Pattern Recognition 70 (2017) 25–43

Contents lists available at ScienceDirect

Pattern Recognition

journal homepage: www.elsevier.com/locate/patcog

The connected-component labeling problem: A review of state-of-the-art algorithms

Lifeng He[a,b,*], Xiwei Ren[a], Qihang Gao[a], Xiao Zhao[a], Bin Yao[a], Yuyan Chao[c]

[a] Artificial Intelligence Institute, College of Electrical and Information Engineering, Shaanxi University of Science and Technology, Shaanxi 710021, PR China
[b] Faculty of Information Science and Technology, Aichi Prefectural University, Aichi 4801198, Japan
[c] Faculty of Environment, Information and Business, Nagoya Sangyo University, Aichi 4888711, Japan

ARTICLE INFO

Article history:
Received 10 June 2016
Revised 7 March 2017
Accepted 15 April 2017
Available online 22 April 2017

Keywords:
Connected-component labeling
Shape feature
Image analysis
Image understanding
Pattern recognition
Computer vision

ABSTRACT

This article addresses the connected-component labeling problem which consists in assigning a unique label to all pixels of each connected component (i.e., each object) in a binary image. Connected-component labeling is indispensable for distinguishing different objects in a binary image, and prerequisite for image analysis and object recognition in the image. Therefore, connected-component labeling is one of the most important processes for image analysis, image understanding, pattern recognition, and computer vision. In this article, we review state-of-the-art connected-component labeling algorithms presented in the last decade, explain the main strategies and algorithms, present their pseudo codes, and give experimental results in order to bring order of the algorithms. Moreover, we will also discuss parallel implementation and hardware implementation of connected-component labeling algorithms, extension for n-D images, and try to indicate future work on the connected component labeling problem.

1. Introduction

Following introduction in famous textbooks on digital image processing [1–3], for an $N \times N$-sized binary image,[1] the pixel at the coordinate (x, y), where $0 \le x \le N-1$ and $0 \le y \le N-1$, in the image is denoted as $b(x, y)$. When it is clear from the context, we also use $b(x, y)$ to denote the value of itself. Foreground pixels are also called object pixels. While not stated otherwise, we assume that the values of object pixels and background pixels are 1 and 0, respectively. Moreover, for convenience, we assume all pixels in the border of an image are background pixels.

For pixel $b(x, y)$, the four pixels $b(x-1, y)$, $b(x, y-1)$, $b(x+1, y)$, and $b(x, y+1)$ are called the *4-neighbors* of the pixel; the four-neighbors together with the four pixels $b(x-1, y-1)$, $b(x+1, y-1)$, $b(x-1, y+1)$, and $b(x+1, y+1)$ are called the *8-neighbors* of the pixel. Two object pixels p and q are said to be *8-connected* (*4-connected*) if there is a path which consists of object pixels a_1, a_2, ..., a_n such that $a_1 = p$ and $a_n = q$, and for all $1 \le i \le n-1$, a_i and a_{i+1} are 8-neighbor (4-neighbor) for each other. For example, object pixels p and q in Fig. 1(a) are 8-connected. An *8-connected (4-connected) component* in a binary image is the maximum set of object pixels in the image such that any of two pixels in the set are 8-connected (4-connected). A connected component is also called an *object*. For convenience, in this article, we will use connected component and object in exactly the same meaning. Moreover, because objects with 8-connectivity are more complicated than those with 4-connectivity, we will only consider 8-connectivity for objects. For example, there are four objects in Fig. 1(a).

For image analysis, image understanding, pattern recognition, and computer vision, we often change an image into a corresponding binary image, where pixels belonging to objects which we want to recognize are transfer to foreground pixels (object pixels) and all other pixels are transfer to background pixels. In order to distinguish different objects in a binary image, connected-component labeling is an indispensable operation, which consists in assigning a unique label to all pixels of each object in the image. After labeling, a binary image will be transferred to a labeled image. For example, Fig. 1(b) is a labeled image of the image shown in Fig. 1(a). Thus, after connected-component labeling, we can extract each object in the (labeled) image by its label, and then, further calculate its shape features such as area, perimeter, circularity, centroid etc. Because connected components in an image may have complicated geometric shapes and complex connectivity, connected-component labeling is said to be more time-consuming than any other fundamental operations on binary images such as

* Corresponding author.
E-mail address: helifeng@ist.aichi-pu.ac.jp (L. He).

[1] For convenience, in this article we only consider $N \times N$-sized binary images. All algorithms can be easily extended to $N \times M$-sized binary images.

http://dx.doi.org/10.1016/j.patcog.2017.04.018

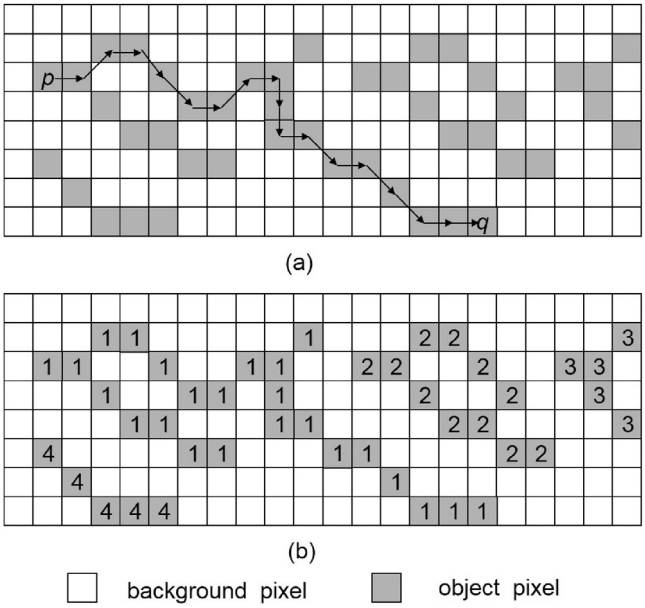


Fig. 1. 8-connected object pixels and connected components.

noise reduction, interpolation, thresholding, and edge detection. Especially, labeling cannot be completed by mere parallel local operation, but needs sequential operations [4].

A lot of connected-component labeling (CCL) algorithms have been proposed since the 1960s. According to the computer architecture and/or data structure being used, CCL algorithms can be divided into five classes: (1) algorithms for the images represented by special structures, for example, run-length structure and hierarchical tree structures, i.e., n-ary trees such as bintree, quadtree, octree, etc. [5–15]; (2) algorithms for parallel machine models such as a mesh-connected massively parallel processors or systolic array processors [16–31]; (3) algorithms for hardware implementation [29–38]; (4) algorithms for 3D and/or n-D images [39–42,98]; (5) algorithms for ordinary computer architectures such as the Von Neumann architecture and two-dimensional images.

Because images represented by special structures are rarely used in practice, we will not discuss CCL algorithms for such images especially. Moreover, because algorithms for ordinary computer architectures and two-dimensional images, i.e., algorithms in the class (5) are the base of algorithms in the other classes, we will mainly focus on the algorithms in class (5), and then discuss their parallel implementation, hardware implementation, and extension for n-D images.

For ordinary computer architectures and two-dimensional images, there are mainly two types of connected component labeling algorithms: algorithms based on label-propagation, and algorithms based on label-equivalence-resolving. In this article we review the methods and strategies of these algorithms proposed in the last decade, and present experimental results in order to bring order to these algorithms. Among others, the state-of-the-art algorithms shown in the following list will be especially reviewed in this paper:

(1) The Contour Tracing Labeling (CTL) algorithm proposed by F. Chang et al. in 2004 [43];
(2) The Hybrid Object Labeling (HOL) algorithm proposed by Herrero in 2007 [44];
(3) The Optimizing Connected-connected Labeling (OCL) algorithm proposed by Wu et al. in 2009 [46];
(4) The Improved Run-based Connected-component Labeling (IRCL) algorithm proposed by He et al. in 2010 [49];
(5) The Block based Connected-component Labeling (BCL) algorithm proposed by Grana et al. in 2010 [50];
(6) The Improved Block based Connected-component Labeling (IBCL) algorithm proposed by H. Chang et al. in 2015 [52];
(7) The Improved Configuration-Transition-based Connected-component Labeling (ICTCL) algorithm proposed by Zhao et al. in 2015 [53].

Among the above algorithms, the first two algorithms are label-propagation ones, and the others are label-equivalence ones.

There were some papers on object labeling comparisons. The paper presented in Ref. [54] reviewed some popular CCL algorithms presented in Refs. [4,43,60] in that time and mainly addressed the capability for real-time video processing, hardware implementation in FPGA for embedded systems, and memory requirements. Especially, this paper also reviewed a one-scan connected-component analysis (CCA) algorithm for objects' features by combined subsequent data analysis step into the first scan of a two-scan labeling algorithm. Without the need for buffering image data, it is very suitable for hardware implementation. The paper presented in Ref. [55] mainly compared two label-equivalence-resolving strategies used in some two-scan CCL algorithms. Moreover, the paper presented in Ref. [56] reviewed some main algorithms presented in Refs. [4,43,46–48,50,65], and provided benchmarks on different processor architectures.

However, in these papers, (1) the principles of corresponding algorithms were not discussed in detail; (2) the state-of-the-art labeling algorithms proposed in Refs. [44,49,52,53] mentioned above were not reviewed in these papers; (3) the strategies for resolving label equivalence for two-scan labeling algorithms were not explained in detail; (4) the pseudo codes of the reviewed algorithms are not given; (5) the single pass connected-component analysis algorithm discussed in Ref. [54] does not generate a labeled image, thus, as mentioned in Ref. [54], the algorithm is not suitable for applications where a labeled image is required; (6) As we will discuss in Section 5.2, any two-scan CCL algorithm reviewed in this paper can be modified as one-scan CCA algorithm in a similar way.

The rest of this paper is organized as follows: we review algorithms based on label propagation in the next section, and algorithms based on label-equivalent resolving in Section III. Comparison and experimental results on various images of state-of-the-art algorithms reviewed in this paper are presented and compared in Section IV. In Section V, we discuss parallel implementation, hardware implementation, and extension for n-D images. We give our concluding remarks and future work in Section 6.

2. Algorithms based on label-propagation

These CCL algorithms [42–44,57,58] first search an unlabeled object pixel, label the pixel with a new label; then, in the later processing, they propagate the same label to all object pixels that are connected to the pixel. Because the labels assigned to object pixels will never change, these algorithms can be easily extended to calculate the shape features of objects, such as area, centroid, perimeter etc., during the labeling.

Although these algorithms usually use the raster scan to find an unlabeled object pixel, for labeling, all of them 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; thus, they are not suitable for pipeline processing, parallel implementation, systolic-array implementation, and hardware implementation. Moreover, although the algorithms based on label-propagation are often called one-scan algorithms, many object pixels will be scanned more than twice, therefore, they are actually not one-scan algorithms.

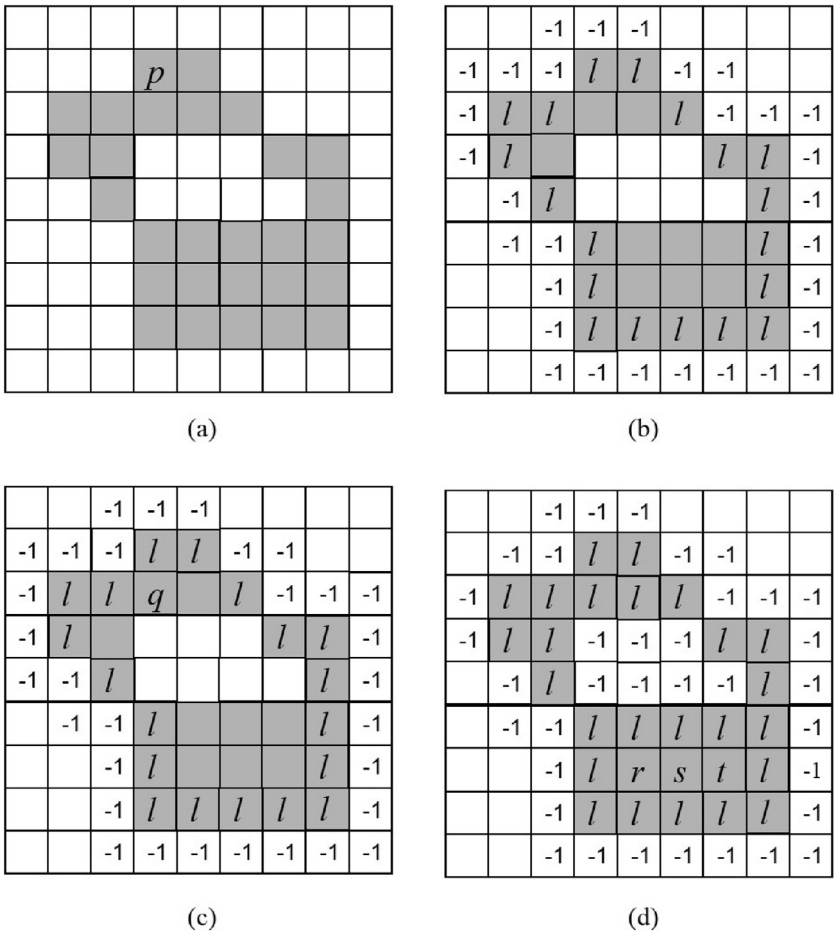


Fig. 2. Examples for explaining the CTL algorithm: p is an object pixel on an external contour of a newly encountered object; (b) after contour tracing on the external contour; (c) q is an object pixel on the internal contour newly encountered; (d) after contour tracing on the internal contour, where r, s, and t are unlabeled object pixels.

