Published in final edited form as:
IEEE Trans Image Process. 2011 August ; 20(8): 2122–2134. doi:10.1109/TIP.2011.2114352.

Two Efficient Label-Equivalence-Based Connected-Component Labeling Algorithms for 3-D Binary Images

Lifeng He,
Shaanxi University of Science and Technology, Shaanxi 710021, China, and also with Aichi Prefectural University, Nagakute, Aichi 480-1198, Japan

Yuyan Chao, and
Graduate School of Environmental Management, Nagoya Sangyo University, Aichi 488-8711, Japan, and also with the College of Mechanical and Electrical Engineering, Shaanxi University of Science and Technology, Shaanxi, China

Kenji Suzuki [Senior Member, IEEE]
Department of Radiology, Division of the Biological Sciences, The University of Chicago, Chicago, IL 60637 USA

Lifeng He: helifeng@ist.aichi-pu.ac.jp; Yuyan Chao: chao@nagoya-su.ac.jp; Kenji Suzuki: suzuki@uchicago.edu

Abstract

Whenever one wants to distinguish, recognize, and/or measure objects (connected components) in binary images, labeling is required. This paper presents two efficient label-equivalence-based connected-component labeling algorithms for 3-D binary images. One is voxel based and the other is run based. For the voxel-based one, we present an efficient method of deciding the order for checking voxels in the mask. For the run-based one, instead of assigning each foreground voxel, we assign each run a provisional label. Moreover, we use run data to label foreground voxels without scanning any background voxel in the second scan. Experimental results have demonstrated that our voxel-based algorithm is efficient for 3-D binary images with complicated connected components, that our run-based one is efficient for those with simple connected components, and that both are much more efficient than conventional 3-D labeling algorithms.

Index Terms

Connected component; label equivalence; labeling algorithm; run; 3-D binary image

I. Introduction

Labeling connected components in a binary image is one of the most fundamental operations in pattern analysis and recognition, computer vision, image understanding, and machine intelligence [7]. A connected component (an object) in a binary image is a set of foreground elements such that for any two elements u and v in the set, there is at least a connected path

$p_1, \ldots, p_n$ between u and v such that $p_1 = u$, $p_n = v$, and for all $1 \leq i \leq n - 1$, p_i and p_{i+1} are neighboring foreground elements belonging to the set.

This notion can be extended to nonbinary images by changing the property of elements that consist of the set, for example, a gray level, a color, a group of gray levels, or a group of colors. Connected-component labeling is defined as assigning to all elements of each connected component a unique label, i.e., the value of each element of a connected component is the same and is unique to that of any other connected component.

Many algorithms have been proposed for labeling 2-D images. For ordinary computer architectures and pixel-based representation images, there are two classes of labeling algorithms.

1. *Label-equivalence-based algorithms:* These algorithms process an image in the raster-scan direction more than once. In the first scan, a provisional label is assigned to each foreground pixel. All provisional labels assigned to the same connected component are called *equivalent labels*, and the relationships between equivalent labels are called *label equivalences*. Any label equivalence is recorded as soon as it is found. After the first scan and resolving of all label equivalences, which means finding a representative label for each group of equivalent labels, each foreground pixel is relabeled by the representative label for the provisional label assigned to the pixel in the first scan. There are multiscan algorithms [8], a four-scan algorithm [20], and two-scan algorithms [9], [10], [12], [16], [17] in this class.
2. *Searching-and-propagation-based algorithms:* These algorithms first search an unlabeled foreground pixel, assign a new label to it, and then propagate the label to all foreground pixels connected to the pixel in later processing [2], [4].

As image acquisition and manipulation technologies have advanced, 3-D images have been widely used in various image-processing and analysis fields [15], [28], such as medical image analysis and computer-aided diagnosis of medical images [5], [6], [18], [21], [22], [27], as well as computer graphics. Labeling of connected components in 3-D binary images is demanded in many cases, for example, for calculating the volume of an organ, and the volume or shape of a lesion, such as a cancer, polyp, or nodule.

