International Journal of Pattern Recognition
and Artificial Intelligence
Vol. 24, No. 4 (2010) 557–579

DOI: 10.1142/S0218001410008032

A RUN-BASED ONE-AND-A-HALF-SCAN CONNECTED-COMPONENT LABELING ALGORITHM

LIFENG HE

Shaanxi University of Science and Technology
Shaanxi, P. R. China

Aichi Prefectural University
Nagakute, Aichi 480-1198, Japan
helifeng@ist.aichi-pu.ac.jp

YUYAN CHAO

Graduate School of Environmental Management
Nagoya Sangyo University Aichi 488-8711, Japan
fauthor@example.com

KENJI SUZUKI

Department of Radiology, The University of Chicago
Chicago, IL 60637, USA
suzuki@uchicago.edu

This paper presents a run- and label-equivalence-based one-and-a-half-scan algorithm for labeling connected components in a binary image. Major differences between our algorithm and conventional label-equivalence-based algorithms are: (1) all conventional label-equivalence-based algorithms scan all pixels in the given image at least twice, whereas our algorithm scans background pixels once and object pixels twice; (2) all conventional label-equivalence-based algorithms assign a provisional label to each object pixel in the first scan and relabel the pixel in the later scan(s), whereas our algorithm assigns a provisional label to each run in the first scan, and after resolving label equivalences between runs, by using the recorded run data, it assigns each object pixel a final label directly. That is, in our algorithm, relabeling of object pixels is not necessary any more. Experimental results demonstrated that our algorithm is highly efficient on images with many long runs and/or a small number of object pixels. Moreover, our algorithm is directly applicable to run-length-encoded images, and we can obtain contours of connected components efficiently.

Keywords: Labeling algorithm; connected component; label equivalence; run-length encoding; raster scan.

1. Introduction

Labeling connected components in a binary image is one of the most fundamental operations in computer vision, pattern recognition, and machine intelligence.[4,11,13,29,32,38] For applications to dynamic images, e.g. automatic detection, robot vision, and automatic tracking, faster labeling algorithms are demanded. Many labeling algorithms have been proposed for addressing this issue. For ordinary computer architectures and pixel-based representation images, there are the following two classes of labeling algorithms:

(1) Raster-scan and label-equivalence-based algorithms. There are multi-scan,[6,8] two-scan,[5,7,9,15–17,24,30,31,35,42] and four-scan[39] algorithms. These algorithms scan an image in the raster scan direction at least twice (thus, all background pixels and all object pixels are processed at least twice). At the first scan, they assign a provisional label to each object pixel, and then relabel the pixel in the later scan(s) (at least once) by resolving label equivalences between provisional labels.
(2) Searching and label propagation algorithms. These algorithms[1,2,12,20,37] first search an unlabeled object pixel, label the pixel with a new label; then, in the later processing, they assign the same label to all object pixels that are connected to the pixel. Although these algorithms usually use the raster scan to find an unlabeled object pixel, for labeling, they all access pixels in an image in an irregular way, depending on the shapes of connected components in the image. Therefore, they are essentially not a raster-scan-type algorithm.

Recently, a run- and label-equivalence-based two-scan algorithm has been proposed.[9] In this algorithm, the run data, which are obtained during the first scan, are recorded in a queue and used for detecting the connectivity of runs in the later processing. Because this algorithm resolves connectivity between runs, for a given image, the number of provisional labels assigned by this algorithm is usually smaller than that assigned by other conventional label-equivalence-based labeling algorithms. This reduces the computation cost required for resolving label equivalences and that for checking the provisional labels in the mask. Therefore, it was demonstrated that this algorithm was very efficient for various images.

In this paper, we present a run- and label-equivalence-based one-and-a-half-scan algorithm, which is an improved version of the above run-based two-scan algorithm (referred to as the *previous algorithm* hereafter for convenience).

Unlike the previous algorithm, which assigns a provisional label to each object pixel as do all other conventional label-equivalence-based algorithms, and uses run data only for assigning provisional labels and resolving label equivalences, our algorithm assigns a provisional label to each run, and also uses run data for labeling object pixels directly. Moreover, our algorithm scans background pixels in the given image only once and does not require relabeling of object pixels. In addition, our algorithm is applicable to run-length-encoded images directly, and it can produce contour data of connected components in an image easily and efficiently.

The rest of the paper is organized as follows: we review the previous algorithm in the next section, and introduce our algorithm in Sec. 3. In Sec. 4, we show the experimental results to demonstrate the efficiency of our algorithm with many long runs and/or a small number of object pixels. We provide a discussion in Sec. 5 and give our concluding remarks in Sec. 6.

2. The Previous Algorithm

For an $N \times M$-size binary image, we use $p[y \times N + x]$ to denote the pixel value at (x, y) in the image, where $0 \leq x \leq N - 1$ and $0 \leq y \leq M - 1$. We assume that the object pixels and background pixels in a given image are represented by 1 and 0, respectively. As in most labeling algorithms, we assume that all pixels on the edges of an image are background pixels, and only consider eight-connectivity.

A *run* is a block of contiguous object pixels in a row. A run from $p[s]$ to $p[e]$ $(s \leq e)$ is described by $r(s, e)$. Let $r(s, e)$ be the current run in the raster scan. Among the runs that were scanned before the current run, a run $r(u, v)$ (in the row immediately above the row that $r(s, e)$ lies in) such that one of its pixels occurs between $p[s - N - 1]$ and $p[e - N + 1]$, i.e. if $u \leq e - N + 1$ and $v \geq s - N - 1$, is connected to $r(s, e)$ with eight-connectivity, as shown in Fig. 1.

The previous algorithm is a two-scan algorithm. In this algorithm, at any point in the first scan, all equivalent labels found so far are combined in a set, called *equivalent label set*, where the smallest label is referred to the representative label. The corresponding relation of a provisional label and its representative label is recorded in a table, called *the representative table*. For convenience, we use $S(t)$ for the set of provisional labels with t as the representative label, and $r\_label[a]$ to represent the representative label of provisional label a. In this way, for any provisional label f in provisional label set $S(t)$, we have $r\_label[f] = t$.

In the first scan, from $i = 0$, the algorithm scans pixel $p(i)$ one by one in the given image in the raster scan direction. When a new run $r(s, e)$ is found, the run data is recorded. At the same time, the eight-connected area with the current run in the above row is detected. If there is no run eight-connected with the current run in the row above the scan row, the current run belongs to a new connected component not found so far. All pixels in the current run are assigned a new label l, the provisional label set corresponding to the connected component, i.e. the current run, is established as $S(l) = \{l\}$, and the representative label of l is set to itself, i.e. $r[l] = l$. Moreover, l increases by 1 for consecutive processing.

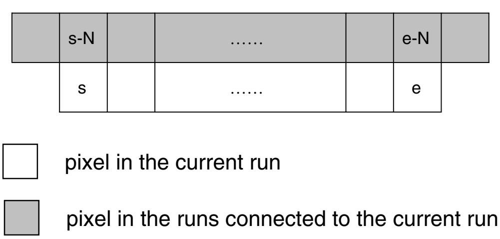


Fig. 1. The range for checking the eight-connectivity of the current run $r(s, e)$ and the runs scanned before the current run.

On the other hand, if there are some runs, say, $r_1, \ldots, r_n$, connected to $r(s,e)$ in the row above the scan row, then $r_1, \ldots, r_n$, and $r(s,e)$ belong to the same connected component. Suppose that $l_1, \ldots, l_n$ are the provisional labels assigned to $r_1, \ldots, r_n$ respectively, and $S(u_1), \ldots, S(u_n)$ are the equivalent label sets containing $l_1, \ldots, l_n$ respectively, then all provisional labels in $S(u_1), \ldots, S(u_n)$ are equivalent labels. Therefore, $S(u_1), \ldots, S(u_n)$ are merged to $S(u)$, where u is the smallest label among $u_1, \ldots$, and u_n. Moreover, all object pixels in the current run $r(s,e)$ are assigned the same provisional label that was assigned to the object pixels of the leftmost one of such runs, i.e. l_1. Moreover, after processing $r(s,e)$, the algorithm removes all data of runs from the queue that end before or at $p[e-N]$, because such runs cannot be connected with any coming run, and therefore are useless for further connectivity detection.