2.1. CTL (Contour-Tracing Labeling) algorithms

The CTL algorithm proposed in Ref. [43] is a Contour-Tracing-based algorithm. It processes pixels in the given binary image one by one in the raster scan. For the current pixel $b(x, y)$, it does nothing if $b(x, y)$ is a background pixel, otherwise, i.e., $b(x, y)$ is an object pixel, it processes $b(x, y)$ as follows, where l is a new label:

(1) If $b(x, y)$ is unlabeled and its upper neighbor, i.e., $b(x, y-1)$, is a background pixel, then $b(x, y)$ (e.g., the pixel p in Fig. 2(a)) must be on an external contour of a newly encountered object. The CTL algorithm assigns label l, to $b(x, y)$ and all pixels on the external contour by contour tracing. At the same time, it also marks all background pixels surrounding the external contour (the object) with -1 (Fig. 2(b)) for distinguishing whether an object pixel is on an unprocessed internal contour or not (in order to prevent to trace a contour multi times). Then, the CTL algorithm increases the value of l by 1.

(2) If the pixel immediately below $b(x, y)$, i.e., $b(x, y+1)$, is an unmarked background pixel (i.e., $b(x, y+1)=0$), $b(x, y)$ must be on a newly encountered internal contour. If $b(x, y)$ is unlabeled (in this case, its left neighbor of $b(x, y)$, i.e., $b(x-1, y)$, must be a labeled pixel) (e.g., the pixel q in Fig. 2(c)), the CTL algorithm assigns the same label of $b(x-1, y)$, to $b(x, y)$.Then, it uses contour tracing to find the internal contour from $b(x, y)$, and assigns the label of $b(x, y)$ to all pixels on the counter. At the same time, it also marks all background pixels surrounding the inter contour (the hole) with -1 (Fig. 2(d)) for avoiding starting contour tracing again at a labeled object on the internal contour.

(3) If $b(x, y)$ is an unlabeled object pixels not being described in (1) and (2), then the left neighbor of $b(x, y)$, i.e., $b(x-1, y)$, must be a labeled pixel (e.g., the pixel r, s, and t in Fig. 2(d)). The CTL algorithm assigns the label of $b(x-1, y)$ to $b(x, y)$.

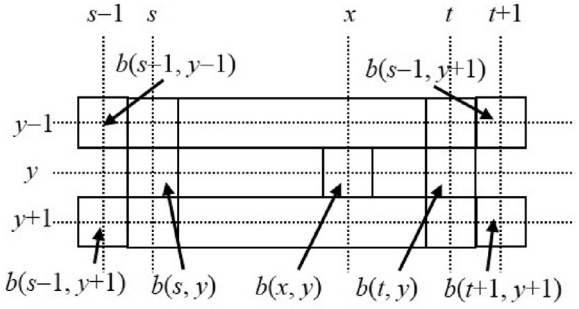


Fig. 3. Processing an unlabled object pixel in the HOL algorithm.

According to the above analysis, the pseudo code of the CTL algorithm can be given as follow, where the label variable l is initialized to 2 for distinguishing whether an object pixel is labeled or not.

```
l ← 2
for each x, y in the raster scan
| if b(x, y) ≥1 // b(x, y) is an object pixel
| | if b(x, y) = 1 & b(x, y − 1) = 0 // b(x, y) is an object pixel on an unprocessed
    external contour
| | | for each pixel b(u, v) on the external contour from
| | | b(x, y), b(u, v) ← l, and for each background
| | | pixel b(m, n) surrounding the external contour,
| | | b(m, n) ← −1
| | | l ← l + 1
| | else if b(x, y + 1) = 0 // b(x, y) is an object pixel on an unprocessed interior
    contour
| | | if b(x, y) = 1
| | | | b(x, y) ← b(x − 1, y)
| | | end of if
| | | for each pixel b(u, v) on the interior contour from
| | | b(x, y), b(u, v) ← b(x, y), and for each
| | | background pixel b(m, n) surrounding the
| | | interior contour b(m, n) ← −1
| | else if b(x, y) = 1
| | | b(x, y) ← b(x − 1, y)
| | end of if
| end of if
end of for
```

The authors of the CTL algorithm proved that the algorithm is a linear algorithm on the sizes of images, i.e., the worst case complexity is $O(N^2)$. Moreover, no memory space is necessary. A great advantage of the CTL algorithm is that through labeling, it can obtain the contours of objects in the image simultaneously.

2.2. HOL (Hybrid Object Labeling) algorithm

The HOL algorithm proposed in Ref. [44] labels an image by use of the recursive and hybrid techniques. It scans pixels of the image in some order (usually raster scan) to find an unlabeled object pixel. Whenever such an unlabeled object pixel $b(x, y)$ is found, the HOL algorithm first assigns each object pixel such that is 8-connected to $b(x, y)$, from $b(s, y)$ to $b(t, y)$ (see Fig. 3) a new label l, and then for each object pixel from $b(s–1, y–1)$ to $b(t+1, y–1)$ and $b(s–1, y+1)$ to $b(t+1, y+1)$ (see Fig. 3), process it in the same way as processing $b(x, y)$ if it is unlabeled. The pseudo code of the HOL algorithm can be given as follow, where the label variable l is also initialized to 2 with the same reason in the CTL algorithm introduced above.

```
l ← 2
for each x, y in the raster scan
| if b(x, y) = 1
| | Label(x, y)
| | l ← l + 1
| end of if
end of for
Label(x, y)
| while b(x − 1, y) = 1
| | x ← x − 1
| end of while
| m ← x
| while b(m, y) = 1
| | b(m, y) ← l
| | m ← m + 1
| end of while
| x ← x − 1
| while x ≤ m
| | if b(x, y − 1) = 1 Label(x, y − 1)
| | if b(x, y + 1) = 1 Label(x, y + 1)
| | x ← x + 1
| end of while
end of Label
```

A great advantage of the HOL algorithm is that an object in an interesting area can be identified and processed without being forced to label all objects in the image, which is a favor property for real time image processing applications. On the other hand, the main disadvantages of the HOL algorithm is that for big objects, it require a large number of recursive calls, which require a large stack to store local variables and register values.

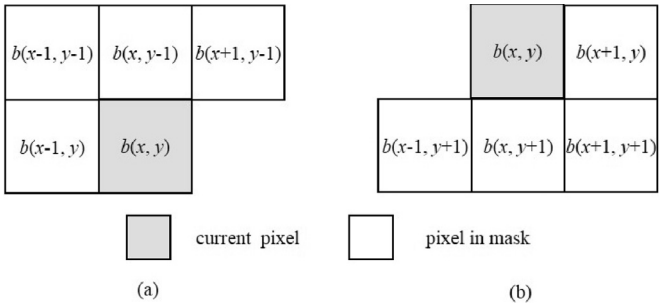


Fig. 4. Masks used in the multi-scan algorithms: (a) forward mask and (b) backward mask.

Although the author of the HOL algorithm did not give the analysis on the time complexity, the experimental results on noise images demonstrated that the HOL algorithm is linear on the size of images, i.e., its time complexity is $O(N^2)$. Moreover, no memory space is necessary. The main problem of the HOL algorithm is that it is a recursive one; thus, it will not work for large size images.

3. Algorithms based on label-equivalent-resolving

All algorithms based on label-equivalent-resolving process a binary image in raster scan, and consist of two steps: in the first step, they assign a provisional label (usually from 1) to each object pixel, which always finishes in the first scan; in the second step, they integrate all provisional labels assigned to each object, which are called *equivalent labels*, to a unique label, which called the *representative label*, and replace the provisional label of each object pixel by its representative label. The process for integrating equivalent labels is called *label equivalence resolving*. Because the maximum number of connected components in a $N \times N$-sized binary image is $N \times N/4$, the maximum number of provisional labels will also be $N \times N/4$.

According the necessary times of raster scan for completing connected-component labeling, these algorithms can be divided into four classes: (1) multi-scan algorithms; (2) four-scan algorithm; (3) two-scan algorithms; (4) one-and-a-half algorithm.

3.1. Multi-scan algorithms

Label-equivalent-resolving-based multi-scan algorithms [2,59,60] label a binary image by scanning the image forward and backward alternatively.

For example, the algorithm introduced in Ref. [2] first scan the image forward once and processes the current pixel in the scan by use of the forward mask shown in Fig. 4(a), which consists of 4-neighbored pixels in the processing area in the scan. For each current pixel in the scan, if it is a background pixel, nothing needs to do. On the other hand, i.e., the current pixel is an object pixel, this algorithm assigns a new provisional label to the pixel if there is no label (object pixel) in the mask, otherwise, all object pixels in the mask are connected (i.e., belong to the same connected component), all labels in the mask are equivalent labels. This algorithm assigns the minimal label in the mask to the current pixel, and rewrites the other labels in the mask to the minimal label. Thus, after the first scan, each object will be assigned a provisional label.

This algorithm integrates provisional labels through propagating minimal labels. After the first scan, this algorithm scans the image backward by use of the backward mask shown in Fig. 4(b), which

also consists of 4-neighbored pixels in the processing area in the scan. For each current pixel, if it is a background pixel, nothing needs to be done. Otherwise, if the provisional labels in the mask are different, it replaces each label in the mask by the minimal label in the mask. Then, this algorithm will alternately scan (process) the image forward and backward for propagating minimal labels assigned to objects until no label changes.

When the algorithm halts, all pixels of an object will be assigned the minimal label among the provisional labels assigned to the object in the first scan. The number of necessary forward and backward scan times of this algorithm depends on structures of objects in the image being labeled. The maximum number of necessary scan times for labeling an $N \times N$-sized binary image will be $N-1$ [61]; thus, the worst-case lower bound of the algorithm is $O(N^3)$. On the other hand, no memory is necessary.

According to the experimental results shown in Ref. [47], multi-scan algorithms are much less efficient than two-scan labeling algorithms to be introduced below, therefore, there is no multi-scan algorithm in state-of-the-art algorithms.

3.2. Four-scan algorithm

Suzuki et al. [61] proposed a method by use of a one-dimensional table, called *label connection table*, which records the relation of a label and the minimal label in the masks, to speed up the propagation of the minimal label assigned to a connected component in forward and backward scans in multi-scan algorithms. Thus, provisional labels propagate not only on the neighbors of object pixels in the mask but also in the table; therefore, the number of scans can be reduced.

Although the authors failed to give a proof, the experimental results on various images showed that the maximum scan times for completing labeling is four. Moreover, the experimental results on noise images also demonstrated that the algorithm is linear on the sizes of images, i.e., the worst-case lower bound of the algorithm for labeling an $N \times N$-sized binary image should be $O(N^2)$. Because this algorithm needs to record the relation of a label and the minimal label in the mask, the necessary memory space for the table is $N^2/4$. According to the experimental results shown in Ref. [47], this algorithm is also much less efficient than state-of-the-art two-scan labeling algorithms to be introduced below, therefore, we do not introduce it in detail.

3.3. Two-scan algorithms

3.3.1. An overview

Two-scan algorithms [2,45–53,62–65] complete labeling by scanning a given image forward twice. They consist of **the following four phases**: (1) provisional label assigning; (2) equivalent label recording; (3) label-equivalence resolving; and (4) label replacing. The first work is completed in the first scan. The second work, i.e., equivalent label recording, in is completed in the first scan, and the third work, i.e., label-equivalence resolving, is completed in the first scan and/or between the first scan and the second scan, the forth work, i.e., label replacing is completed in the second scan.

According to the construction of basic elements for labeling, two-scan algorithms can be divided into pixel-based algorithms, run-based algorithms and block-based algorithms.

For the first work, i.e., provisional label assigning, two strategies have been used. The first one is exactly the same as in the multi-scan algorithms introduced above, i.e., assigning the minimal provisional label in the mask to the current object pixel. Based on the observation that all labels in the mask of an object pixel are equivalent labels and will be finally integrated, the second one (proposed in Ref. [47]) assigns any provisional label in the mask to the current object pixel. Because the first one needs to calculate the minimal label in the mask and the second one does not, the second one is much more efficient than the first one. Therefore, all state-of-the-art two-scan labeling algorithms use the second strategy for provisional label assigning.

3.3.2. Equivalent label recording and label-equivalence resolving

As introduced above, when the current pixel in the first scan is an object pixel, all provisional labels in the mask are equivalent labels. It is clear that for an $N \times N$-sized image, the maximum number of connected components in a binary image is $N \times N/4$, i.e., the maximum number of provisional labels to be assigned is $N \times N/4$. Three methods have been proposed for equivalent label recording and label-equivalence resolving.

(1) *2D Table strategy*

An $L \times L$-sized two-dimension table T is used for equivalent label recording and label-equivalence resolving [63,64], where L is the maximum possible provisional label to be assigned, i.e., $N \times N/4$. All elements of the table will be initialized to 0. If u and v are found to be equivalent, $T[u][v]$ to is set to 1. After the first scan, all pairs of equivalent labels will be recorded in the table. Obviously, all provisional labels in the same column and the same row are equivalent labels. Label-equivalence resolving can be completed by analyzing the table.

Moreover, it is obvious that if u and v are equivalent, then we can set either $T[u][v] = 1$ or $T[v][u] = 1$. Thus, when u and v are found to be equivalent, if we always set $T[m][n] = 1$, where $m = \max\{u, v\}$ and $n = \min\{u, v\}$, then the half of the table will be enough for equivalent label recording and label-equivalence resolving. Although this cannot reduce the necessary memory size, but can reduce the cost for label-equivalence resolving to half.

Because $L = N \times N$, the size of the table will be $N^2 \times N^2/16$. Thus, the time complexity for analyzing the table is $O(N^4)$, and the necessary memory space is $N^4/16$. Both are much larger than those of the other two methods to be introduced below. Therefore, no state-of-the-art two-scan algorithm takes this strategy for equivalent label recording and label-equivalence resolving.

(2) *Union-find-tree strategy*

The second method for equivalent label recording and label-equivalence resolving is by use of the union-find-tree strategy [66–69]. This strategy is realized by use of union-find trees: (1) each node is a provisional label with a link to a node in the tree that it belongs to; (2) all provisional labels in a tree are equivalent labels; (3) the label of the root node of a tree is used the representative label of all provisional labels in the tree.