Labeling of connected components in 3-D binary images has been studied from the 1980s. Lumia *et al*. [12] and Shirai [19] proposed label-equivalence-based two-scan labeling algorithms by using an equivalent-label table for recording label equivalences and that for resolving label equivalences, respectively. Thurfjell *et al*. [23] proposed a label-equivalence-based multiscan algorithm, where label equivalences are recorded and resolved through a translation table. On the other hand, Udupa and Ajjanagadde [25] and Borgefors *et al*. [3] proposed searching-and-propagation-based algorithms for 3-D images. Recently, Hu *et al*. [11] proposed two iterative-recursion-based labeling algorithms. The experimental results demonstrated in the paper show that the two algorithms were more efficient than other conventional algorithms. The authors claimed that their algorithms were more efficient than label-equivalence-based algorithms.

This paper presents two label-equivalence-based labeling algorithms for 3-D binary images. One is voxel based, extended from the two-scan algorithm proposed in [10] for labeling 2-D binary images, where an efficient method for deciding the order for checking voxels in the mask is proposed. The other is run based, extended from the run-based two-scan algorithm proposed in [9] for labeling 2-D binary images, and improved by assigning provisional labels to runs (rather than foreground voxels) and also using run data for only labeling (processing) foreground voxels in the second scan (thus, no background voxel is processed in the second scan). The experimental results demonstrated that our voxel-based algorithm is efficient for images with complicated connected components, and the run-based one is efficient for images with simple connected components.

For convenience, we assume that the foreground pixels (voxels) and background pixels (voxels) in a given image are represented by 1 and 0, respectively. As in most labeling algorithms, we assume that all pixels (voxels) on the edges of an image are background pixels (voxels).

II. Label-Equivalence-Based Two-Scan Labeling Algorithms for 2-D Binary Images

For an $N \times M$-size 2-D binary image, we use $p(x, y)$ to denote the pixel at (x, y) in the image, where $1 \leq x \leq N$ and $1 \leq y \leq M$. The label assigned to $p(x, y)$ is denoted as $label(x, y)$. Moreover, we only consider 8-connectivity for connected components.

Label-equivalence-based two-scan labeling algorithms complete labeling in two scans by processing pixels one by one in the raster-scan direction. They need to perform three tasks: 1) assigning to each foreground pixel a provisional label and recording label equivalences, where a data structure is used to record each label equivalence whenever found; 2) resolving label equivalences, which means to find a unique representative label for every group of equivalent labels; and 3) relabeling foreground pixels, i.e., replace the provisional label assigned to each pixel by its representative label. These algorithms can be divided into two classes: pixel- and run-based algorithms.

The first task in pixel-based algorithms is completed by use of the mask shown in Fig. 1 in the first scan, which consists of the four processed neighbor pixels of the current pixel. For each current foreground pixel $p(x, y)$, if there is no foreground pixel in the mask, this means that $p(x, y)$ does not connect with any processed foreground pixels, i.e., at this point, $p(x, y)$ belongs to a connected component consisting of itself only, $p(x, y)$ is assigned a new provisional label, and no label equivalence needs to be recorded. Otherwise, i.e., if there are some foreground pixels in the mask, it is obvious that all foreground pixels in the mask belong to the same connected component; thus, all provisional labels assigned to the foreground pixels in the mask are equivalent labels. The algorithms proposed in [12], [16], and [17] check all pixels in the mask, assign the minimum label in the mask to $p(x, y)$, and record all different labels in the mask as equivalent labels. On the other hand, the algorithm presented in [10] checks pixels in the mask in the optimal order derived by case analysis, assigns the label first found in the mask to $p(x, y)$, and records different provisional labels in the mask as equivalent labels only if they become equivalent due to the existence of $p(x, y)$.

There are mainly three methods for the recording of label equivalences and resolving label equivalences. One is using a 2-D equivalent-label table *EqLabel*(*u*, υ) [12], [16], [17], which is initialized to be 0 for all *u* and υ, to record label equivalences. If provisional labels *a* and *b* are found to be equivalent, then *EqLabel*(*a*, *b*) is set to 1. After the first scan, all groups of equivalent labels can be found by analysis of the equivalent-label table. The main problem of this method is that the size of the equivalent table is very large, proportional to the square of the number of provisional labels in an image; thus, the square of the size of the image,[1] and thus the complexity of the analysis of the equivalent label, is also proportional to the square of the size of the image.