When the first scan finishes, all provisional labels that were assigned to a connected component in the given image will be combined with a common and unique representative label. During the second scan, the provisional label of each object pixel is relabeled by the representative label of that provisional label. If we set $r\_label[0] = 0$ in advance, where $r = r\_label[l]$ means that the representative label of the provisional label l is r, this relabeling process can be completed by the following simple operation[a]

$$\begin{aligned} &\text{for}(i = N; i < N \times (M-1); i{+}{+}) \\ &\qquad p[i] = r\_label[p[i]]; \\ &\text{end of for} \end{aligned} \tag{1}$$

3. Our Algorithm

As described above, the previous algorithm assigns a provisional label to each object pixel, and it uses only run data for assigning provisional labels and resolving label equivalences: after processing run $r(s,e)$, it discards all data of runs that end before or at $p[e-N]$, because such runs are not connected to any coming unprocessed run. Moreover, after resolving label equivalences, the algorithm needs to make another scan to assign a final label to each object pixel.

In comparison, our algorithm, in addition to assigning provisional labels and resolving label equivalences, also uses run data for avoiding relabeling object pixels.

[a] From experimental results, using the operations shown in formula (1) for relabeling, where all pixels are relabeled consecutively (although the value of background pixels is unchanged), the relabeling is more efficient than using the following operations, where only object pixels are relabeled (usually, pixels are processed intermittently):

$$\begin{aligned} &\text{for}(i = N; i < N \times (M-1); i{+}{+}) \\ &\qquad \text{if}(p[i]) \\ &\qquad\qquad p[i] = r\_label[p[i]]; \\ &\qquad \text{end of if} \\ &\text{end of for} \end{aligned}$$

Because all object pixels of a run obviously belong to the same connected component, and by labeling, they should be assigned the same label, instead of assigning a provisional label to each object pixel, we can assign a provisional label to each run. Thus, after all label equivalences between runs are resolved, all runs belonging to a connected component will have the same representative label; then, by use of the recorded run data, all object pixels of a run are assigned the same label of the run directly without scanning background pixels again. The pseudo code of the first scan of our algorithm is shown in Fig. 2, where $run[y]$ indicates the provisional label of the yth run, $resolve(p, q)$ means the merging processing of the equivalent label set $\mathcal{S}(p)$ and $\mathcal{S}(q)$, the detail of which can be found in Ref. 9. Moreover, n is the variable for recording run data, l for provisionally labeling on runs, and j for searching the runs that are connected with the current run.

By initializing n, l and j to 1 (*Line 1*), from $i = N$, our algorithm scans pixel $p[i]$ one by one in the given image in the raster scan direction until $i >= N \times (M - 1)$

```
1 : n = 1, l = 1, j = 1;
2 : for(i = N; i < N × (M − 1); i++)
3 :     if(p(i))
4 :         run_s[n] = i;
5 :         i++;
6 :         while(p(i))
7 :             i++;
8 :         end of while
9 :         run_e[n] = i − 1;
10 :        while(run_e[j] < run_s[n] − N − 1)
11 :            j++;
12 :        end of while
13 :        if(run_e[j] ≤ run_e[n] − N)
14 :            run[n] = run[j];
15 :            j++;
16 :            while(run_e[j] ≤ run_e[n] − N)
17 :                resolve(r_label[run[j]], r_label[run[n]]);
18 :                j++;
19 :            end of while
20 :            if(run_s[j] ≤ run_e[n] − N + 1)
21 :                resolve(r_label[run[j]], r_label[run[n]]);
22 :            end of if
23 :        else if(run_s[j] ≤ run_e[n] − N + 1)
24 :            run[n] = run[j];
25 :        else
26 :            run[n] = l;
27 :            r_label[l] = l;
28 :            l++;
29 :        end of if
30 :    end of if
31 :    n++;
32 : end of for
```

Fig. 2. The pseudo code of the first scan of our algorithm.

(*Line 2*). When a new run $r(s, e)$ is found, the run data are recorded by $run\_s[n] = s$ and $run\_e[n] = e$ (from *Line 3* to *Line 9*).

For the current run $r(s, e)$, by increasing j, we pass all recorded runs in the above row that end before $p[s - N - 1]$ (if any) (which are not connected to $r(s, e)$) (from *Line 10* to *Line 12*). Then, we check whether the next recorded run, say, r_t, ends before/at $p[e - N]$. If it does, we assign the current run the same provisional label of the run (*Line 14*), and for each other consecutive recorded run $run[j]$ that ends before/at $p[e - N]$, we merge the equivalent label set $S(r\_label[run[j]])$ and $S(r\_label[run[n]])$ (from *Line 16* to *Line 19*). Further, we check whether the next recorded run starts before at $p[e - N + 1]$. If it does, we merge the equivalent label sets $S(r\_label[run[j]])$ and $S(r\_label[run[n]])$ (from *Line 20* to *Line 22*). On the other hand, if r_t does not end before/at $p[e - N]$, we check whether it starts before/at $p[e - N + 1]$ (*Line 23*). If it does, we assign the current run the same provisional label of the run (*Line 24*). Otherwise, we assign the current run a new provisional label l (*Line 26*), set the representative label of l to itself (*Line 27*), and increase l by 1 for provisionally labeling the next run.

After processing the current run, n increases by 1 for recording and processing the next run (*Line 31*).

Notice that after processing the current run, the value of j corresponds to the first run that ends at/after $p(e - N + 1)$. Thus, for processing the next current run $r(s', e')$, where $s' > e + 1$, because any recorded run $run[j']$ such that $j' < j$ ends before $p(e - N)$, it is impossible to be connected with the run $r(s', e')$. Therefore, for detecting runs connected with the run $r(s', e')$, we can start from the jth run.

After the scan, the starting points and end points of all runs are recorded in $run\_s[\ ]$ and $run\_e[\ ]$, respectively, and all runs that belong to a connected component will have the same representative label. Then, using the recorded run data, we assign the representative label corresponding to a run to all object pixels of the run directly without scanning any background pixels again. This work can be completed easily by the following operations:

$$
\begin{aligned}
&\text{for}(i = 1; i < n; i{+}{+})\\
&\qquad \text{for}(j = run\_s[i]; j \le run\_e[i]; j{+}{+})\\
&\qquad\quad p[j] = r\_label[run[i]];\\
&\qquad \text{end of for}\\
&\text{end of for}
\end{aligned}
\tag{2}
$$

Obviously, no relabeling is executed in our algorithm, and background pixels are scanned only once.

4. Experimental Results

Our algorithm can be implemented similarly as the previous algorithm is done, except for recording all run data. Because the maximum number of runs in an $N \times M$-size image is $N \times M/2$, we can use one $N \times M/2$-sized 1D array to record the number of runs, and two $N \times M/2$-sized 1D arrays to record the starting points and end points of all runs.

We implemented our algorithm with the C language on a PC-based workstation (Intel Pentium Duo 930 3.0 GHz + 3.0 GHz CPUs, 2 GB Memory, Mandriva Linux OS). All execution times shown in this section were obtained by using one core.

Images used for testing included four types: artificial images, natural images, texture images and medical images. All images are 512×512 pixels in size.