When a new provisional label l is assigned to an object pixel, we create a tree with root l only where the link of the node is set to itself. Whenever two labels u and v are found to be equivalent, we calculate the root of node u and that of node v. Without loss of generality, **suppose that m and n are the roots of the two nodes respectively, then if $m=n$, it means that node u and node** v belong to the same tree, nothing needs to be done. On the other hand, if $m > n$, we change the link of node m to node n, otherwise, i.e., $m < n$, we change the link of node n to node m. Thus, at any processing point, all and all provisional labels assigned to an object in the processed area will be recorded in a tree, where the root node will be the minimal label in the tree. Therefore, after the first scan, all and only all provisional labels assigned to an object will be recorded in a tree.

The union-find-tree strategy can be implemented by use of a one-dimension table T with the size of $N \times N/4$, which is, as introduced above, the maximum number of provisional labels possibly assigned to an $N \times N$-sized image. There are many methods for implementation, and the most efficient method for implementation was proposed by Wu et al. in [46]. In this method, a parent node

of a node m is represented by $T[m]$. The three main operations are *NewTree, FindRoot* and *SetRoot*. The operation *NewTree*(l) establishes a new tree with the root l only by executing the following procedure.

```
NewTree(l)
| T[l] ← l
end of NewTree
```

The operation *FindRoot*(u) finds the root of a node u by executing the following procedure.

```
FindRoot(u)
| r ← u
| while T[r] ≠ r
| | r ← T[r]
| end of while
| return r
end of FindRoot
```

The operation *SetRoot*(u, r) changes all nodes on the path from the node u to its root to point directly to *root* r by executing the following procedure.

```
SetRoot(u, r)
| while T[u] ≠ u
| | v ← T[u]
| | T[u] ← r
| | u ← v
| end of while
| T[u] ← r
end of SetRoot
```

This operation is known as the path compression. A great advantage by use of the path compression is reducing the height of a tree, thus, reducing the cost for finding the root of a node.

When a new provisional label l is assigned to an object pixel, *NewTree*(l) is called to establish a new tree with the root l only. When two provisional labels m and n are found to be equivalent, all labels in the tree with node m and all labels in the tree with node n are equivalent labels. The label equivalence of m and n is resolved, i.e., the two trees are combined, by calling the following procedure *Union*(m, n).

```
Union(m, n)
| r ← FindRoot(m)
| if r ≠ n
| | s ← FindRoot(n)
| | if(r > s) r ← s
| | SetRoot(n, r)
| end of if
| SetRoot(m, r)
| return r
end of Union(m, n)
```

Obviously, when combining two trees, the procedure *Union* always selects the root with the smaller label as the root of the combined tree, thus, the root of a tree always has the smallest label in the tree and the parent node of a node always has a label not larger than its own label, i.e., $T[l] \leq l$.

After the first scan, all and only all equivalent labels assigned to a connected component will be with nodes in the corresponding tree. In order to complete label equivalence resolving, i.e., to guarantee each node in a tree directly pointing to the root of the tree, the following procedure *Flatten1* should be executed, where L is the maximum provisional label.

```
Flatten1(L)
| for i ← 1 to L
| | T[l] ← T[T[l]]
| end of for
end of Flatten1
```

If consecutive final labels are necessary, the following procedure *Flatten2*, should be executed.

```
Flatten2(L)
| k ← 1
| for i ← 1 to L do
| | if T[i] ≠ i T[i] ← T[T[l]]
| | else
| | | T[i] ← k
| | | k ← k+1
| | end of if
| end of for
end of Flatten2
```

The worst-case lower bound of this strategy for equivalent label recording and label-equivalence resolving has been proved to be $O(N^2)$ [69]. Because an $N \times N/4$-sized table is needed for implementing this strategy, the necessary memory space is $N^2/4$.

(3) *Equivalent-label-set strategy*

Lastly, equivalent label recording and label-equivalence resolving can be also completed by use of *equivalent label sets* [70]. By this strategy, all equivalent provisional labels are combined in an equivalent label set, where the minimal label in a set is called the *representative label* of the set as well as all labels in the set. For convenience, an equivalent label set with the representative label r is denoted as to $S(r)$, and the representative label of a label m is denoted as to $R[m]$.

When a new provisional label l is assigned to an object pixel, the equivalent label set $S(l)=\{l\}$ is established. Whenever two labels u and v are found to be equivalent, then all labels in $S(\alpha)$ and $S(\beta)$, where $\alpha=R(u)$ and $\beta=R(v)$, are equivalent labels, thus, $S(\alpha)$ and $S(\beta)$ should be combined, i.e., $S(\gamma)=S(\alpha) \cup S(\beta)$, where γ is the smaller one of α and β.

This strategy can be efficiently implemented by three one-dimension arrays [70], where an equivalent label set is realized by a list, and the list head is the representative label of the set. For convenience, hereafter we will also use $S(u)$ to denote the corresponding equivalent label list. One of the three arrays, R, is used for representing the representative label of a label, $R[l]=r$ indicates that the representative label of the provisional label l is r. Another, *Last*, is used to indicating the last element in the list: $Last[a]=b$ means that the last element in the list $S(a)$. The last one, *Next*, is used to indicating the next element of an element: $Next[m]=n$ means that the next element of m in the list is n. Especially, $Next[u]=0$ means that the element u is the last element in the list.

When a new provisional label l is assigned to an object pixel, a list $S(l)=\{l\}$ is established by executing the following procedure *NewSet*:

```
NewSet(l)
| R[l] ← l
| Last[l] ← l
| Next[l] ← 0
end of NewSet(l)
```

On the other hand, two equivalent label lists $S(u)$ and $S(v)$ can be combined by executing the following procedure *CombineSet*:

```
CombineSet(u, v)
| if u > v
| | for i ← u to i ≠ 0 do
| | | R[i] ← v
| | | i ← Next[i]
| | end of for
| | Next[Last[v]] ← u
| | Last[v] ← Last[u]
| else if u < v
| | for i ← v to i ≠ 0 do
| | | R[i] ← u
| | | i ← Next[i]
| | end of for
| | Next[Last[u]] ← v;
| | Last[u] ← Last[v];
| end of if
end of CombineSet
```

Notice that the case where $u = v$ means that the two equivalent sets $S(u)$ and $S(v)$ are the same, thus, nothing needs to be done.

It should be noted that the path-compression optimizations of the union-find algorithm introduced above can be achieved naturally in the strategy. Because all labels in an equivalent label set are always at the same level, there is no path that needs to be compressed further. Moreover, by this strategy, at any point of the first scan, all provisional labels assigned to each connected component in the processed area of the given image will be combined in an equivalent label set where the minimal label is the representative label. Thus, as soon as the first scan completes, all provisional labels assigned to each connected component in the given image will be combined in an equivalent label set with the minimal label as the representative label. In other words, label equivalence resolving is finished in the first scanning. In comparison, to complete label equivalence resolving, after the first scan, the union-find strategy introduced above needs to execute the procedure *Flatten1* or the procedure *Flatten2*. Therefore, this strategy is more efficient than the union-find-tree strategy.

However, final labels generated by the above equivalent-label-set strategy are not consecutive. In order to generate consecutive final labels, the following procedure *ConsecutiveLabel* should be executed after the first scan,[2] where L is the maximum provisional label:

```
ConsecutiveLabel(L)
| k ← 1
| for i ← 1 to L do
| | if R[i]=i
| | | R[i] ← k,
| | | k ← k+1
| | end of if
| end of for
End of ConsecutiveLabel
```

The worst-case lower bound of time complexity of this strategy for equivalent label recording and label-equivalence resolving has been proved to be $O(N^2)$. Because three $N \times N/4$-sized tables needed to implement the strategy, the necessary memory space is $3 \times N^2/4$.

3.3.3. State-of-the-art two-scan labeling algorithms

According to the basic elements to be considered for connectivity, there are pixel-based two-scan labeling algorithms, run-based two-scan labeling algorithms, and block-based two-scan labeling algorithms. For convenience, we do not consider the case for consecutive final labels.

(1) *Pixel-Based Algorithms*

[2] In order to generate consecutive final labels, any two-scan labeling algorithm need to execute a similar procedure.

Pixel-based algorithms naturally consider connectivity between pixels. In recent ten years, several two-scan CCL algorithms have been proposed. Because the limitation of space, we mainly review the Optimizing connected-component labeling (OCL) algorithm proposed in Ref. [46] and the Improved Configuration-Transmit-based Connected-component Labeling (ICTCL) algorithm proposed in Ref. [53].

3.3.3.1. OCL algorithm. The OCL algorithm proposed in Ref. [46] uses the mask shown in Fig. 4(a) for processing the current pixel, and uses the union-find-tree strategy for recording and resolving label equivalence.

In the first scan, if the current pixel is a background pixel, nothing needs to be done. On the other hand, if the current pixel $b(x, y)$ is a foreground pixel, the OCL algorithm checks the pixels in the mask by use of the order $b(x, y-1) \rightarrow \boldsymbol{b(x+1, y-1)} \rightarrow b(x-1, y-1) \rightarrow b(x-1, y))$, which is considered to be efficient by observation.

According to the above analysis, the pseudo code for the OCL algorithm to process the current pixel $b(x, y)$ can be shown as follows, where l is initialized to 1.

```
OCL-Processing(b(x, y))
| if b(x, y) > 0
| | if b(x, y − 1) > 0
| | | b(x, y) ← FindRoot(b(x, y − 1))
| | else if b(x + 1, y − 1) > 0
| | | if b(x − 1, y) > 0
| | | | b(x, y) ← Union(b(x + 1, y − 1), b(x − 1, y))
| | | else if b(x − 1, y − 1) > 0
| | | | b(x, y) ← Union(b(x − 1, y − 1), b(x − 1, y))
| | | else
| | | | b(x, y) ← FindRoot(b(x − 1, y))
| | | end of if
| | else if b(x − 1, y) > 0
| | | b(x, y) ← FindRoot(b(x − 1, y))
| | else if b(x − 1, y − 1) > 0
| | | b(x, y) ← FindRoot(b(x − 1, y − 1))
| | else
| | | b(x, y) ← NewTree(l),
| | | l ← l + 1
| | end of if
end of if
end of OCL-Processing(b(x, y))
```

When the first scan finished, the procedure *flatten1* is called to make the height of each tree to be 1, i.e., the parent of each node in a tree is the root of the tree. Thus, for each label l, $T[l]$ will be its representative label. Then, by replacing each label with its representative label in the second scan, each pixel of each connected component in the given image will be assigned a unique label.

The pseudo code for the OCL algorithm can be given as follows.

```
%% The OCL algorithm
l ← 1;
for y ← 1 to y < N − 1 do
| for x ← 1 to x < N −1do
| | OCL-Processing(b(x, y))
| | x ← x+1
| end of for
| y ← y+1
end of for
flatten1(l − 1);
for y ← 1 to y < N − 1 do
| for x ← 1 to x < N − 1 do
| | b(x, y) ← T[b(x, y)]
| | x ← x+1
| end of for
| y ← y+1
end of for
```

The authors of the OCL algorithm proved that the worst-case time complexity the OCL algorithm is $O(N^2)$. Moreover, because

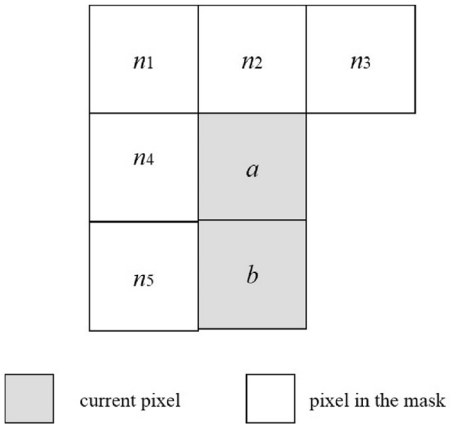


Fig. 5. The mask used in the CTCL algorithm.

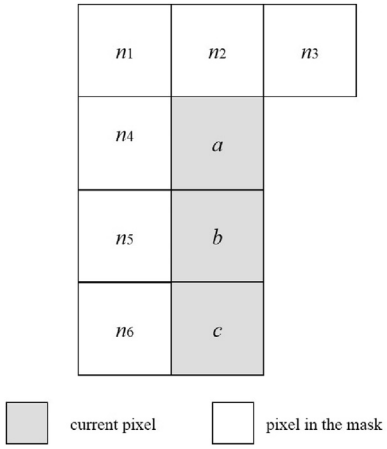


Fig. 6. The mask for the ICTCL algorithm.

it adopted union-find-tree strategy for resolving label equivalence, the necessary memory space is $N^2/4$.

3.3.3.2. ICTCL algorithm. In 2009, He et al. proposed an efficient two-scan CCL algorithm, which also uses the mask shown in Fig. 4(a) for processing the current pixel, and uses the equivalent-label-set strategy for recording and resolving label equivalence. Moreover, in order to reduce the times for checking pixels in the mask as small as possible, it uses Kaught-map analysis to obtain the optimal order for checking the pixels in the mask. The average number of pixels to be checked for processing an object pixel in the first scan is 33/16, less than 4 in previous conventional two-scan CCL algorithms.

This algorithm was improved in Ref. [48]. Because the information whether the pixel $b(x-1, y)$ is a foreground pixel or not can be obtained without any cost in the processing, we don't need to check the pixel. In the other words, the pixel $b(x-1, y)$ can be removed from the mask shown in Fig. 4(a). The average number of pixels to be checked for processing an object pixel in the first scan decreases to 7/4.

In 2014, He et al. proposed a configuration-transmit-based two-scan labeling algorithm (*CTCL algorithm*) [51], which uses the mask shown in Fig. 5 to scan **image lines** over each other line and process pixels two by two, and uses the equivalent-label-set strategy for recording and resolving label equivalence. Moreover, by considering the transition of configuration of pixels in the mask, it utilizes the information detected in processing the last two pixels for processing the current two pixels. By this algorithm, any pixel in the mask checked in processing the current pixel will not be checked again when processing the next two pixels; thus, the average number of pixels to be checked for processing an object pixel in the first scan is reduced to 3/4.