Another method is using binary trees to record label equivalences and using a union-find algorithm [14], [24] to resolve label equivalences. Label equivalences can be resolved partially during the first scan, but a final processing must be made to complete the work after the first scan.

The third method is using equivalent-label sets and a representative-label table to record and resolve label equivalences [10]. In this method, at any point in the first scan, all equivalent labels belonging to a connected component found so far are combined in an equivalent-label set, where the smallest label is referred to as the representative label. The corresponding relationship of a provisional label and its representative label is recorded in a representative table. For convenience, we use *S*(*t*) for the set of provisional labels with *t* as the representative label, and *r_label*[*a*] to represent the representative label of provisional label *a*. In this way, for any provisional label *l* in provisional label set *S*(*t*), we have *r_label*[*l*] = *t*. On the other hand, if *r_label*[*u*] = υ, then we know that provisional label *u* belongs to equivalent-label set *S*(υ).

As we have known, when the current foreground pixel is assigned a new provisional label, say, *NewLabel*, it means that the current foreground pixel does not connect with any foreground pixel that has been scanned before. In other words, up to now, all we know is that the current foreground pixel belongs to a connected component consisting of itself only. Thus, the equivalent-label set corresponding to the connected component is established as *S*(*NewLabel*) = {*NewLabel*}, and the representative label of *NewLabel* is set to itself, i.e., *r_label*[*NewLabel*] ← *NewLabel*. During the first scan, whenever two different provisional labels, *a* and *b*, are found to be equivalent, the label equivalence could be resolved as follows: suppose that *u* and υ belong to equivalent-label sets *S*(*m*) and *S*(*n*), respectively, where *r_label*[*u*] = *m* and *r_label*[υ] = *n*. Then, all provisional labels in *S*(*m*) and *S*(*n*) are known to be equivalent. Therefore, *S*(*m*) and *S*(*n*) should be considered to be combined together. If *m* and *n* are equal, this means that *u* and υ belong to the same equivalent-label set; thus, nothing has to be done. On the other hand, if *m* < *n*, then *S*(*n*) is combined into *S*(*m*), i.e., for each label *w* in *S*(*n*), we set its representative label to *m* by *r_label*[*w*] ← *m*; otherwise, i.e., *m* > *n*, *S*(*m*) is combined into *S*(*n*), i.e., for each label *w* in *S*(*m*), we set its representative label to *n* by *r_label*[*w*] ← *n*. The pseudocode of this processing, denoted as *resolve*(*u*,υ), can be summarized as follows:

[1]For an *N* × *M* binary image, the largest number of provisional labels is *N* × *M*/4.

```
m ← r_label[u];
n ← r_label[υ];
if (m < n)
   S(m) ← S(m) ∪ S(n);
   for each label w ∈ S(n)
      r_label[w] ← m;
   end of for
else if (m > n)
   S(n) ← S(m) ∪ S(n);
   for each label w ∈ S(m)
      r_label[w] ← n;
   end of for
end of if
```

With this method 1) we do not need to calculate the minimum label in the mask, and the average time for checking the pixels in the mask is 2.75, which is less than four times that are required by other pixel- and label-equivalence-based two-scan algorithms (because all of these algorithms assign the minimum label in the mask to the current foreground pixel, and they need to check all 4 pixels in the mask) [10], and 2) any label equivalence will be resolved as soon as it is found in the first scan; therefore, when the first scan is finished, all label equivalences are resolved, i.e., all label equivalences are combined into equivalent-label sets with unique representative labels.

After label equivalences are resolved, the task (3) can be completed by scanning of the image just once again. For example, in the algorithm proposed in [10], this task can be finished as follows:

```
for (x = 2; x < N; x ← x + 1)
   for (y = 2; y < M; y ← y + 1)
      label(x, y) ← r_label[label(x, y)];
   end of for
end of for
```

According to the experimental results shown in [10], the third method is superior to the other two methods for various types of images. For convenience, we denote the algorithm proposed in [10] as the fast connected-component labeling (FCL) algorithm.

