in the University of Chicago (USA) as a research associate. He is currently a professor, director of Artificial Intelligence Institute, and academic dean of the College of Electrical & Information Engineering at Shannxi University of Science and Technology, China, and a professor at Aichi Prefectural University, Japan. His research interests include intelligent image processing, computer vision, automated reasoning, and artificial intelligence. He is a senior member of IEEE.

Yu-Yan Chao received her B.E. degree in mechanical engineering from the Northwest Institute of Light Industry, China, in 1984, and M.S. and the Ph.D. degrees in computer science 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 at the Nagoya Institute of Technology. She is currently a professor at the Nagoya Sangyo University, Japan, and a guest professor in the Shaanxi University of Science and Technology, China. Her research interests include image processing, graphic understanding, CAD, and automated reasoning.



Kenji Suzuki received his B.S. and M.S. degrees in electrical and electronic engineering from the Meijo University, Nagoya, Japan, in 1991 and 1993, respectively, and his Ph.D. degree (by Published Work) in information engineering from the Nagoya University, in 2001. From 1993 to 1997, he worked in the Research and Development Center at the Hitachi Medical Corporation as a researcher. From 1997 to 2001, he worked at the Faculty of Information Science and Technology at the Aichi Prefectural University, as a faculty member. In 2001, he joined the Kurt Rossmann Laboratories for Radiologic Image Research in the Department of Radiology, Division of the Biological Sciences at The University of Chicago. Since 2006, he has been an assistant professor in the Department of Radiology, the Committee of Medical Physics, and the Cancer Research Center. His research interests include medical image analysis, machine learning, computer vision, and pattern recognition. He was elected as a senior member of the IEEE in 2004.