In order to further reduce the average number of pixels to be checked for processing an object pixel, the ICTCL algorithm proposed in Ref. [53] uses the mask shown in Fig. 6 to process pixels three by three. There are 16 configurations of the mask as shown in Fig. 7, where a meaningless pixel means that whether it is an object pixel or a background pixel does not influence the labeling result for the current pixels, and an uncertain pixel means that whether it is an object pixel or a background pixel is not known yet.

For example, in the case shown in Fig. 8, suppose that after processing n_4, n_5, and n_6, we know that n_1, n_2, n_4, n_5, and n_6 are background pixel, background pixel, object pixel, background pixel, and object pixel; thus, we know that the configuration of the mask for processing p_1, p_2, and p_3 is C_6 shown in Fig. 7. Therefore, we can process p_1, p_2, and p_3 without checking n_1, n_2, n_4, n_5, and n_6. In a similar way, after processing p_1, p_2, and p_3 based on the configuration C_6, we know that n_2, n_3, p_1, p_2, and p_3 are background pixel, unknown pixel, background pixel, object pixel, and background pixel. Thus, the configuration for processing q_1, q_2, and q_3 will be C_{12}. By the ICTCL algorithm, the average number of pixels to be checked for processing an object pixel in the first scan decreases to 1/2.

The pseudo codes for processing the current pixels p_1, p_2, and p_3 in the case shown in Fig. 8 can be given as follows.

```
%% The pseudo codes for processing the current pixels p1, p2, and p3 for the
    configuration C6
if p1 > 0
| ......
else if p2 > 0
| CombineSet(R[n4], R[n6]);
| p2 ← R[n4];
| if p3 > 0
| | p3 ← p2
| end of if
| x ← x+1
| go to the procedure for the configuration C12
else if p3 > 0
| ......
end of if
```

Because both the CTCL algorithm and the ICTCL algorithm adopt equivalent-label-set strategy for resolving label equivalence, when the first scan finished, for each label l, its representative label can be found by $R[l]$. Then, by replacing each label with its representative label in the second scan, the pixels of each connected component in the given image will be assigned a unique label.

The experimental results on noise images with densities from 0 to 1 show that both the CTCT algorithm and the ICTCT algorithm are linear on image sizes; thus, the worst-case time complexity of the two algorithms should be $O(N^2)$. Moreover, because they use equivalent-label-set strategy for resolving label equivalence, the necessary memory space will be $3 \times N^2/4$.

By experimental results shown in Ref. [53], the ICTCL algorithm is only a little more efficient than the CTCL algorithm on various kind images. For example, the ICTCL algorithm is only 1.05 times faster than the CTCL algorithm on noise images with densities from 0 to 1. On the other hand, the ICTCL algorithm is much more complicated than the CTCL algorithm: the source code of the ICTCL algorithm is five times longer than that of the CTCL algorithm. Therefore, the authors conjectured that it would not become more efficient by simultaneously processing pixels four by four or more. The further details about the ICTCL algorithm can be found in Ref. [53].

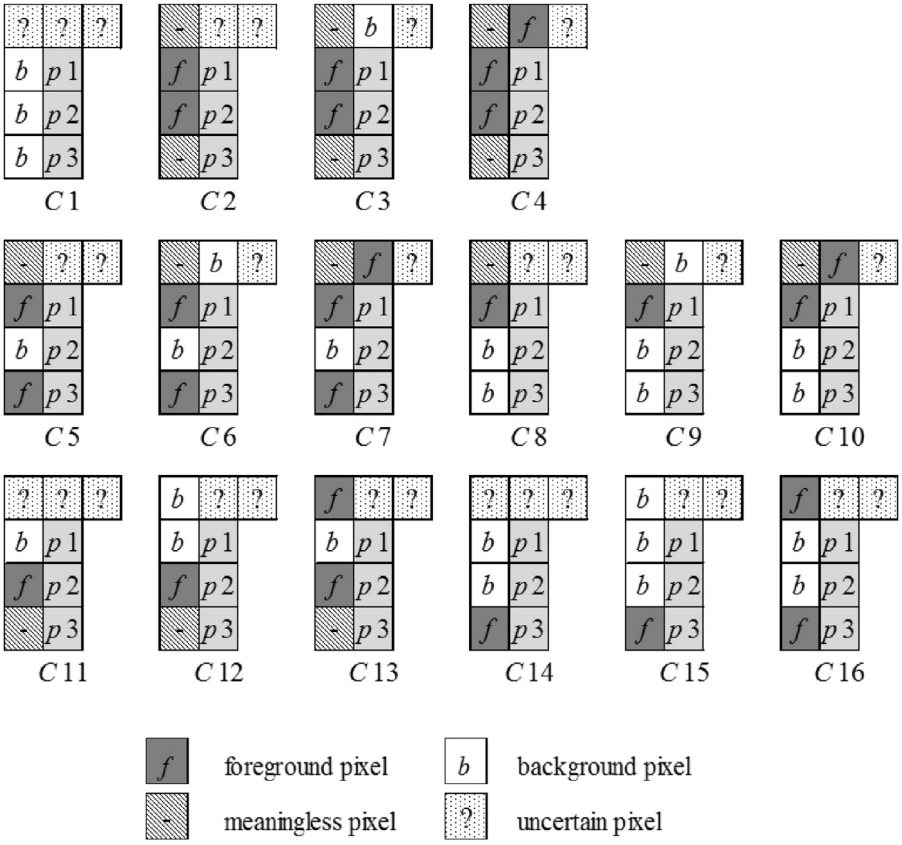


Fig. 7. 16 patterns of configurations to be considered in the ICTCL algorithm.

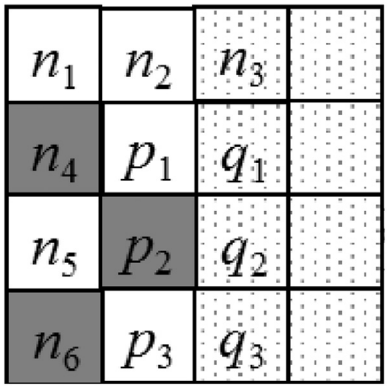


Fig. 8. An example.

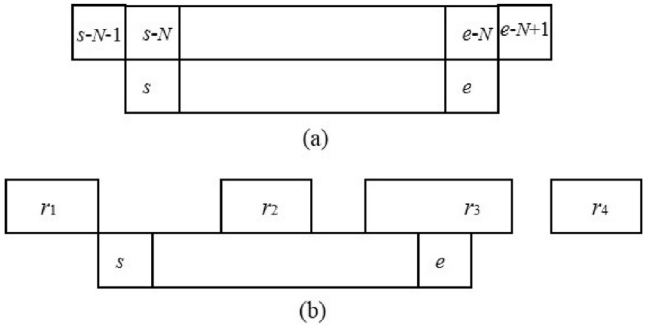


Fig. 9. (a) the 8-connectivity scope of a run, and (b) an example for 8-connected runs.

(2) *Run-based two-scan Labeling Algorithm*

A *run* is a maximal sequence of consecutive object pixels in a row. Because all object pixels in a run certainly belong to the same object and will be finally assigned the same label, run-based algorithms take a run as a super-object pixel, and consider the connectivity between runs. For convenience, in this subsection, we use $p[y \times N + x]$ to denote the pixel as well as its value at (x, y) in the image. A run from $p[s]$ to $p[e]$ is represented as $r[s, e]$. Let $r[s, e]$ be the current run in the raster scan. Among the runs that have been processed before processing the current run, a run $r[u, v]$ in the row immediately above the current row such that one of its pixels occurs between $p[s-N-1]$ and $p[e-N+1]$, i.e., $u \leq e-N+1$ and $v \geq s-N-1$, is 8-connected to $r[s, e]$, as shown in Fig. 9(a). For convenience, a run 8-connected to r $r[s, e]$ is called 8-connected run of $r[s, e]$. For example, the runs r_1, r_2, and r_3 shown in Fig. 9(b) are the 8-connected runs of $r(s, e)$.

The Improved Run-based connected-component Labeling (IRCL) algorithm proposed in Ref. [49] is an improvement of the run-based CCL algorithm proposed in Ref. [44]. Based on the observation that all pixels in a run belong to the same connected component and will finally be assigned the same label, the IRCL algorithm considers a run as a *super-object pixel*. In the first scan, instead of assigning a label to each pixel in a run, it assigns a label to each run. Because the number of runs is usually much smaller than that of object pixels, the cost for assigning a label to each run will be much smaller than assigning a label to each object, which directly leads to reduce the operations for resolving label equivalence. Moreover, the IRCL algorithm records all run data obtained in the first scan and uses them for final label assignment to avoid scan the image again; thus, background pixels will be processed

only in the first scan. In this meaning, the IRCL algorithm is a one-and-a-half scan labeling algorithm. The pseudo codes of the IRCL algorithm can be given as follows, where l is the variant for label, i for the number of runs, $L[i]$ the provisional label of the ith run, $S(i)$ and $E(i)$ the starting point and the ending point of the ith run, respectively.

```
%% The IRCL algorithm
l ← 1, i ← 1;
for each run starting from s and ending at e found in the raster scan
| S(i) ← s, E(i) ← e;
| if there is not a run such that S(j) ≤ e − N + 1 and E(j) ≥ s − N − 1
| | L[i] ← l; // assign the run a new provisional label
| | NewSet(l);
| | l ← l + 1;
| else
| | let r_x, ..., r_{x+n} be the runs such that for x ≤ j ≤ x + n, S(j) ≤ e − N + 1 and E(j)
    ≥ s − N − 1
| | for j ← 1 to n do
| | | CombineSet(R[L[x]], R[L[x + j]])
| | end of for
| | L[i] ← R[L[x]];
| end of if
| i ← i + 1;
end of for
for x ← 1 to i − 1 do // final label assignment
| for y ← S(x) to E(x) do
| | p(y) ← R[L[x]];
| end of for
end of for
```

The experimental results on noise images with densities from 0 to 1 shows that the IRCL algorithm is linear on image sizes; thus, the worst-case time complexity of the IRCL algorithm should be $O(N^2)$. Moreover, because the IRCL algorithm needs $N^2/4$ memory space for recording run label and $2 \times N^2/4$ memory space for recording the run data (starting point and ending point), and uses the equivalent-label-set strategy for label-equivalence-resolving, the necessary memory space is $N^2/4 + 2 \times N^2/4 + 3 \times N^2/4$, i.e., $3 \times N^2/2$. Further details can be found in Ref. [49].

(3) *Block-based two-scan labeling algorithms*

An area 2×2 pixels in an image is called a *block*. A block is called an *object block* if there is one or more object in the block, otherwise a *background block*. Because object pixels in an object block certainly belong to the same object, and thus, will be finally assigned the same label, similar as the IRCL algorithm, block-based algorithms consider blocks as *super-object* pixels, and process an image block by block. In the first scan, they assign each object block a provisional label, and resolve label equivalence among object blocks. Object-block labels will be used for final assignment in the second scan. Moreover, block-based algorithms use the equivalent-label-set strategy for resolving label equivalence.

3.3.3.3. BCL algorithm. In the first scan, the BCL algorithm proposed in Ref. [50] uses the block-based scan mask shown in Fig. 10, which consists of four blocks with 16 pixels to process the current block. By analyzing the 20 pixels in the current block and block-based scan mask, the BCL algorithm obtains the four necessary central pixels and 12 neighboring pixels for advanced analysis, and ignores pixels *a, f, l*, and *q* in Fig. 10, which are useless for processing the current block. The decision tree generated by this algorithm contains 210 nodes, with 211 leaves sparse over 14 levels. In total, 65,536 cases are considered.

The length of the source code of the BCL algorithm provided by their author is more than 1600 lines, much longer than those of algorithms introduced above. Although the authors of the BCL algorithm could not give the time complexity analysis, the experimental results on noise images with densities from 0 to 1 show

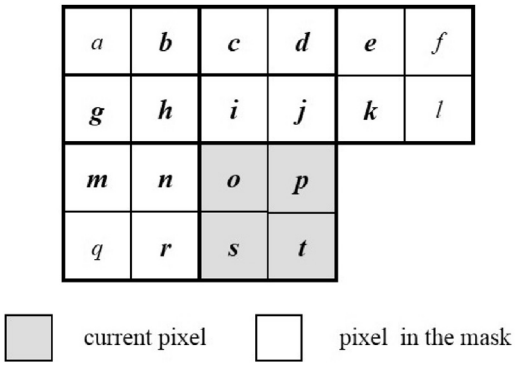


Fig. 10. The mask used in the BCL algorithm.

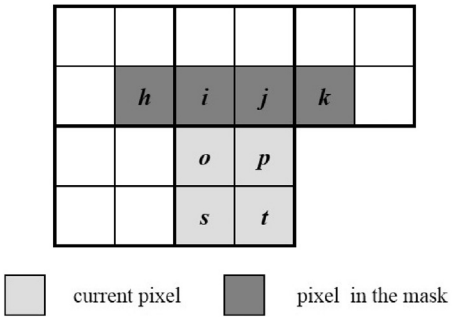


Fig. 11. The mask used in the IBCL algorithm.

that the BCL algorithm is also linear on image sizes, i.e., the worst-case time complexity of the BCL algorithm is $O(N^2)$. On the other hand, because the maximum number of blocks in a binary image is $N \times N/4$, the BCL algorithm needs an $N^2/4$-sized memory space for recording block labels, and $3 \times N^2 4$-sized memory space by use of the equivalent-label-set strategy for resolving label equivalence. Thus, the necessary memory space of the BCL algorithm is $N^2/4 + 3 \times N^2 4$, i.e., N^2.

3.3.3.4. IBCL algorithm. The IBCL algorithm proposed in Ref. [52] is an improvement of the BCL algorithm. Because only pixels *n, r, h, i, j* and *k* in the mask shown in Fig. 11 will connect to object pixels in the current block, the IBCL algorithm does not consider other pixels in the mask. Moreover, by use of the same strategy proposed in the IFCL algorithm, it utilizes the information about pixels *n* and *r* obtained during processing the last block for processing the current block. In this way, the IBCL algorithm only checks pixels *h, i, j* and *k*, much smaller than those checked in the BCL algorithm. As a result, the number of leaf nodes and depth levels in the constructed binary decision tree are 27 and 7 respectively, either is much smaller than that in the binary decision tree constructed by the BCL algorithm, which are 211 and 14, respectively.