The artificial images contain the following four type images:

(1) Specialized-pattern image set
There are stair-like, spiral-like, saw-tooth-like, checker-board-like, and honey-comb-like connected-component images[39];

(2) Ising image set[19]
190 images are made of samples of the Ising model $p(\sigma) \propto \exp(K \sum \delta[\sigma_i, \sigma_j])$, where $K = J/k_B T$, ten samples per value of K, ranging from 0.05 to 0.95 in steps of 0.05.[19] Such set of images can be considered as a parametric modelization of natural textures and satellite-like imagery. The densities of these images range from 49% to 92%.

(3) Overlapped-block image set[20]
This set is composed of images with a random distribution of 50 square blocks of object pixels, where overlap of blocks is allowed, with block size ranging from 5×5 to 100×100 in steps of 5, 10,000 different images for each block size. The densities of these images range from 0.5% to 75.8%.

(4) Uniform noise image set
41 noise images were generated by thresholding the images containing uniform random noise, where the value of a pixel is between 0 and 1000, with 41 different threshold values from 0 to 1000 with a step of 25. Because connected components in such noise images have complicated geometrical shapes and complex connectivity, severe evaluations of labeling algorithms can be performed with these images. The densities of these images range from 0.5% to 99.2%.

On the other hand, 50 natural images, including landscape, aerial, fingerprint, portrait, still-life, snapshot, and text images, which were obtained from the Standard Image Database (SIDBA) developed by the University of Tokyo[b] and the image database of the University of Southern California,[c] were used for realistic testing of labeling algorithms. In addition, seven texture images, which were downloaded from the Columbia-Utrecht Reflectance and Texture Database,[d] and 25 medical images obtained from a medical image database of The University of Chicago were used for

[b] http://sampl.ece.ohio-state.edu/data/stills/sidba/index.htm

[c] http://sipi.usc.edu/database/

[d] http://www1.cs.columbia.edu/CAVE/software/curet/index.php

Table 1. The summary of the classes of conventional algorithms used in comparisons.

Algorithm	Class	Reference
CT	Contour-tracing-based algorithm	Ref. 2
SAUF	Label-equivalence-based two-scan	Ref. 42
LTS	Label-equivalence-based two-scan	Ref. 10
HO	Label-propagation-based algorithm	Ref. 20
RTS	Run- and label-equivalence-based two-scan	Ref. 9

testing. All of these images were transformed into binary images by means of Otsu's threshold selection method.[27]

We compared our algorithm with the newest representative labeling algorithms in each algorithm class as follows: the Contour-Tracing (CT) labeling algorithm proposed in Ref. 2, the Scan plus Array-based Union-Find (SAUF) labeling algorithm proposed in Ref. 42, the Linear-time Two-Scan (LTS) labeling algorithm proposed in Ref. 10, the Hybrid Object (HO) labeling algorithm proposed in Ref. 20, and the Run-based Two-Scan (RTS) labeling algorithm proposed in Ref. 9. The algorithm classes of the conventional algorithms are summarized in Table 1. The program of the CT algorithm was downloaded from the authors' website at http://dar.iis.sinica.deu.tw/Download, and all other codes were provided by their authors, i.e. all implementations were done by the original authors. All algorithms were implemented in C language and compiled by GNU C complier version 4.2.3.

The Ising images were used for testing the execution times versus the value of K. The results are shown in Fig. 3, where for each K, the time is the average of the running times on the corresponding ten sample images with K, and the

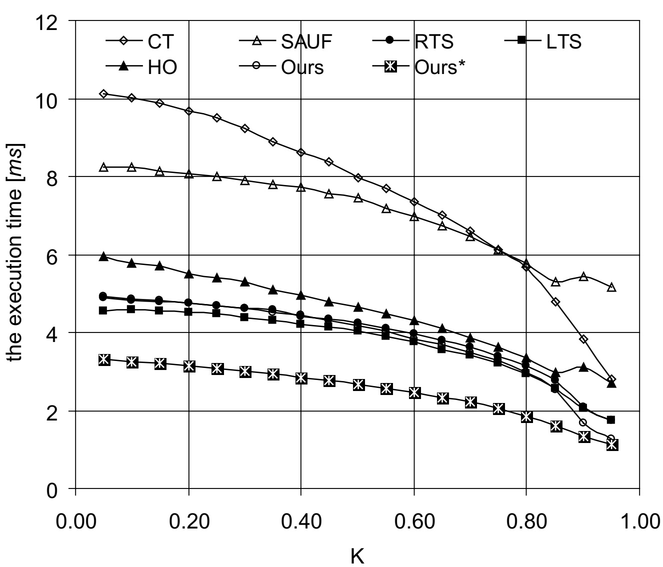


Fig. 3. Execution time versus the value of K.

0	1	2	3	4	5	6
0	0	0	0	0	0	0
0	1	0	0	1	0	0
0	1	1	1	1	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	1	1	0	1	0
0	0	0	0	0	0	0

Fig. 4. An example of a 7×7 binary image: its run-length encoding is 02114411400223550.

symbol $*$ indicates the results of our algorithm when images are given in a run-length-encoding format. In our implementation, the run-length encoding is defined as follows: a row in an image is encoded by: $n, s_1, e_1, \ldots, s_n, e_n$, where n is the number of runs in the row, s_i $(1 \leq i \leq n)$ is the starting pixel location of the ith run in the row, and e_i is the ending pixel location of the ith run in the row. For example, the binary image shown in Fig. 4 is encoded as 02114411400223550.

The overlapped-block images were used for testing the execution time versus the block size. The results are shown in Fig. 5, where for each block size, the time is the average of the running times on the corresponding 10,000 different images.

The noise images were used for testing the execution times versus the density (i.e. the number of object pixels divided by the total number of pixels in the image) in an image, where the times were obtained by averaging the execution times for 5000 runs.

Specialized-pattern images, natural images, medical images, and texture images (i.e. all real images) were used for testing the maximum, mean, and minimum

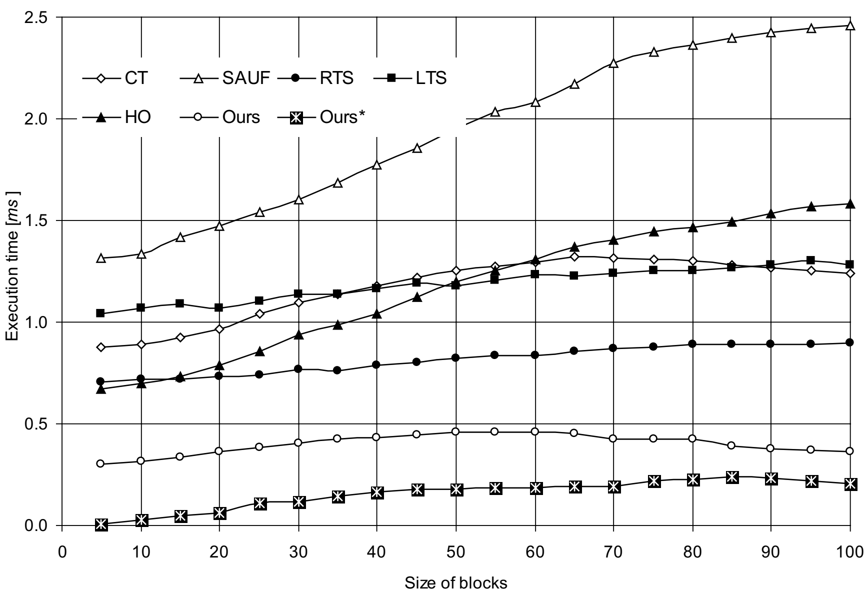


Fig. 5. Execution time versus the block size.