On the other hand, the algorithm proposed in [9] is a run-based labeling algorithm, where a *run* means a block of contiguous foreground pixels in a row. The run data can be obtained easily in the first scan. Unlike pixel-based algorithms, which resolve label equivalences between foreground pixels, this algorithm resolves label equivalences between runs. For convenience, we use $r(s, e)$ to denote a run starting from $p(s, t)$ and ending at $p(e, t)$. Thus, for the current run $r(s, e)$ being processed in the raster scan, during the processed runs, a run $r(u, \upsilon)$ that lies in the row immediately above the current row such that it contains one of $p[s - N - 1]$, $p[s - N]$, …, $p[e - N + 1]$ (see Fig.2), i.e., $u \leq e - N + 1$ and $\upsilon \geq s - N - 1$, is 8-neighbored with the current run. The method for recording and resolving the label equivalences used in this algorithm is exactly the same as in the FCL algorithm.

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

On the other hand, if there are runs, e.g., $r_1, \ldots, r_n$, eight-neighbored to $r(s, e)$ in the range, then $r_1, \ldots, r_n$, and $r(s, e)$ belong to the same connected component. Suppose that $l_1, \ldots, l_n$ are the provisional labels assigned to $r_1, \ldots, r_n$, respectively, and $S(u_1), \ldots, S(u_n)$ are the equivalent-label sets containing $l_1, \ldots, l_n$, respectively; then all provisional labels in $S(u_1), \ldots, S(u_n)$ are equivalent labels. Therefore, $S(u_1), \ldots, S(u_n)$ are merged into $S(u)$, where u is the minimum label among $u_1, \ldots,$ and u_n. Moreover, all foreground pixels in the current run $r(s, e)$ are assigned the provisional label l_1. Moreover, after processing of $r(s, e)$, all data of runs that end before or at $p[e - N]$ are removed because such runs cannot be connected with any coming run and therefore are useless for further connectivity detection.

Because this algorithm resolves connectivity between runs, for an image, the number of provisional labels assigned by this algorithm might be much smaller than that assigned by other conventional label-equivalence-based labeling algorithms. This will reduce the computation cost required for resolving label equivalences. Therefore, it is very efficient for images with a large average length of runs.

III. Proposed Two Labeling Algorithms for 3-D Binary Images

For a $U \times V \times W$-size 3-D binary image, we use $v(x, y, z)$ to denote the voxel at (x, y, z) in the image, where $1 \leq x \leq U$, $1 \leq y \leq V$, and $1 \leq z \leq W$. The label assigned to $v(x, y, z)$ is denoted as $label(x, y, z)$.

For 3-D binary images, there are 6-connectivity (i.e., face connectivity), 18-connectivity, (i.e., edge and face connectivity), and 26-connectivity (vertex, edge, and face connectivity). Because 6-connectivity and 18-connectivity are subcases of 26-connectivity, we will consider only 26-connectivity in this paper.

A. Proposed Voxel- and Label-Equivalence-Based Labeling Algorithms

Similarly to the case for labeling 2-D binary images, when using a label-equivalence-based labeling algorithm for labeling 3-D binary images, we need to do three tasks: 1) assign a provisional label to each foreground voxel and find label equivalences in the first scan; 2) record and resolve label equivalences; and 3) relabel foreground voxels.

The tasks of recording and resolving label equivalences, as well as relabeling are not essentially different for any dimension of an image, but the task for assigning a provisional label to each foreground voxel and finding label equivalences in the first scan depends on the dimension of images. As introduced in Section II, the method of using equivalent label

sets and a representative label table for recording and resolving label equivalences proposed in [9] and [10] is the most efficient approach to the best of our knowledge; therefore, we employ this method in our algorithm. Thus, assigning a provisional label to each foreground voxel, and finding and resolving label equivalences, will be done simultaneously in the first scan. For convenience, we call this task the *assigning-finding--resolving task*-.