Compared to the BCL algorithm, the IBCL algorithm just reduced the cases to be considered, both the worst-case time complexity and the necessary memory space of the IBCL algorithm are the same as those of the BCL algorithm.

4. Comparison

Because the ICTCL algorithm is the most efficient in pixel-based label-equivalent-resolving algorithm, the IRCL is the most efficient run-based algorithm, the IBCL algorithm is the most efficient block-based algorithm, hereafter, we will only compare the HOL algorithm, the CTL algorithm, the ICTCL algorithm, the IRCL algorithm, and the IBCL algorithm. The source codes of the CTL algorithm and those of the IBCL were download from the authors' websites

Table 1
Time complexity and necessary space of the state-of-the-art connected-component labeling algorithms.

Algorithm	Time complexity	Memory space
CTL	$O(N^2)$	0
HOL	$O(N^2)$	0
IRCL	$O(N^2)$	$3 \times N^2/2$
ICTCL	$O(N^2)$	$3 \times N^2/4$
IBCL	$O(N^2)$	N^2

[71] and [72], respectively. The source codes of the IRCL algorithm and those of the ICTCL were provided by their authors. On the other hand, because the HOL algorithm can be implemented easily according to the pseudo code provided by the author, we implemented the HOL algorithm by ourselves.

Our experiments were performed on a Lenovo QiTianM4350 by use of an Intel Core i5-3470 CPU @ 3.20 GHz processor and 4GB RAM, and the OS is ubuntu14.04 64bit. All algorithms used in our test were implemented in C language and compiled by the GNU C complier (version 4.8.4) with the option -O3. All experimental results presented in this section were obtained by averaging of the execution time for 5000 runs.

4.1. Time complexity and necessary memory space

According to the analysis given in Section II and Section III, the time complexity and the necessary memory space for the five state-of-the-art connected-component labeling algorithms are shown in Table 1.

For time complexity, all the five state-of-the-art algorithms are linear on image sizes. Label-propagation-based algorithms, i.e., the HOL algorithm and the CTL algorithm need no memory. On the other hand, for equivalent-label-based algorithms, the memory space necessary for the IBCL algorithm is the same as the image size. The ICTCL algorithm and the IRCL algorithm need a little more and smaller than the IBCL algorithm, respectively.

4.2. Experimental results on noise images

Noise images with four sizes (128×128, 256×256, 512×512, 1024×1024 pixels) are used for test. For each size, we threshold a same size image containing uniform noise with random values from 0 to 1000 with 41 different threshold values from 0 to 1000 in steps of 25. Thus, 41 noise images whose densities (i.e., the percentages of foreground pixels) are from 0.05% to 99.22% will be generated. Because foreground pixels in such noise images have complex connectivity, we can make severe evaluations of algorithms on these images. Nine noise images with densities 0.1, 0.2, ..., and 0.9, respectively, are shown in Fig. 12.

4.2.1. Execution times versus image sizes

Because noise images with different sizes have good similarities, we use all noise images to test the linearity of the execution times of various algorithms versus image sizes. The results for maximum execution times and average execution times on different size noise images are shown in Fig. 13(a) and (b), respectively, where *ms* is the abbreviation of millisecond. Notice that the right vertical axle in each figure is for the CTL algorithm, and the left vertical axle is for the other algorithms.

From Fig. 13, we can find that all algorithms are linear on image sizes, and for either the maximum execution time or the average execution time, the order of the performances of the algorithms is: the CTL algorithm < the IRCL algorithm < the HOL algorithm < the IRCL algorithm < the ICTCL algorithm.

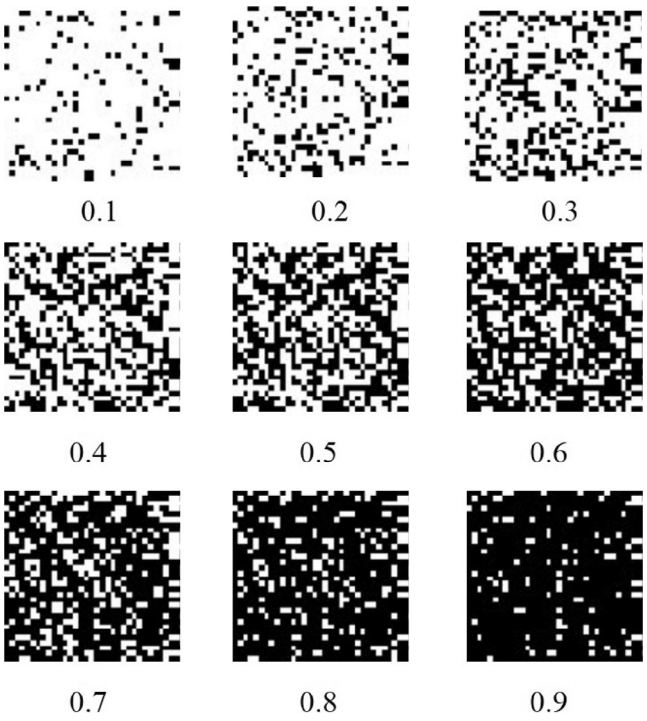


Fig. 12. Noise images with various densities.

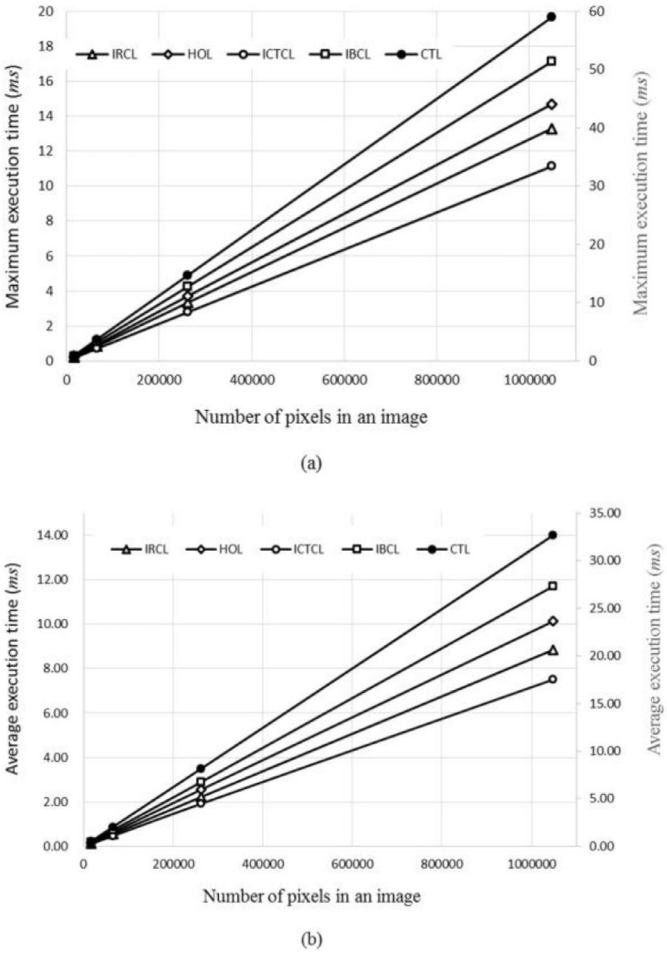


Fig. 13. Execution times versus image sizes: (a) maximum execution times, and (b) average execution times.

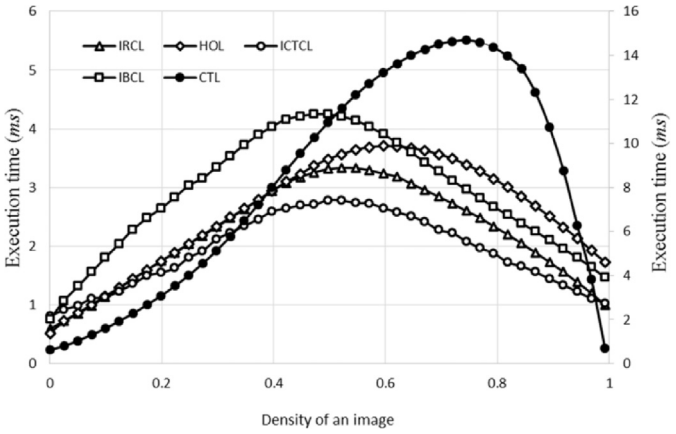


Fig. 14. Executing times versus the density of an image.

Table 2
Execution time (*ms*) versus the density of a 512×512-sized noise image.

Density	IRCL	CTL	HOL	ICTCL	IBCL
0.000	0.595	0.622	**0.517**	0.804	0.754
0.025	0.726	0.803	**0.724**	0.920	1.073
0.050	**0.853**	1.042	0.866	0.992	1.318
0.075	**0.990**	1.306	1.006	1.108	1.562
0.099	**1.134**	1.595	1.153	1.136	1.802
0.124	1.287	1.933	1.295	**1.232**	2.043
0.149	1.443	2.282	1.448	**1.360**	2.287
0.174	1.600	2.681	1.599	**1.496**	2.483
0.199	1.750	3.081	1.745	**1.552**	2.645
0.224	1.894	3.529	1.894	**1.628**	2.836
0.248	2.037	4.010	2.043	**1.808**	3.033
0.273	2.179	4.543	2.192	**1.920**	3.156
0.298	2.330	5.133	2.342	**2.116**	3.349
0.323	2.480	5.778	2.491	**2.232**	3.537
0.348	2.635	6.482	2.639	**2.348**	3.730
0.372	2.786	7.228	2.790	**2.460**	3.894
0.397	2.944	8.010	2.940	**2.592**	4.040
0.423	3.081	8.790	3.098	**2.644**	4.161
0.448	3.178	9.541	3.229	**2.696**	4.215
0.472	3.258	10.257	3.373	**2.708**	4.249
0.497	3.314	10.945	3.484	**2.784**	4.246
0.522	3.334	11.593	3.569	**2.780**	4.212
0.547	3.331	12.201	3.642	**2.736**	4.142
0.571	3.296	12.725	3.688	**2.732**	4.038
0.596	3.236	13.222	3.706	**2.648**	3.908
0.621	3.182	13.635	3.703	**2.580**	3.761
0.646	3.065	14.006	3.677	**2.508**	3.602
0.670	2.958	14.277	3.633	**2.416**	3.433
0.695	2.847	14.513	3.569	**2.276**	3.275
0.720	2.726	14.639	3.495	**2.224**	3.115
0.745	2.599	14.697	3.389	**2.084**	2.963
0.769	2.485	14.597	3.279	**1.968**	2.818
0.794	2.336	14.363	3.151	**1.880**	2.671
0.818	2.202	13.969	3.004	**1.728**	2.532
0.843	2.047	13.398	2.854	**1.660**	2.394
0.868	1.893	12.311	2.683	**1.560**	2.250
0.893	1.730	10.740	2.508	**1.444**	2.104
0.918	1.562	8.777	2.326	**1.340**	1.954
0.943	1.385	6.270	2.136	**1.224**	1.806
0.968	1.201	3.850	1.928	**1.112**	1.646
0.992	**1.002**	1.716	1.733	1.032	1.471

4.2.2. Execution times versus image densities

We used 512×512-sized noise images for testing the execution times versus image densities for all algorithms. The results are shown in Fig. 14 (where similar as in Fig. 13 the right vertical axle is for the CTL algorithm, and the left vertical axle is for the other algorithms) and Table 2 (where the bold numbers indicate the smallest execution time on the corresponding noise image).

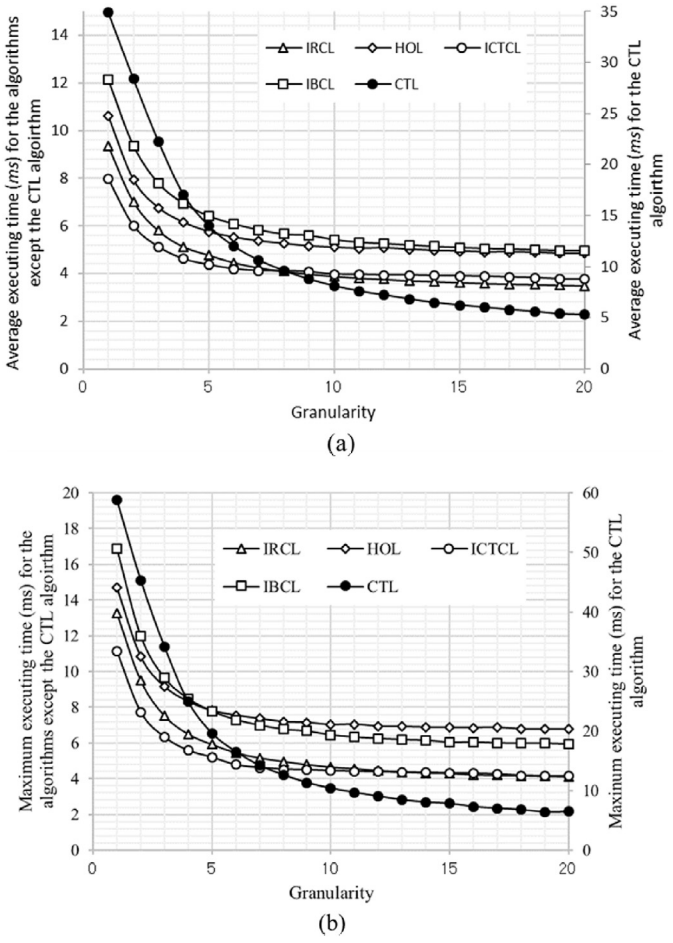


Fig. 15. Executing times versus granularities.