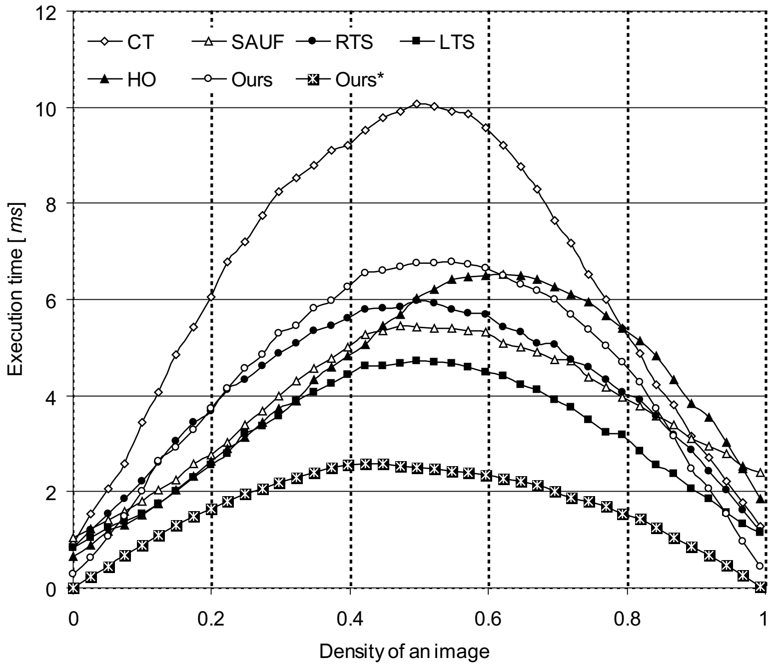


Fig. 6. Execution time versus the object pixel density in an image.

execution times of labeling algorithms. The results are shown in Table 2, where the times were obtained by averaging of the execution times for 5000 runs.

The number of images for which our algorithm was not faster than each of the conventional algorithms are shown in Table 3. In the case where an image was given in run-length encoding, our algorithm was the fastest of all.

Table 2. Comparison of execution times [ms] for various kinds of images.

Image Type		#CC	CT	SAUF	HO	RTS	LTS	Ours	Ours*
Special	max.	16,129	7.51	2.28	4.91	2.54	1.60	2.21	1.18
	mean	4096	3.89	1.22	1.92	1.44	0.85	1.1	0.59
	min.	1	1.18	0.33	0.42	0.85	0.28	0.56	0.24
Natural	max.	2660	4.28	3.23	3.37	2.90	2.33	2.65	0.77
	mean	847	2.34	2.07	2.04	1.71	1.49	1.27	0.30
	min.	19	1.13	1.29	1.13	0.95	0.95	0.26	0.04
Textural	max.	1525	3.69	2.87	2.70	2.54	2.03	2.60	1.21
	mean	281	2.66	2.57	2.10	1.67	1.56	1.58	0.93
	min.	20	1.58	2.37	1.56	1.17	1.14	0.57	0.36
Medical	max.	372	2.59	2.28	1.63	1.71	1.47	1.54	0.27
	mean	83	1.92	1.89	1.25	1.37	1.23	0.99	0.14
	min.	1	1.52	1.54	0.92	1.19	0.93	0.77	0.09

Note: #CC, number of connected components.

Table 3. The number of images for which our algorithm was not faster than each of the conventional algorithms.

Image Type	Total	CT	SAUF	HO	RTS	LTS
Special	5	0	3	2	0	3
Natural	50	0	0	4	0	13
Medical	25	0	0	1	0	1
Textural	7	0	0	0	1	2
Noise	41	0	31	22	28	35
Ising	190	0	0	2	48	154
Block	200,000	0	0	0	0	0

The results for six representative images are illustrated in Fig. 7, where the object pixels are displayed in black. The various characteristics of the six images are shown in Table 4, where N_o, N_r and L_r are the number of object pixels, the number of runs, and the average length of runs, respectively.

5. Discussion

5.1. *The complexity of our algorithm*

To complete labeling, our algorithm performs the following procedures:

(1) Record the start points and the end points of runs and assign provisional labels to runs in first scan;
(2) Create an equivalent label set and set the representative label for each new provisional label;
(3) Resolve label equivalences;
(4) Assign final labels to object pixels.

For an $N \times M$-pixel image, both the maximum number of runs and the maximum number of object pixels have an order of $\mathcal{O}(N \times M)$. Therefore, the orders of procedures (1) and (4) are $\mathcal{O}(N \times M)$. In procedure (2), only constant operations are executed for each new provisional label. Because the order of the maximum number of provisional labels is $\mathcal{O}(N \times M)$, the order of procedure (2) is also $\mathcal{O}(N \times M)$. On the other hand, the method for resolving label equivalences in our algorithm is exactly the same as in the LTS algorithm. According to Ref. 10, the order is also $\mathcal{O}(N \times M)$. Thus, the order of our algorithm is $\mathcal{O}(N \times M)$, i.e. the running time of our algorithm is linear versus the number of pixels in an image.

We used 32×32, 64×64, 128×128, 256×256, and 512×512 size uniform noise images to test the linearity of the algorithms, where for each of 32×32, 64×64, 128×128, and 256×256 size, 41 noise images were generated in the same way that the 512×512 size noise images were introduced above. The maximum and average execution times versus the number of pixels in an image are shown in Figs. 8(a) and 8(b), respectively. As we see, the maximum and average execution times of each of these algorithms are linear versus image size.

CT: 3.37 SAUF: 2.10
HO: 1.82 RTS: 2.09
LTS: 1.70 Ours: 1.93
Ours : 0.59

(a)

CT: 3.25 SAUF: 2.56
HO: 2.37 RTS: 2.18
LTS: 1.82 Ours: 1.88
Ours : 0.38

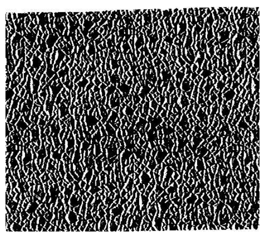
(b)

CT: 2.18 SAUF: 2.23
HO: 1.90 RTS: 1.60
LTS: 1.40 Ours: 1.04
Ours : 0.18

(c)

CT: 1.13 SAUF: 1.29
HO: 0.96 RTS: 0.95
LTS: 0.99 Ours: 0.26
Ours : 0.04

(d)



CT: 3.69 SAUF: 2.87
HO: 2.71 RTS: 2.54
LTS: 2.03 Ours: 2.60
Ours : 1.21

(e)

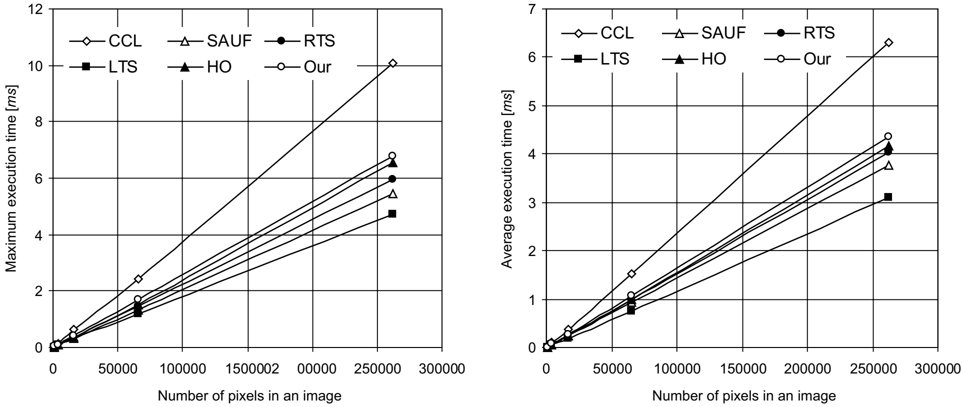
CT: 1.98 SAUF: 2.01
HO: 1.32 RTS: 1.44
LTS: 1.32 Ours: 1.06
Ours : 0.16

(f)

Fig. 7. Execution time [*ms*] of labeling algorithms for the six representative images: (a) a text image; (b) a fingerprint image; (c) a portrait image; (d) a snapshot image; (e) a texture image; and (f) a medical image.