4.2.3. Execution times versus pixel granularities

As mentioned in Ref. [56], general noise images might be biased and thus favor some CCL algorithms. In order to analyze algorithms' behavior depending on some image properties: size of connected components and size of the smaller element compared to the algorithm neighborhood scale, we extend random images by changing the pixel granularity from 1 to 20. For each granularity g, where a block of object pixels has a size of $g \times g$, 19 g-random images, each of which has a size of 1024×1024, are created for each density which changes from 5 to 95 in steps of 5. Thus, $20 \times 19 = 380$ images are created in all.

The results for average execution times and maximum execution times on the above granularity images of the five algorithms are shown in Fig. 15(a) and (b) (where *ms* is the abbreviation of millisecond, and the right vertical axle is for the CTL algorithm, and the left vertical axle is for the other algorithms) and Table 3 (where g means granularity, the bold numbers indicate the smallest execution time on the corresponding granularity image), respectively.

From Fig. 15(a) and (b), and Table 3, we can find that for all five algorithms, both the average executing time and the maximum executing time decrease with the increase of the granularity. The CTL algorithm takes more time than the others in all cases. For both the average executing times and the maximum executing times, those of the HOL algorithm and the IBCL algorithm are close, and those of the IRCL algorithm and the ICTCL algorithm are close. Moreover, for both the average executing times and the maximum executing times, the ICTCL is the most efficient for low granular-

Table 3
Average executing times and maximum executing times versus granularities.

g	Average executing time					Maximum executing time				
	CTL	IRCL	HOL	ICTCL	IBCL	CTL	IRCL	HOL	ICTCL	IBCL
1	34.88	9.35	10.62	**7.98**	12.12	58.76	13.27	14.73	**11.15**	16.88
2	28.39	7.02	7.94	**6.02**	9.37	45.33	9.50	10.84	**7.74**	12.00
3	22.24	5.80	6.76	**5.12**	7.78	34.15	7.53	9.16	**6.34**	9.65
4	17.07	5.13	6.15	**4.62**	6.94	24.89	6.50	8.32	**5.58**	8.48
5	14.01	4.78	5.76	**4.37**	6.42	19.62	5.92	7.81	**5.20**	7.77
6	12.01	4.46	5.52	**4.20**	6.08	16.54	5.45	7.55	**4.78**	7.30
7	10.63	4.24	5.36	**4.12**	5.83	14.23	5.16	7.36	**4.62**	6.99
8	9.62	**4.09**	5.26	4.10	5.66	12.65	4.97	7.19	**4.53**	6.77
9	8.79	**3.98**	5.15	4.08	5.60	11.34	4.79	7.14	**4.50**	6.66
10	8.13	**3.89**	5.10	3.97	5.43	10.39	4.65	7.02	**4.45**	6.45
11	7.62	**3.81**	5.05	3.97	5.31	9.70	4.57	7.05	**4.42**	6.32
12	7.23	**3.76**	5.07	3.94	5.26	9.07	4.46	6.93	**4.38**	6.25
13	6.84	**3.70**	5.01	3.94	5.20	8.46	4.38	6.92	**4.37**	6.19
14	6.49	**3.67**	4.96	3.91	5.13	8.07	**4.33**	6.88	4.34	6.15
15	6.25	**3.62**	4.94	3.90	5.09	7.86	**4.30**	6.86	4.32	6.04
16	6.02	**3.59**	4.90	3.88	5.05	7.31	**4.22**	6.83	4.30	6.05
17	5.80	**3.56**	4.90	3.85	5.03	7.05	**4.19**	6.81	4.26	6.00
18	5.62	**3.54**	4.89	3.83	5.00	6.82	**4.18**	6.78	4.18	5.98
19	5.39	**3.51**	4.84	3.77	4.97	6.49	**4.14**	6.77	4.16	5.98
20	5.31	**3.49**	4.84	3.76	4.96	6.49	**4.11**	6.77	4.14	5.93

Table 4
Maximum, mean and minimum execution times (ms) of algorithms on various kinds of images.

Image type		IRCL	CTL	HOL	ICTCL	IBCL
Natural	Max.	1.60	5.55	1.87	**1.50**	2.26
	Mean	**1.07**	2.55	1.28	1.09	1.50
	Min.	**0.69**	0.88	0.73	0.84	0.97
Medical	Max.	**1.07**	3.06	1.41	1.10	1.52
	Mean	**1.06**	2.47	1.27	1.08	1.48
	Min.	**0.80**	1.45	0.91	0.87	1.12
Textural	Max.	1.56	1.95	1.85	**1.37**	1.95
	Mean	**0.92**	1.94	1.17	0.99	1.31
	Min.	**0.97**	1.41	1.54	1.01	1.41
Artificial	Max.	1.13	4.21	1.98	**1.04**	1.42
	Mean	0.66	2.45	0.98	**0.64**	0.93
	Min.	0.26	0.39	0.32	**0.23**	0.38

ity images, and the IRCL algorithm is the most efficient for high granularity images.

4.3. Experimental results on realistic image sets

The realistic image set consists of 50 natural images including landscape, aerial, fingerprint, portrait, still-life, snapshot, and text images are obtained from the Standard Image Database (SIDBA) developed by the University of Tokyo [73] and the image database of the University of Southern California [74], respectively. In addition, seven texture images were downloaded from the Columbia-Utrecht Reflectance and Texture Database [75], and 25 medical images were obtained from a medical image database of The University of Chicago. All of these images were 512×512 pixels in size, and were binarizied by means of Otsu's method [76]. Moreover, four artificial images with specialized shape (stair-like, spiral-like, saw-tooth-like, checker-board-like) patterns.

The maximum, mean, and minimum execution times of each algorithm on each kind of images are shown in Table 4, where the bold numbers indicate the smallest execution time on the corresponding case.

From Table 4, we can find that (1) the performance of the IRCL algorithm and that of the ICTCL algorithm are close and better than the other algorithms; (2) the performance of HOL algorithm and the IBCL algorithm are close; and (3) the performance of the CT algorithm is the worst in all cases.

Table 5
The number of each kind of images on which an algorithm is the fastest.

Image type	Total	IRCL	CTL	HOL	ICTCL	IBCL
Natural	50	31	0	0	19	0
Medical	25	23	0	0	2	0
Textural	7	2	0	0	5	0
Artificial	4	1	0	1	2	0
Noise	41	4	0	2	35	0

The execution times (in ms) for the selected six images are illustrated in Fig. 16, where the foreground pixels are displayed in black, and where the value of D indicates the density of an image.

The number of each kind of images on which an algorithm are the fastest are shown in Table 5, where the noise images are those images used in Section 4.2.2.

From Table 5, we can find that the HOL algorithm is the most efficient for few cases, and the IRCL algorithm and the ICTCL algorithm are the most efficient on about half of images, respectively. Moreover, the IRCL algorithm is especially efficient on the medical images and the natural images, and the ICTCL algorithm is especially efficient on the noise images and the natural images.

5. Relevant issues

In this section, we introduce issues regarding to parallel implementation, hardware implementation, and extension of CCL algorithms for 3D and/or n-D images.

5.1. Parallel implementation

As mentioned in Introduction, CCL processing is a time-consuming work. For many real-time applications, the speed of a CCL algorithm on an ordinary computer might not be sufficient. In such cases, we might need to do CCL processing in parallel.

There are some CCL algorithms proposed for parallel computation. The algorithm proposed in Ref. [20] labels connected components in a binary image as follows: (1) Compute a seed pixel for each object in the image by shrinking operation; (2) Assign a unique label to each seed pixel; (3) Propagate each label to all pixels of the corresponding object. In the worst case, this algorithm needs to do $N/2$ times shrinking operations for computing seed pixels of objects.

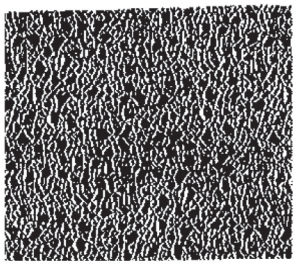
D: 0.543 IRCL: 1.339
CTL: 3.790 HOL: 1.495
ICTCL: 1.304 IBCL: 1.896

D: 0.609 IRCL: 1.042
CTL: 1.850 HOL: 1.467
ICTCL: 1.064 IBCL:1.489

D: 0.177 IRCL: 0.700
CTL: 0.877 HOL: 0.815
ICTCL: 0.868 IBCL: 0.985

D: 0.151 IRCL: 1.282
CTL: 3.249 HOL: 1.302
ICTCL: 1.180 IBCL:1.724

D: 0.461 IRCL: 0.974
CTL: 1.727 HOL: 1.415
ICTCL: 1.012 IBCL: 1.403



D: 0.545 IRCL: 1.477
CTL: 1.947 HOL: 1.759
ICTCL: 1.368 IBCL:1.947

Fig. 16. The density and execution times (ms) for the selected six images: (a) a fingerprint image; (b) a portrait image; (c) a snapshot image; (d) a text image; (e) a medical image; (f) a texture image.

The algorithm proposed in Ref. [77] is also a CCL algorithm proposed for parallel computation. This algorithm takes the index of an object pixel in the raster scan as its initial provisional label. Then for each object pixel, this algorithm checks all object pixels in the 8-connected directions until a background is met, find the minimal label among the pixels, and replaces all labels with the minimal label among them. This process repeats until no label changes. In the worst case, $N/2$ times repeats are needed. Moreover, it does not generate consecutive labels. The algorithm proposed in Ref. [78] can be considered as a simple version of the above algorithm, which only processes horizontal (row) and vertical (column) directions.

$b(x-3, y)$	$b(x-2, y)$	$b(x-1, y)$	$b(x, y)$	$b(x+1, y)$
$b(x-3, y+1)$	$b(x-2, y+1)$	$b(x-1, y+1)$	$b(x, y+1)$	$b(x+1, y+1)$
$b(x-3, y-1)$	$b(x-2, y-1)$	$b(x-1, y-1)$	$b(x, y-1)$	$b(x+1, y-1)$

Fig. 17. An example for processing different rows in a delay.

The largest advantage of these algorithms is that no memory is necessary. However, in the worst case, they need to process an image $N/2$ times. Therefore, unless memory space is strictly limited or memory access cost is extremely high, these algorithms should not be considered.

Another method for parallel CCL computation is to parallelize a CCL labeling algorithm for ordinary computer. Because all label-propagation algorithms need to find an unlabeled object pixel first for labeling each object, and there is not a method for simultaneously finding an unlabeled pixel for all objects,[3] they are not suitable for parallelization.

On the other hand, because label-equivalence-based CCL algorithms process pixels in a binary image in raster scan order (a regular way), they are suitable for parallelization. There are two methods for parallelizing a two-scan CCL algorithm. One is based on divide-and-conquer method [21–23,79–82] which works as follows:

(1) Divide the given $N \times N$ image into k pieces $(N \times N)/k$ subimages;
(2) Label each subimage provisionally by use of the first scan of the selected CCL algorithm for ordinary computer in parallel;
(3) Merge the results of subimages by processing the boundaries of those subimages;
(4) Replace each label with its representative label in parallel.

By this method, for labeling an $N \times N$ image, if we ignore the cost for merging subimages, the running time will be T/p by use of p processors, where T is the running time for labeling the image by use of the corresponding CCL algorithm on an ordinary computer with the same performance processor.

The other method for parallelizing a two-scan CCL algorithm is to process different rows on different time points [27,83]. For example, as shown in Fig. 17, in the first scan of a two-scan CCL algorithm, after processing $b(x, y)$, it is possible to process $b(x-1, y+1)$, and after processing $b(x-1, y+1)$, it is possible to process $b(x-2, y+2)$, and so on. In other words, a pixel can be processed as soon as its top-right neighbor pixel has been processed. This will generate a delay of processing a pixel for each row; thus, N delays for processing an $N \times N$ image. If we ignore the delays for processing different rows, the running time for labeling an $N \times N$ image by use of p processors will be T/p, where T is the running time for labeling the image by use of the corresponding CCL algorithm on an ordinary computer with the same performance processor.

[3] Although we can process different subregions in parallel to find unlabeled pixels for different objects, many pixels of an object might be found unlabeled simultaneously which will be assigned different labels. Thus, there would be a lot of redundancy if we made label-propagating processes starting from these pixels simultaneously.

According to above analysis, both methods can approximately achieve linear property on the number of processors. For other issues about parallel connected-component labeling, such as CUDA implementation, GPU implementation, usage of shared memory and global memory, readers can consult Refs. [77,78,82,84–86].

5.2. Hardware implementation

For real-time embedded image processing systems, where images data are usually streamed in, the hardware implementation of a CCL algorithm is necessary. Because label propagation CCL algorithms access an image in an irregular way, they are also not suitable for hardware implementation. On the other hand, because label-equivalence CCL algorithms process pixels in an image in raster scan order, they are suitable for hardware implementation with streamed image data. There are pixel-based CCL hardware implementation [34–36,87] and run-based CCL hardware implementation [88,89], almost all of which are realized by use of FPGA. Moreover, although there were few hardware implementations based on multi-scan CCL algorithms before 2003 [34,35], hardware implementations are made almost based on two-scan CCL algorithms nowadays.

In the cases where an external storage such as DRAMs are available and the initial (immediate, labeled) image can be stored in a buffer, we can completely implement a label-equivalence CCL algorithm by use of FPGA. However, for the cases where only internal memory is available, it is necessary to use memory as small as possible. If connected components (objects) in the image to be processed are simple and the number of objects is smaller, we can modify a CCL algorithm for hardware implementation with small memory [90]. On the other hand, if image data are given in stream, and there is no buffer for storing image as in many real-time image processing systems, it might be difficult for implementing a label-equivalence CCL algorithm by use of FPGA.

In fact, in many real-time image processing systems, instead of a labeled image, only simple features of connected components (objects) such as areas and centroids, are needed. In such cases, we can accumulate shape features of connected components as follows, where $A[l]$ and $(X[l], Y[l])$ are for the area and the centroid the object with the representative label l, and the equivalent-label-set strategy is used for resolving label equivalence.

(1) For each object pixel $b(x, y)$ being processed in the first scan of a label-equivalence-based CCL algorithm, let l be the label assigned to $b(x, y)$, we do:

$$A[l] \leftarrow A[l]+1, X[l] \leftarrow X[l]+x, Y[l] \leftarrow Y[l]+y[l].$$

(2) Whenever merging two equivalent label $S(u)$ and $S(v)$ into $S(w)$, we do:

$$A[w] \leftarrow A[u]+A[v], X[w] \leftarrow X[u]+X[v], Y[w] \leftarrow Y[u]+Y[v].$$

According to the equivalent-label-set strategy, after the first scan, all provisional label assigned to an object will be combined in an equivalent label set with the same representative label. Thus, for example, for the equivalent label set with the representative label t, i.e., $S(t)$, the area and the centroid of the corresponding object should be $A[t]$ and $(X[t]/A[t], Y[t]/A[t])$, respectively. In this way, we can calculate areas and centroids of objects in the given image by single pass without a labeled image (without the second scan).

Based such an idea, many single-pass CCA algorithms have been proposed [91–96]. In the cases where the number of objects is small and the shapes of objects is simple, only small memory is necessary. However, for general cases, the maximum number of connected components is $N \times N/4$ for an $N \times N$ image, the internal memory of an FPGA might still be insufficient.

Other issues on hardware implementation of CCL algorithms such as latency reduction, circuit design, memory arrangement, and implementing strategies, readers can consult Refs. [90,91,93,94,97].

5.3. CCL algorithms for 3D and/or n-D images

There were some CCL algorithms proposed for 3D images. There are also label-propagation CCL algorithms and label-equivalence CCL algorithms.

In Ref. [39], Udupa proposed a label-propagation CCL algorithm for 3D images. For each unlabeled voxel in the surface of an object, it assigns the voxel a new label and then propagates the label to all voxels on the surface of the object by tracing the boundary surface of the object, and then assigns the same label to all voxels enclosed in the boundary surface. The CTL algorithm introduced in Section 2.1 should be a version of this algorithm for 2D images.

In Ref. [41], Hu et al. proposed an iterative recursion CCL algorithm for 3D images. For each unlabeled object pixel be found, it assigns the pixel a new label. Then, for each object pixel being processed, it propagates the label to all object pixels in a defined region, and calls this procedure recursively for all unprocessed object pixels on the boundary of the region. The experimental results showed that this algorithm is more efficient than the above algorithm proposed in [39].

On the other hand, the algorithm proposed in Ref. [40] is a label-equivalence one. It uses the equivalent label table to handle label equivalence. This algorithm needs vast memory space and is not as efficient as the one proposed in Ref. [41].

He et al. proposed two label-equivalence CCL algorithms for 3D images [42]. One is pixel-based, which is an extension of the CCL algorithm for 2D images proposed in Ref. [47]. It uses the equivalent-label-set strategy for resolving label equivalence, and proposed an efficient order for checking 26 voxels in the mask. The other one is run-based, which is an extension of the CCL algorithm for 2D images proposed in Ref. [45]. The experimental results showed that both the two algorithms are much more efficient than the label propagation algorithm proposed in Ref. [41]. Moreover, Sutheebanjard [98] applied decision tree strategy for extending the CCL algorithm for 2D images proposed in Ref. [48], and achieved the efficiency close to the pixel-based one proposed in Ref. [42].

In fact, any CCL algorithm for 2D images can be easily extended to a corresponding one for n-D images. The general method for extending a label-propagation CCL algorithm for 2D images to a corresponding one for n-D images is as follows:

(1) Find an unlabeled object pixel by searching the given image in some way (usually raster scan);
(2) Assign the pixel a new label;
(3) Propagate the label to all object pixels connected to the pixel by use of the similar strategy as used in the label-propagation CCL algorithm for 2D images.

On the other hand, the general method for extending a label-equivalence CCL algorithm for 2D images to a corresponding one for n-D images is as follows:

(1) Provisional label assignment. For each current object pixel (or super-pixel) in the raster scan, check whether there are labels (object pixels or super-pixels) in the processed region neighbored to the current object pixel (super pixel). If there are, assign any of them the current pixel (super pixel), otherwise assign a new label to the current pixel (super-pixel);
(2) Equivalent label recording and resolving. Record all labels in the in the processed region neighbored to the current object pixel (super pixel) as equivalent labels, and resolve label

equivalence by use of the same strategy used in the label-equivalence CCL algorithm for 2D images;

(3) Label replacement. Scan the given image once again or use the recorded super-pixel data, assign to each object pixel the representative label of the provisional label assigned to the pixel (the super-pixel that the pixel belongs).

Of course, as shown in Ref. [42], in order to enhance the efficiency of a CCL algorithm for 3D or n-D images, special strategies relative to connectivity of 3D or n-D images should be considered.

6. Concluding remarks and future work

In this paper, we reviewed most state-of-the-art connected-component labeling algorithms presented in the last decade, introduced the main strategies and algorithms. Summing up experimental results on various kinds of images, the IRCL algorithm and the ICTCL algorithm are efficient in almost cases. However, in the cases where we only need to find an object in an interesting area of an image, the HOL algorithm should be selected, and in the cases where contours of objects in an image are also need to be extracted, the CTL algorithm should be selected.

We also discussed parallel implementation, hardware implementation, and extension of CCL algorithms. Because label propagation CCL algorithms access pixels in a binary image in an irregular way, they are not suitable for parallel implementation and hardware implementation. On the other hand, label equivalence CCL algorithms process pixels in the raster scan order, they are suitable for parallelization. Moreover, a label equivalence CCL algorithm for 2D images can be easily extended for n-D images.

For future work relative to the CCL problem, the following issues might be considered:

(1) Find efficient methods for merging two equivalent label sets for equivalent-label-set strategy. In the current method, when merging an equivalent label set $S(u)$ into another equivalent label set $S(v)$, the representative label of each label in $S(u)$, i.e., u, will be reset to v. If we can always merge the shorter one into the longer one, we can reduce the operation for resetting the representative label. This will be especially efficient for large complicated images, 3D and/or m-D images.

(2) Propose new label-equivalence resolving strategies that are suitable for parallel implementation and hardware implementation. All of the existing label-equivalence resolving strategies consist of sequential operations, which are inefficient and unsuitable for parallel implementation and hardware implementation.

(3) Parallel implementation of the IRCL algorithm, and the ICTCL algorithm, the IBCL algorithm on multiple GPUs to further shorten the running time of CCL processing;

(4) Hardware and parallel implementation of the ICTCL algorithm, the BCL algorithm, and the IRCL algorithm on multiple FPGAs to further enhance the efficiency of CCL processing for embedded image processing systems;

(5) Propose new strategies for parallel implementation of state-of-the-art CCL algorithms, e.g., processing borders of subimages first, and then processing subimages;

(6) Propose new strategies for hardware implementation of state-of-the-art CCL algorithms, e.g., designing a circuit for processing a region as larger as possible at a time;

(7) Propose new single-pass algorithms for connected component analysis that can calculate more shape features such as perimeter, circularity, and Euler number, and consider the corresponding parallel implementation and hardware implementation.

(8) Formulate a standard of selection of different CCL or CCA algorithms for special cases in real-time image processing systems. For example, if there are only very few BLOBs (Binary Large OBjects, in which there is usually no hole) in images, multi-scan label-equivalence-based CCL algorithms, which do not need memory and process an image regularly, will be very efficient and suitable for parallel implementation and hardware implementation.

(9) Make comparison on benchmarks for running times of state-of-in-the-art CCL algorithms for different computer architectures.

(10) Efficient parallel implementation and hardware implementation of state-of-the-art CCL algorithms for 3-D and n-D images.

Acknowledgments

We would like to thank our editor and the anonymous referees for their valuable comments, which greatly improved this paper. This work was supported by the National Natural Science Foundation of China under Grant No. 61471227, No. 6160127 and No. 61603234, and the Ministry of Education, Science, Sports and Culture, Japan, Grant-in-Aid for Scientific Research (C), 26330200, 2014.

References

[1] R.C. Gonzalez, R.E. Woods, Digital Image Processing, Addison-Wesley, Reading, Massachusetts, 1993.
[2] A. Rosenfeld, A.C. Kak, Digital Picture Processing, 2, second ed., Academic Press, San Diego, CA, 1982.
[3] R. Szeliski, Computer Vision: Algorithms and Applications, Springer, 2011.
[4] A. Rosenfeld, J.L. Pfalts, Sequential operations in digital picture processing, J. ACM 13 (4) (1966) 471–494.
[5] C.R. Dyer, Computing the Euler number of an image from its quadtree, Comput. Graph. Image Process. 13 (3) (1980) 270–276.
[6] C.L. Jackins, S.L. Tanimoto, Octrees and their use in representing 3D objects, Comput. Graph. Image Process. 14 (3) (1980) 249–270.
[7] H. Samet, Connected component labeling using quadtrees, J. ACM 28 (3) (1981) 487–501.
[8] I. Gargantini, Separation of connected component using linear quad- and oct-trees, in: Proc. 12th Conf. Numerical Mathematics and Computation, 37, Winnipeg, University of Manitoba, 1982, pp. 257–276.
[9] M. Taminen, H. Samet, Efficient octree conversion by connectivity labeling, in: Proc. SIGGRAPH 84 Conf., Minneapolis, MN, 1984, pp. 43–51.
[10] H. Samet, The quadtree and related hierarchical data structures, Comput. Surv. 16 (2) (1984).
[11] H. Samet, H. Tamminen, Computing geometric properties of images represented by linear quadtrees, *IEEE Trans. PAMI* PAMI- 7 (2) (1985) 229–240.
[12] H. Samet, M. Tamminen, An improved approach to connected component labeling of images, in: Proc. IEEE Conf. Computer Vision and Pattern Recognition, Miami, Florida, 1986, pp. 312–318.
[13] H. Samet, M. Taminen, Efficient component labeling of images of arbitrary dimension represented by linear bintrees, IEEE Trans. Pattern Anal. Mach. Intell. PAMI- 10 (4) (1988) 579–586.
[14] J. Hecquard, R. Acharya, Connected component labeling with linear octree, Pattern Recognit. 24 (6) (1991) 515–531.
[15] M.B. Dillencourt, H. Samet, M. Tamminen, A general approach to connected–component labeling for arbitrary image representations, J. ACM 39 (2) (1992) 253–280.
[16] D.S. Hirschberg, A.K. Chandra, D.V. Sarwate, Computing connected components on parallel computers, Commun. ACM 22 (8) (1979) 461–464.
[17] D. Nassimi, S. Sahani, Finding connected components and connected ones on a mesh connected parallel compute, SIAM J. Comput. 9 (4) (1980) 744–757.
[18] Y. Schiloach, U. Vishkin, An O (log$n$) parallel connectivity algorithm, J. Algorithms 3 (1982) 57–67.
[19] L.W. Tucker, Labeking connected components on a massively parallel tree machine, in: Conf. in: *Proc. IEEE Computer Vision and Pattern Recognition*, Miami, Florida, 1986, pp. 124–129.
[20] M. Manohar, H.K. Ramapriyan, Connected component labeling of binary images on a mesh connected massively parallel processor, Comput. Vis. Graph. Image Process. 45 (2) (1989) 133–149.
[21] H.M. Alnuweiri, V.K. Prasanna, Parallel architectures and algorithms for image component labelling, IEEE Trans. Pattern Anal. Mach. Intell. 14 (1992) 1014–1034.
[22] K.P. Bekhale, P. Banerjee, "Parallel algorithms for geometric connected component labeling on a hypercube multiprocessor, IEEE Trans. Comput. 41 (6) (1992) 699–709.