Table 4. Characteristics of images shown in Fig. 7.

Image	N_o	N_r	L_r	Density
(a)	38,950	17,916	2.17	0.15
(b)	142,232	15,597	9.12	0.54
(c)	159,154	5737	27.74	0.61
(d)	46,326	1370	33.81	0.18
(e)	142,876	20,855	6.84	0.55
(f)	110,266	5336	20.66	0.42



(a) (b)

Fig. 8. Linearity of the execution time versus image size: (a) maximum execution time and (b) average execution time.

5.2. *Adapting our algorithm for four-connectivity*

Our algorithm can be easily adapted for four-connectivity. The only thing that we need to do is change the range for checking connectivity. In the case of four-connectivity, for the current run $r(s, e)$, the range for finding the runs connecting with $r(s, e)$ from processed runs is shown in Fig. 9.

Thus, for $N \times M$-size binary images, a processed run $r(u, v)$ (in the row immediately above the row that $r(s, e)$ lies in) such that one of its pixels occurs between $p[s - N]$ and $p[e - N]$, i.e. if $u \leq e - N$ and $v \geq s - N$, is connected to $r(s, e)$ with the four-connectivity.

The work for recording and resolving label equivalences, and relabeling object pixels can be done exactly the same way as in the case for eight-connectivity.

5.3. *Application to run-length-encoding images*

Run-length encoding is used in image compression and in image representation for relatively simple graphic images such as icons, line drawings, and animations.[14,23,28,34,36,40] Run-length-based encoding is also used in signal transmission between fax machines.[44,45]

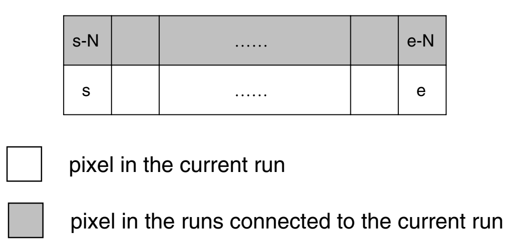


Fig. 9. The range for checking the four-connectivity of the current run $r(s, e)$ and the processed runs.

When images are given in run-length-based encoding (compressed) format, conventional pixel-based labeling algorithms require a decoding (decompressing) process prior to labeling; therefore, the whole running time will increase.

Two run-based connected-component-labeling algorithms have been proposed for run-length-encoding images. One, proposed in Ref. 33, is an improvement of the propagation-type algorithm proposed in Ref. 32 by use of block sorting and tracing of runs to realize label propagation for connected runs. It performs a searching step and a propagation step iteratively on the run data. In the searching step, the image is scanned until an unlabeled run is found; then the run is assigned a new label. In the propagation step, the new label propagates to neighbor runs above or below the current row until all of the runs that belong to the connected component are labeled by the same one.

The other algorithm, proposed in Ref. 37, is a run-based contour-tracing algorithm. It records run data in the order of raster scanning and performs labeling by local and global run tracing (contour tracing). In the local run tracking, the next run for run tracking along the contour is determined. In the global run tracing, the run is labeled by tracing the contour at each run.

Both of the above two run-based labeling algorithms belong to searching and label-propagation algorithms; thus, they require preprocessing for formatting run data. According to the experimental results shown in Ref. 2, the CT algorithm was faster than the above two algorithms. The experimental results shown in Sec. 4 demonstrated that our algorithm was more efficient than the CT algorithm. Therefore, our algorithm should be faster than the above two algorithms.

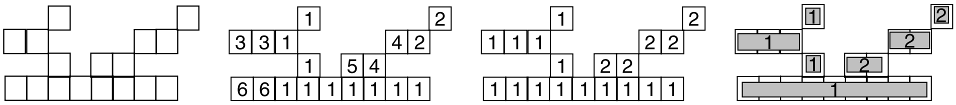
Regentova *et al.*[3] presented a two-scan label-equivalence-based labeling algorithm to reduce operations for connected-component detection by processing run-length-encoded images directly. This algorithm detects connectivity among runs in a similar way as does our algorithm, and it uses a conventional two-scan algorithm to resolve label equivalences.[e]

Our algorithm can be extended easily to labeling of run-length-encoded images directly and efficiently, as shown in Sec. 4. The connectivity detection in our algorithm and in Regentova's algorithm is similar for run-length-encoded images. From the experimental results shown in Ref. 9, the method for resolving label equivalences used in our algorithm was faster by at least a factor of 18.5 than that used in any other conventional two-scan algorithms proposed before Ref. 3 was published. Therefore, our algorithm should be much faster than Regentova's algorithm.

5.4. *Comparison with conventional label-equivalence-based raster-scan algorithms*

There are two main differences between our algorithm and conventional label-equivalence-based raster-scan algorithms: (1) all conventional label-equivalence-based raster-scan algorithms scan all pixels of the given image at least twice, whereas

[e] The paper did not explain which algorithm was used.



(a) (b) (c) (d)

Fig. 10. Labeling results after the first scan by conventional algorithms and our algorithm: (a) an original connected component; (b) by the SAUF algorithm and the LTS algorithm (as well as by all conventional label-equivalence-based algorithms except for the RTS algorithm); (c) by the RTS algorithm; and (d) by our algorithm.

our algorithm scans background pixels once, and object pixels twice; (2) all conventional label-equivalence-based raster-scan algorithms assign a provisional label to each object pixel in the first scan and relabel it in the later scan(s) (i.e. the value of each object pixel is rewritten at least once[f]), our algorithm does not assign a provisional label to any object pixel, but assigns a final label directly. Thus, no relabeling is necessary (see Fig. 10).

When the average length of runs in a given image is large, our algorithm will be more efficient than conventional label-equivalence-based algorithms because the time for relabeling of object pixels in conventional label-equivalence-based algorithms is large, whereas our algorithm does not need to do such work. On the other hand, when the number of object pixels in a given image is small (i.e. the majority of pixels are background pixels), our algorithm will also be more efficient than conventional label-equivalence-based algorithms, because background pixels are processed at least twice in conventional label-equivalence-based algorithms, but only once in our algorithm.

However, when the average length of runs is small and the number of object pixels is large (i.e. there are a large number of short runs in an image), the computation cost required for assigning labels to object pixels by using the run data shown in formula (2) (which processes runs one by one, similar to processing of a large number of data stored in a lot of arrays) will be larger than that of relabeling of object pixels (e.g. all pixels are processed consecutively in an array, as shown in formula (1)). For such cases, our algorithm will not be as efficient as for the other cases discussed above.

The above analyses are consistent with the experimental results shown in Sec. 4. For the Ising images, on the one hand, the connectivities of connected components in these images are quite complicated, and on the other hand, because the densities of all these images are larger than 49%, the average length of runs in each image is relatively large. Moreover, the densities of Ising images increase with the value of K. That is, the average length of runs in an Ising image increases with K, as shown in Fig. 11. Thus, for the images such that the value of K is smaller than 0.7, our algorithm was a little slower than the LTS algorithm, but faster than the others, and for the images with K larger than 0.7, our algorithm was the fastest.

[f] In implementation, as explained in footnote *a*, by using the processing given in formula (2), where the value of each background pixel is also rewritten in the second scan (although the value of the background pixels does not change) is more efficient than rewriting the values of object pixels only. In this case, the values of all pixels are rewritten once.

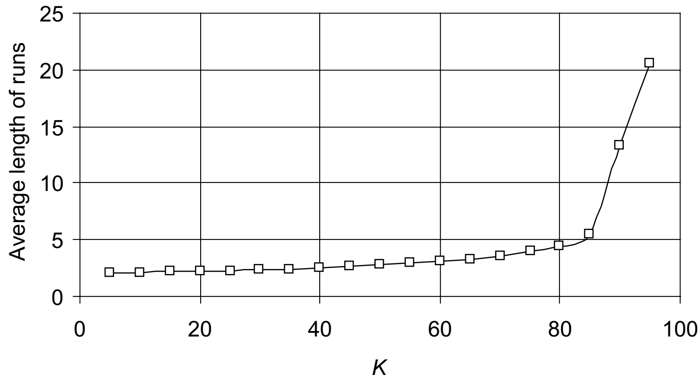


Fig. 11. The average length of runs versus K in the Ising images.

For overlapped-block images, when the block size is small, the number of object pixels is also small (i.e. the number of background pixels is large), on the other hand, when the block size is large, the average length of runs in an image is also large.[g] Both are favorite cases for our algorithm — it was the fastest for all such images.

For the uniform random set given there, when the number of object pixels is small (i.e. the average density is low) or the average length of runs is large (i.e. the density is high and the number of runs is small) in an image, our algorithm was faster than other conventional algorithms. However, for the rest of the set, our algorithm was not as efficient as the SAUF algorithm and the LTS algorithm, which is even worse than the previous algorithm.

It is worth mentioning that: (1) the average execution times of our algorithm for the various types of real images used in Sec. 4 are smaller than those of other conventional algorithms; (2) for almost all of the real images used in Sec. 4, i.e. the images except the artificial images, our algorithm was faster than the SAUF algorithm and the RTS algorithm; (3) for most natural images and textural images, as well as almost all of the medical images, our algorithm was faster than the LTS algorithm (see Table 3).

On the other hand, for the images shown in Figs. 7(a), 7(b) and 7(e), which have many short runs (i.e. many complex connected components) our algorithm was slower than the LTS algorithm, but for the images shown in Figs. 7(c), 7(d) and 7(f), which have many long runs (i.e. many simple connected components), our algorithm was the fastest of all algorithms.

When images are given in run-length encoding, conventional raster-scan algorithms require additional time for decoding; thus, they will take more time with respect to the cases where images are given in a non-run-length-encoded format such as the PBM format. In comparison, as shown in Sec. 4, when images were given in run-length encoding, our algorithm was much faster with respect to the case where

[g] For a $B \times B$ overlapped-block image, the average length of runs in the image is equal to or larger than B.

images were given in the PBM format. In fact, when images were given in run-length encoding, for each image used in Sec. 4, our algorithm was the fastest of all labeling algorithms, and on average, our algorithm was 4.3 times faster with respect to the case where the images were given in the PBM format.

However, for labeling $N \times M$-size images, the memory space necessary for the SAUF algorithm, the RTS algorithm, the LTS algorithm, and our algorithm is $N \times M/4$, $3 \times N \times M/4 + 2 \times N/2$, $3 \times N \times M/4$, and $9 \times N \times M/4$, respectively. Thus, our algorithm requires more memory than the others.

5.5. *Comparison with searching and label propagation algorithms*

All searching and label-propagation algorithms access an image in an irregular way. Therefore, they are not suitable for pipeline processing, parallel implementation,[6,18,25,41] or systolic-array implementations.[26] Moreover, all such algorithms usually process the background pixels around object pixels twice, the other background pixels once, and all object pixels at least twice (maybe more).

In comparison, our algorithm processes an image in the raster-scan order; therefore, it is suitable for hardware implementation, pipeline processing, and parallel implementation. Moreover, our algorithm processes all background pixels only once and all object pixels exactly twice (i.e. one scan plus direct assignment of the final labels to object pixels). As demonstrated in the experimental results in Sec. 4, our algorithm was faster than the CT algorithm for all real images, and the HO algorithm for almost all of real images.

Moreover, similar to the CT algorithm and the HO algorithm, our algorithm can also output contour data of connected components in the given image easily during labeling. For each run, the starting and end pixels of the run are obviously contour pixels, and an inner object pixel of the run is a contour pixel if and only if its upper or lower pixel is a background pixel. Obviously, there is no other contour pixel. Thus, the work for outputting contour pixels in an $N \times M$-size image can be completed (instead of the processes shown in formula (2)) during labeling of object pixels with the following processes, where $c[\ ]$ is the output image for contour data:

```
for(i = 1; i < n; i++)
    s = run_s[i], e = run_e[i];
    l = r_label[run[i]];
    c[s] = l, c[e] = l;
    p[s] = l, p[e] = l;
    for(j = s + 1; j < e; j++)
        p[j] = l;
        if(p[j − N] < 1 || p[j + N] < 1)
            c[j] = l;
        end of if
    end of for
end of for
```

By the above processing, all and only all contour pixels in each connected component will be marked as the same label corresponding to the connected component in the output contour image $c[\ ]$.

For comparison, we used the noise images, the Ising images, and the overlapped block images to test the performances of the CT-labeling algorithm, the HO algorithm and our algorithm with outputting contours. The results are shown in Figs. 12–14, respectively, where *CT-con*, *HO-con* and *our-con* denote the data on ordinary format images obtained by the CT-labeling algorithm, the HO algorithm and our algorithm, *CT-con**, *HO-con** and *our-con** for those on run-length-encoding-format images, respectively.

The results demonstrate that, (1) for the noise images, when images were given in ordinary format, for images with very low density (close to 0), very high density (close to 1), and those with density between 0.47 and 0.79, our algorithm was fastest; for images with density between 0.03 and 0.47, the HO algorithm was fastest; and other images, the CT algorithm was fastest. On the other hand, when images were given in run-length-encoding format, the HO algorithm was faster than the CT algorithm on images with density smaller than 0.74, and our algorithm was much faster than the other two algorithms for all images; (2) for all Ising images and overlapped block images, our algorirhm was faster than both the CT algorithm and the HO algorithm.

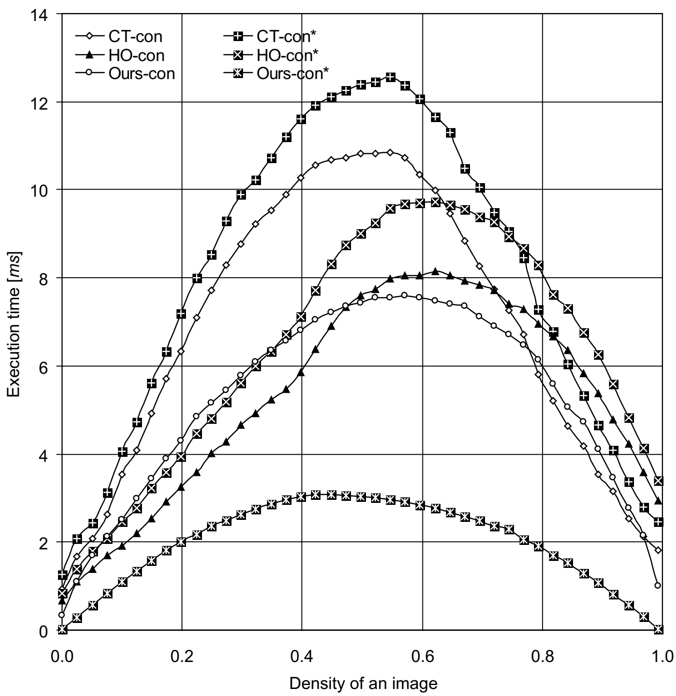


Fig. 12. Execution time with outputting contours for the noise images.