[23] S. Olariu, J.L. Schwing, J. Zhang, Fast component labelling and convex hull computation on reconfigurable meshes, Image Vis. Comput. 11 (7) (1993) 447–455.
[24] A. Choudhary, R. Thakur, Connected component labeling on coarse grain parallel computers: an experimental study, J. Parallel Distrib. Comput. 20 (1994) 78–83.
[25] P. Bhattacharya, Connected component labeling for binary images on a reconfigurable mesh architectures, J. Syst. Archit. 42 (4) (1996) 309–313.
[26] K.B. Wang, T.L. Chia, Z. Chen, Parallel execution of a connected component labeling operation on a linear array architecture, J. Inf. Sci. Eng. 19 (2003) 353–370.
[27] C. Lin, S. Li, T. Tsai, A scalable parallel hardware architecture for connected component labeling, in: *Image Processing (ICIP), 2010 17th IEEE International Conference on*„ 2010, pp. 3753–3756.
[28] Cyrus Harrison, Data-Parallel Mesh Connected Components Labeling and Analysis. Euro Graphics Symposium on Parallel Graphics and Visualization, 2014.
[29] S.W. Yang, M.H. Shen, H.H. Wu, H.E. Chien, P.K. Weng, Y.Y. Wu, VLSI architecture design for a fast parallel label assignment in binary image, circuits and systems, in: ISCAS2005, IEEE International Symposium on, 3, 2005, pp. 2393–2396.
[30] H. Flatt, S. Blume, S. Hesselbarth, T. Schunemann, P. Pirsch, Parallel architecture for connected component labeling based on fast label merging„ in: Application-Specific Systems, Architectures and Processors, International Conference on, 2008, pp. 144–149.
[31] G. Qingyi, T. Takaki, I. Ishii, A fast multi-object extraction algorithm based on cell-based connected components labeling, IEICE Trans. Inf. Syst. 95 (2) (2012) 636–645.
[32] X.D. Yang, Design of fast connected components hardware, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (1988) 937–944 June.
[33] Y. Shin, R. Sridhar, V. Demjanenko, S. Palumbo, A special-purpose content addressable memory chip for real-time image processing", *Solid-State Circuits*, IEEE J. 27 (5) (1992) 737–744.
[34] D. Crookes, K. Benkrid, FPGA implementation of image component labeling, in: Proc. SPIE 3844, Reconfigurable Technology: FPGAs for Computing and Applications, 17, 1999.
[35] K. Benkrid, S. Sukhsawas, D. Crookes, A. Benkrid, An FPGA-based image connected component labeller, in: Field Programmable Logic and Application, Springer, Berlin Heidelberg, 2003, pp. 1012–1015.
[36] D. Lee, S. Jin, P. Thien, J. Jeon, FPGA based connected component labeling, in: Control, Automation and Systems." ICCAS'07. International Conference on, IEEE, 2007, pp. 2313–2317.
[37] F. Zhao, Z. Zhang, Hardware acceleration based connected component labeling algorithm in real-time ATR system, in: Proc. SPIE 8784, Fifth International Conference on Machine Vision (ICMV 2012): Algorithms, Pattern Recognition, and Basic Technologies, 2013 87841S.
[38] Q. Gu, T. Takaki, I. Ishii, Fast FPGA-based multi object feature extraction, Circuits Syst. Video Technol. IEEE Trans. 23 (1) (2013) 30–45 On page(s).
[39] J.K. Udupa, V.G. Ajjanagadde, Boundary and object labeling in three-dimensional images, Comput. Vis. Graph. Image Process. 51 (3) (1990) 355–369.
[40] L. Thurfjell, E. Bengtsson, B. Nordin, A new three-dimensional connected components labeling algorithm with simultaneous object feature extraction capability, CVGIP 54 (4) (1992) 357–364.
[41] Q. Hu, G. Qian, W.L. Nowinski, Fast connected-component labeling in three dimensional binary images based on iterative recursion, Comput. Vis. Image Understanding 99 (2005) 414–434.
[42] L. He, Y. Chao, K. Suzuki, Two efficient label-equivalence-based connected–component labeling algorithms for three-dimensional binary images, IEEE Trans. Image Process. 20 (8) (2008) 2122–2134.
[43] F. Chang, C.J. Chen, C.J. Lu, A linear-time component-labeling algorithm using contour tracing technique, Comput. Vis. Image Understanding 93 (2004) 206–220.
[44] J. Martin-Herrero, Hybrid object labelling in digital images, Mach. Vis. Appl. 18 (1) (2007) 1–15.
[45] L. He, Y. Chao, K. Suzuki, A run-based two-scan labeling algorithm, IEEE Trans. Image Process. 17 (5) (2008) 749–756.
[46] K. Wu, E. Otoo, K. Suzuki, Optimizing two-pass connected-component labeling algorithms, Pattern Anal. Appl. 12 (2) (2009) 117–135.
[47] L. He, Y. Chao, K. Suzuki, K. Wu, Fast connected-component labeling, Pattern Recognit. 42 (9) (2009) 1977–1987.
[48] L. He, Y. Chao, K. Suzuki, An efficient first-scan method for label-equivalence-based labeling algorithms, Pattern Recognit. Lett. 31 (1) (2010) 28–35.
[49] L. He, Y. Chao, K. Suzuki, A run-based one-and-a-half-scan connected-component labeling algorithm, Int. J. Pattern Recognit. Artif. Intell. 24 (4) (2010) 557–579.
[50] C. Grana, D. Borghesani, R. Cucchiara, Optimized block-based connected components labeling with decision trees, IEEE Trans. Image Process. 9 (6) (2010) 1596–1609.
[51] L. He, X. Zhao, Y. Chao, K. Suzuki, Configuration-transition-based connected–component labeling, IEEE Trans. Image Process. 23 (2) (2014) 943–951.
[52] W. Chang, C. Chiu, J. Yang, Block-based connected-component labeling algorithm using binary decision trees, Sensors 15 (2015) 23763–23787.
[53] X. Zhao, L. He, B. Yao, Y. Chao, A New connected-component labeling algorithm, IEICE Trans. Inf. Syst. E98-D No.11 (2015) 2013–2016.
[54] R. Walczyk, A. Armitage, T.D. Binnie, Comparative study on connected component labeling algorithms for embedded video processing systems, in: *Proceedings of 2010 International Conference on Image Processing, Computer Vision, & Pattern Recognition* (IPCV), 2010 176.
[55] U.H. Hernandez-Belmonte, V. Ayala-Ramirez, R.E. Sanchez-Yanez, A comparative review of two-pass connected component labeling algorithms, in: Mexican International Conference on Artificial Intelligence, Springer, Berlin Heidelberg, 2011, pp. 452–462.
[56] L. Cabaret, L. Lacassagne, L. Oudni, A review of world's fastest connected component labeling algorithms: speed and energy estimation, International Conference on Design and Architectures for Signal and Image Processing, 2014 Oct 2014.
[57] K. Shoji and J. Miyamichi, "Connected component labeling in binary images by run-based contour tracing," *IEICE Trans. Inf. Syst.*, Vol. J83-D-II, No. 4: 1131–1139 (in Japanese).
[58] Y. Shima, T. Murakami, M. Koga, H. Yashiro, H. Fujisawa, A high-speed algorithm for propagation-type labeling based on block sorting of runs in binary images, in: Proc. 10th Int. Conf. Patt. Recogn, 1990, pp. 655–658.
[59] A. Rosenfeld, Connectivity in digital pictures, J. ACM 17 (1) (1970) 146–160.
[60] R.M. Haralick, in: Some Neighborhood Operations, Plenum Press, New York, 1981, pp. 11–35.
[61] K. Suzuki, I. Horiba, N. Sugie, Linear-time connected-component labeling based on sequential local operations, Comput. Vis. Image Understanding 89 (2003) 1–23.
[62] T. Gotoh, Y. Ohta, M. Yoshida, Y. Shirai, Component labeling algorithm for video rate processing, in: Proc. SPIE, Advances in Image Processing, 804, 1987, pp. 217–224.
[63] T. Goto, Y. Ohta, M. Yoshida, Y. Shirai, High speed algorithm for component labeling, *IEICE* Trans. Inf. Syst. J72-D-II (2) (1989) 247–255 (in Japanese).
[64] R.M. Haralick, L.G. Shapiro, in: Computer and Robot Vision, vol. I, Addison-Wesley, Reading, MA, 1992, pp. 28–48.
[65] R. Lumia, L. Shapiro, O. Zungia, A new connected components algorithm for virtual memory computers, Comput. Vis. Graph. Image Process. 22 (2) (1983) 287–300.
[66] R. Tarjan, Efficiency of a good but not linear set union algorithm, J. ACM 22 (2) (1975) 215–225.
[67] R. Tarjan, J. Leeuwen, Worst-case analysis of set union algorithms, J. ACM 31 (2) (1984) 245–281.
[68] Z. Galil, G. Italiano, Data structures and algorithms for disjoint set union problems, ACM Comput. Surv. 23 (3) (1991) 319–344.
[69] C. Fiorio, J. Gustedt, Two linear time union-find strategies for image processing, Theor. Comput.Sci. 154 (2) (1996) 165–181.
[70] L. He, Y. Chao, K. Suzuki, A linear-time two-scan labeling algorithm, *IEEE International Conference on Image Processing* (ICIP), 2007 pp.V-241-V-244, Sept. 2007.
[71] The website for download the source code of the CTL algorithm: http://dar.iis.sinica.deu.tw/Download.
[72] The website for download the source code of the IBCL algorithm: http://j.mp/CCITsourcecode.
[73] http://sampl.ece.ohio-state.edu/data/stills/sidba/index.htm.
[74] http://sipi.usc.edu/database/.
[75] http://www1.cs.columbia.edu/CAVE/software/curet/index.php.
[76] N. Otsu, A threshold selection method from gray-level histograms, IEEE Trans. Syst. Man Cybern. 9 (1979) 62–66.
[77] Y. Soh, H. Ashraf, Y. Hae, I. Kim, Fast parallel connected component labeling algorithms using CUDA based on 8-directional label selection, Int. J. Latest Res. Sci. Technol. 3 (2) (2014) 187–190.
[78] O. Kalentev, A. Rai, S. Kemnitz, R. Schneider, Connected component labeling on a 2D Grid Using CUDA, J. Parallel Distrib. Comput. (2011) 615–620.
[79] J. Park, C. Looney, H. Chen, Fast connected component labeling algorithm using a divide and conquer technique, in: Computers and Their Applications, 4, 2000, pp. 4–7.
[80] R. McColl, O. Green, D. Bader, A new parallel algorithm for connected components in dynamic graphs, in: High Performance Computing (HiPC), 2013 20th International Conference on, IEEE, 2013, pp. 246–255.
[81] S. Gupta, D. Palsetia, M. Patwary, A. Agrawal, A. Choudhary, A new parallel algorithm for two-pass connected component labeling, in: *Parallel & Distributed Processing Symposium Workshops* (*IPDPSW*), *2014 IEEE International Conference on*, IEEE, 2014, pp. 1355–1362.
[82] A. Kasagi, K. Nakano, Y. Ito, Fast image component labeling on the GPU, Bull. Networking Comput. Syst. and Softw. 3 (1) (2014) 76–80.
[83] R. Cypher, J. Sanz, L. Snyder, Algorithms for image component labeling on SIMD mesh-connected computers, IEEE Trans. Comput. 39 (2) (1990) 276–281.
[84] A. Choudhary, R. Thakur, Evaluation of connected component labeling algorithms on shared and distributed memory multi processors, in: *Proceedings of Parallel Processing Symposium, Sixth International*, 1992, pp. 362–365.
[85] K. Hawick, A. Leist, D. Playne, Parallel graph component labelling with GPUs and CUDA, Parallel Comput. 36 (2010) 655–678.
[86] O. Stava, B. Benes, Connected component labeling in CUDA, in: GPU Comput. Gems, 2010, pp. 569–581.
[87] M. Jablonski, M. Gorgon, Handel-C implementation of classical component labelling algorithm, in: *Euromicro Symposium on Digital System Design* (DSD 2004), Rennes, 2004, pp. 387–393.
[88] K. Appiah, A. Hunter, P. Dickinson, J. Owens, A run-length based connected component algorithm for FPGA implementation, in: I*nternational Conference on Field-Programmable Technology*, Taibei, 2008, pp. 177–184.

[89] K. Appiah, A. Hunter, P. Dickinson, H. Meng, Accelerated hardware video object segmentation: from foreground detection to connected components labelling, Comput. Vis. Image Understanding 114 (11) (2010) 1282–1291.
[90] Y. Ito, K. Nakano, Low-latency connected component labeling using an FPGA, Int. J. Found. Comput. Sci. 21 (03) (2010) 405–425.
[91] D. Bailey, C. Johnston, Single pass connected components analysis, in: Proceedings of Image and Vision Computing, Hamilton, New Zealand, 2007, pp. 282–287.
[92] C. Johnston, D. Bailey, FPGA implementation of a single pass connected components algorithm, in: *The 4th IEEE International Symposium on Electronic Design, Test and Applications* (DELTA 2008), Hong Kong, 2008, pp. 228–231.
[93] N. Ma, DG Bailey, CT Johnston, Optimised single pass connected components analysis, in: International Conference on Field-Programmable Technology, Taibei, 2008, pp. 185–192.
[94] D. Bailey, C. Johnston, N. Ma, Connected components analysis of streamed images, in: *Field Programmable Logic and Applications, IEEE International Conference on*, 2008, pp. 679–682.
[95] Y. Li, K. Mei, P. Dong, An efficient and low memory requirement algorithm for extracting image component information, Int. J. Adv. Intell. 3 (2) (2011) 255–267.
[96] F. Zhao, H. Lu, Z. Zhang, Real-time single-pass connected components analysis algorithm, EURASIP J. Image Video Process. 21 (1) (2013) 2013.
[97] N. Ngan, E. Dokladalova, M. Akil, F. Contou-Carrere, Fast and efficient FPGA implementation of connected operators, J. Syst. Archit. 57 (8) (2011) 778–789.
[98] P. Sutheebanjard, Decision tree for 3-D connected components labeling, in: *Information Technology in Medicine and Education (ITME)*, 2012 *IEEE International Symposium on*, 2, 2012, pp. 709–713.

Lifeng He received the B.E. degree from the Northwest Institute of Light Industry, China, in 1982, a second B.E. degree from Xian Jiaotong University, China, in 1986, the M.S. and Ph.D. degrees in AI and computer science from Nagoya Institute of Technology, Japan, in 1994 and 1997, respectively. His research interests include image processing, computer vision, automated reasoning, pattern recognition, string searching, and artificial intelligence.

Xiwei Ren received the B.E. degree and the M.S. degree from the Shaanxi University of Science and Technology, China, in 2003 and 2006, respectively. From 2006 to 2010, he was an assistant engineer in the College of Electrical and Information Engineering at the Shaanxi University of Science and Technology. Since 2010, he has been a senior engineer. His research interests include graph theory, image processing, and artificial intelligence.

Qihang Gao received the B.E. degree from the Shaanxi University of Science and Technology, China, in 2015. Now he is a graduate student in the College of Electrical and Information Engineering at the Shaanxi University of Science and Technology. His research interests include image processing, artificial intelligence and simulation.

Xiao Zhao received the B.E. degree and the M.S. degree from the Shaanxi University of Science and Technology, China, in 2001 and 2006, respectively. From 2001 to 2006, she was an assistant professor in the College of Electrical and Information Engineering at the Shaanxi University of Science and Technology. Since 2007, she has been a Lecturer. Her research interests include image processing, artificial intelligence, and string searching.

Bin Yao received the B.E. degree and the M.S. degree from the Shaanxi University of Science and Technology, China, in 2003 and 2006, respectively. From 2006 to 2010, he was an assistant professor in the College of Electrical and Information Engineering at the Shaanxi University of Science and Technology. Since 2010, he has been a Lecturer. His research interests include graph theory, image processing, and artificial intelligence.

Yuyan Chao received the B.E. degree from the Northwest Institute of Light Industry, China, in 1984, and the M.S. and Ph.D. degrees from Nagoya University, Japan, in 1997 and 2000, respectively. From 2000 to 2002, she was a special foreign researcher of the Japan Society for the Promotion of Science at the Nagoya Institute of Technology. She is a professor at the Nagoya Sangyo University, Japan, and a guest professor at the Shaanxi University of Science and Technology, China. Her research interests include image processing, graphic understanding, CAD, pattern recognition, and automated reasoning.