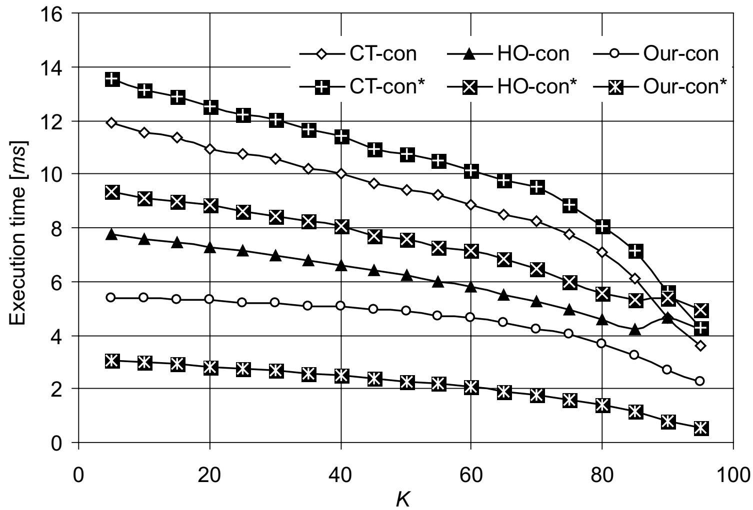


Fig. 13. Execution time with outputting contours for the Ising images.

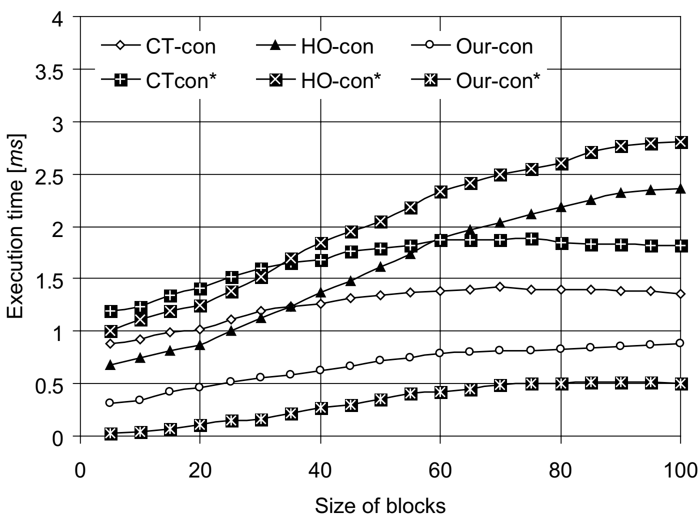


Fig. 14. Execution time with outputting contours for the overlapped block images.

Moreover, for all natural images, medical images, texture images, and the specialized pattern images used in Sec. 4, the experimental results showed that our-con algorithm was faster than the CT-con algorithm and the HO-con algorithm. Moreover, when the images are given in run-length encoding, our-con algorithm was much faster than the CT-con algorithm and the HO algorithm for all images.

On the other hand, (1) the CT algorithm and the HO algorithm do not require additional memory, but our algorithm requires three $N \times M/4$-sized and three

$N \times M/2$-sized 1D arrays for implementation for labeling $N \times M$-size images; (2) label propagation algorithms usually generate consecutive labels naturally, but our algorithm, the same as other label-equivalence-based algorithms, needs an additional processing for generating consecutive labels[10]; (3) label propagation algorithms are able to characterize the objects at the same time when being labeled, whereas all label-equivalence-based labeling algorithms, including our algorithm, need an additional processing to do this work; (4) our algorithm cannot be used for labeling with region growing by similarity measures[21] (because our algorithm only checks connectivity at the ends of each run), whereas the work can be easily done with propagation algorithms (even pixel- and label-equivalence-based algorithms); (5) to achieve the best performance of our algorithm, we have to use the data segment for our data structures,[h] thus, our algorithm is not suitable for the case where the maximum size of images to be used is unknown.

6. Concluding Remarks

In this paper, we proposed a run- and label-equivalence-based one-and-a-half-scan labeling algorithm. Experimental results demonstrated that our algorithm was faster than any other labeling algorithms for images with many long runs or a small number of object pixels. For images with a lot of short runs, the LTS algorithm was the fastest. When images were given in run-length encoding, our algorithm was much faster than any other labeling algorithms.

For future work, we plan to extend our algorithm to include 3D connected-component labeling[12,17] for some 3D machine vision applications (e.g. Ref. 22), and to develop an algorithm for parallel architectures.

The dataset used in our test can be downloaded from http://www.aichi-pu.ac.jp/ist/~helifeng/ and the source codes of our algorithm can be obtained by contacting The University of Chicago's Office of Technology & Intellectual Property (http://www.ibridgenetwork.org/uctech/).

Acknowledgments

We thank the Managing Editor X. Jiang and the anonymous referees for their valuable comments that improved this paper greatly. We are grateful to J. Martin-Herrero for providing the HO program, the Ising image set and source codes for generating overlapped-block images as well as many other kind help. This paper was partially supported by the KAYAMORI Foundation of Informational Science Advancement and the Hibi Science Foundation, Japan.

[h] The experimental results showed that, with respect to the use of static allocation, the running time of our algorithm for an image might be three times longer if we dynamically allocated the memory for our data structures. However, the CT algorithm and the HO algorithm have no such problem, because it needs no additional data structures.

References

1. D. H. Ballard, *Computer Vision* (Englewood, Prentice-Hall, New Jersey: 1982).
2. F. Chang, C. J. Chen and C. J. Lu, A linear-time component-labeling algorithm using contour tracing technique, *Comput. Vis. Imag. Underst.* **93** (2004) 206–220.
3. E. Regentova, S. Latifi, S. Deng and D. Yao, An algorithm with reduced operations for connected components detection in ITU-T Group 3/4 coded images, *IEEE Trans. Patt. Anal. Mach. Intell.* **24**(8) (2002) 1039–1047.
4. R. C. Gonzalez and R. E. Woods, *Digital Image Processing* (Addison Wesley, 1992).
5. T. Gotoh, Y. Ohta, M. Yoshida and Y. Shirai, Component labeling algorithm for video rate processing, *Proc. SPIE*, Advances in Image Processing, Vol. 804 (1987), pp. 217–224.
6. R. M. Haralick, Some neighborhood operations, *Real Time/Parallel Computing Image Analysis* (Plenum Press, New York, 1981), pp. 11–35.
7. R. M. Haralick and L. G. Shapiro, *Computer and Robot Vision*, Vol. I (Addison-Wesley, Reading, MA, 1992), pp. 28–48.
8. A. Hashizume, R. Suzuki, H. Yokouchi *et al.*, An algorithm of automated RBC classification and its evaluation, *Bio Med. Engin.* **28**(1) (1990) 25–32.
9. L. He, Y. Chao and K. Suzuki, A run-based two-scan labeling algorithm, *IEEE Trans. Imag. Process.* **17**(5) (2008) 749–756.
10. L. He, Y. Chao, K. Suzuki and K. Wu, Fast connected component labeling, *Patt. Recogn.* **42** (2009) 1977–1987.
11. Y. Huang, T. Chang, Y. Chen and F. E. Sandnes, A back propagation based real-time license plate recognition system, *Int. J. Patt. Recogn. Artifi. Intell.* **22**(2) (2008) 233–251.
12. Q. Hu, G. Qian and W. L. Nowinski, Fast connected-component labeling in three-dimensional binary images based on iterative recursion, *Comput. Vis. Imag. Underst.* **99** (2005) 414–434.
13. X. Jiang and Y. Chen, *Facial Image Processing, Applied Pattern Recognition* (2008), pp. 29–48, http://dx.doi.org/10.1007/978-3-540-76831-9_2.
14. S. D. Kim, J. H. Lee and J. K. Kim, A new chain-coding algorithm for binary images using run-length codes, *Comput. Vis. Graph. Imag. Process.* **41**(1) (1988) 114–128.
15. M. Komeichi, Y. Ohta, T. Gotoh, T. Mima and M. Yoshida, Video-rate labeling processor, *Proc. SPIE*, Vol. 1027, *Image Processing II* (1988), pp. 69–76.
16. R. Lumia, L. Shapiro and O. Zungia, A new connected components algorithm for virtual memory computers, *Comput. Vis. Graph. Imag. Process.* **22**(2) (1983) 287–300.
17. R. Lumia, A new three-dimensional connected components algorithm, *Comput. Vis. Graph. Imag. Process.* **23**(2) (1983) 207–217.
18. M. Manohar and H. K. Ramapriyan, Connected component labeling of binary images on a mesh connected massively parallel processor, *Comput. Vis. Graph. Imag. Process.* **45**(2) (1989) 133–149.
19. J. Martin-Herrero, Hybrid cluster identification, *J. Phys. A: Math. Gen.* **37** (2004) 9377–9386.
20. J. Martin-Herrero, Hybrid object labelling in digital images, *Mach. Vision Appl.* **18**(1) (2007) 1–15.
21. J. Martin-Herrero, Comments on "A new operational method for estimating noise in hyperspectral images", *IEEE Geosci. Rem. Sens. Lett.* **5**(4) (2008) 705–709.
22. J. Martin-Herrero and C. Germain, Microstructure reconstruction of fibrous C/C composites from X-ray microtomography, *Carbon* **45**(6) (2007) 1242–1253.
23. G. Nagy, S. C. Seth and S. D. Stoddard, Document analysis with an expert system, *Proc. ACM Conf. Document Processing Systems* (1988), pp. 169–176.
24. S. Naoi, High-speed labeling method using adaptive variable window size for character shape feature, *IEEE Asian Conf. Computer Vision*, Vol. 1 (1995), pp. 408–411.

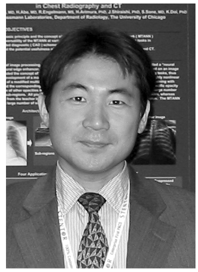
25. D. Nassimi and S. Sahani, Finding connected components and connected ones on a mesh connected parallel compute, *SIAM J. Comput.* **9**(4) (1980) 744–757.
26. C. J. Nicol, A systolic approach for real time connected component labeling, *Comput. Vis. Imag. Underst.* **61**(1) (1995) 17–31.
27. N. Otsu, A threshold selection method from gray-level histograms, *IEEE Trans. Syst. Man Cybern.* **9** (1979) 62–66.
28. C. S. Partridge, Method of skeletonizing a binary image using compressed run length data, US Patent 6058219, May 2000 (http://www.patentstorm.us/patents/6058219.html).
29. C. Ronsen and P. A. Denjiver, *Connected Components in Binary Images: The Detection Problem* (Research Studies Press, 1984).
30. A. Rosenfeld and J. L. Pfalts, Sequential operations in digital picture processing, *J. ACM* **13**(4) (1966) 471–494.
31. A. Rosenfeld, Connectivity in digital pictures, *J. ACM* **17**(1) (1970) 146–160.
32. A. Rosenfeld and A. C. Kak, *Digital Picture Processing*, Vol. 2, 2nd edn. (Academic Press, San Diego, CA, 1982).
33. 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, *Proc. 10th Int. Conf. Patt. Recogn.* (1990), pp. 655–658.
34. J. Shin, H. Hwang and S. Chien, Detecting fingerprint minutiae by run length encoding scheme, *Patt. Recogn.* **39**(6) (2006) 1140–1154.
35. Y. Shirai, Labeling connected regions, in *Three-Dimensional Computer Vision* (Springer-Verlag, 1987), pp. 86–89.
36. N. Shiraishi, Image data compression apparatus for compressing both binary image data and multiple, US Patent 6941023 (http://www.patentstorm.us/patents/6941023-claims.html), Sept. 2005.
37. K. Shoji and J. Miyamichi, *Connected Component Labeling in Binary Images by Run-Based Contour Tracing*, The Transactions of the Institute of Electronics, Information and Communication Engineers D-II, Vol. J83-D-II, No. 4: 1131–1139 (in Japanese).
38. S. N. Srihari and H. Srinivasan, Comparison of ROC and likelihood decision methods in automatic fingerprint verification, *Int. J. Patt. Recogn. Artif. Intell.* **22**(3) (2008) 535–553.
39. K. Suzuki, I. Horiba and N. Sugie, Linear-time connected-component labeling based on sequential local operations, *Comput. Vis. Imag. Underst.* **89** (2003) 1–23.
40. T. Tsuiki, T. Aoki and S. Kino, Image processing based on a runlength coding and its application to an intelligent facsimile, *Proc. Conf. Record, GLOBECOM'82*, pp. B6.5.1–B6.5.7, Nov. 1982.
41. K. B. Wang, T. L. Chia and Z. Chen, Parallel execution of a connected component labeling operation on a linear array architecture, *J. Inform. Sci. Engin.* **19** (2003) 353–370.
42. K. Wu, E. Otoo and K. Suzuki, Optimizing two-pass connected-component labeling algorithms, *Patt. Anal. Appl.* **12** (2009) 117–135.
43. X. D. Yang, Design of fast connected components hardware, *Proc. IEEE Conf. Computer Vision and Pattern Recognition* (Ann Arbor, MI, June 1988), pp. 937–944.
44. CCITT Recommendation T.4, *Standardization of Group 3 Facsimile Apparatus for Document Transmission*, Terminal Equipment and Protocols for Telematic Services, Vol. VII.3, Geneva, 1985.
45. CCITT Recommendation T.6, *Facsimile Coding Control Functions for Group 4 Facsimile Apparatus*, Terminal Equipment and Protocols for Telematic Services, Vol. VII, Fascicle, VII.3 Geneva, 1985.

Lifeng He received his B.E. degree from Northwest Institute of Light Industry, China, in 1982, the second B.E. degree from Xian Jiaotong University, China, in 1986, the M.S. and the Ph.D. degrees in AI and computer science from Nagoya Institute of Technology, Japan, in 1994 and 1997, respectively. He is an associate professor in Aichi Prefectural University, Japan and a guest professor in the Shaanxi University of Science and Technology, China. From 2006 to 2007, he works in the University of Chicago (USA) as a research associate.

His research interests include image processing, automated reasoning, and artificial intelligence.

Yuyan Chao received her B.E. degree from Northwest Institute of Light Industry, China, in 1984, and the M.S. and the Ph.D degrees from Nagoya University, Japan, in 1997 and 2000, respectively. From 2000 to 2002, she was a special foreign researcher of Japan Society for the Promotion of Science in Nagoya Institute of Technology. She is a professor in 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 engineering from Meijo University, Japan, in 1991 and 1993, respectively, and his PhD degree in engineering from Nagoya University, Japan, in 2001. From 1993 to 1997, he worked in Research & Development Center at Hitachi Medical Co. as Researcher. From 1997 to 2001, he worked in the Department of Applied Information Science and Technology at Aichi Prefectural University, Japan, as a faculty member. In 2001, he joined the Department of Radiology at the University of Chicago, as Research Associate, and then, he was promoted to Research Associate (Assistant Professor). Since 2006, he has been Assistant Professor of Radiology, Graduate Program in Medical Physics, and Comprehensive Cancer Center. He has published 190 papers (including 65 peer-reviewed journal papers). He is an inventor/co-inventor on 30 patents (including 11 granted patents). He was awarded 25 grants including an NIH R01 grant. He has been serving as the Editor-in-Chief and an Associate Editor of eight leading international journals, including *Medical Physics, International Journal of Biomedical Imaging, and Algorithms*. He has been serving as a referee for 35 international journals. He has been serving as an organizer and a program committee member of 30 international conferences. He has received numerous international awards, including a Paul C. Hodges Award, RSNA Certificate of Merit Awards, a CRF Young Investigator Award, a SPIE Honorable Mention Poster Award, and an IEEE Outstanding Member Award. He has been a Senior Member of IEEE since 2004. He has been served as the Senior Member Upgrade Chair, IEEE since 2008.