Similar to label-equivalence-based labeling algorithms for 2-D binary images, when processing a foreground voxel to complete the assigning--finding--resolving task, we need to check the processed voxels neighboring the foreground voxel. The mask for this purpose consists of the 13 processed voxels neighboring the current foreground voxel, as shown in Fig. 3.

For the current foreground voxel, because all foreground voxels in the mask and the foreground voxel belong to the same connected component, and because all provisional labels assigned to voxels of a connected component will be replaced by the same representative label after resolving equivalent provisional labels, we can assign any provisional label in the mask (if any) to the current foreground voxel. If we suppose that the probability of a voxel being a foreground voxel is the same for all voxels in the mask, then the order for checking the voxels in the mask has no influence on the efficiency of assigning provisional labels. On the other hand, the efficiency of finding label equivalences in the mask depends substantially on the order for checking the voxels in the mask.

Let us consider the configuration in the mask shown in Fig. 4 for the current foreground voxel $v(x, y, z)$, where v_2, v_9, and v_{13} are foreground voxels and v_1, $v_3 - v_8$, and $v_{10} - v_{12}$ are background voxels. For convenience, we use l_i to denote the provisional label assigned to v_i, and $l_i = 0$ if v_i is a background voxel.

If we check the voxels in the mask in the order $v_1 \rightarrow v_2 \rightarrow \cdots \rightarrow v_{13}$, denoted as *Order 1*, we will first check v_1. Because v_1 is a background voxel, we go to check v_2. Because v_2 is a foreground voxel, we assign the provisional label assigned to v_2, i.e., l_2, to $v(x, y, z)$. Then, we need to find all foreground voxels in the mask that are not connected with each other without the existence of $v(x, y, z)$, i.e., we should check those voxels in the mask that are not 26 neighbors of v_2, i.e., v_4, v_7, v_{10}, v_{11}, v_{12}, and v_{13} [see Fig. 4(b)]. Because v_4, v_7, v_{10}, v_{11}, and v_{12} are background voxels, we need to do nothing. When we check voxel v_{13}, we find that it is a foreground voxel; therefore, we should consider the label equivalence between l_2 and l_{13}. The condition that the label equivalence between l_2 and l_{13} should be recorded is that voxel v_9 is a background voxel. Because v_9 is a foreground voxel, nothing further needs to be done.

On the other hand, if we check the voxels in the mask from v_9, denoted as *Order 2*, we find that v_9 is a foreground voxel; then, we assign provisional label l_9 to $v(x, y, z)$. Because v_9 connects with all voxels in the mask as well as $v(x, y, z)$, all label equivalences in the mask must have been resolved. Therefore, nothing further needs to be done.

As discussed earlier, by Order 1, we should check nine voxels in the mask, but by Order 2, we need to check only one voxel. Therefore, for processing a foreground voxel, the order for checking voxels in the mask is a key factor for efficiency.

For 2-D binary images, He *et al*.[10] proposed an efficient checking order for the assigning--finding task by using the Karnaugh map to analyze the $2^4 = 16$ configurations in the mask. However, for 3-D binary images, because there are $2^{13} = 8192$ configurations in the mask, it is impossible to use the same method for finding the optimal processing order. On the other hand, there are 13! = 6227020800 possible orders for checking voxels in the mask. Therefore, it is also impossible to test all orders for finding the optimal one.

Because at any point in the first scan, all label equivalences between the provisional labels assigned to the processed foreground voxels have been recorded, when processing the current foreground voxel, for finding the new label equivalences in the mask caused by the occurrence of the current foreground voxel, we need to consider only the label equivalences between different connected parts in the mask. In other words, we need not consider the label equivalences between the provisional labels assigned to each connected part in the mask. For this reason, under the condition that the probability of each voxel in the mask being a foreground voxel is the same, checking the voxels in the mask in the order of the numbers of their neighbor voxels will be efficient because the greater the number of the neighbor voxels of a voxel in the mask, the less the number of the voxels that need to be checked.

The number of the neighbor voxels of each voxel in the mask is shown in Table I.

According to the earlier discussion and Table I, the order of checking voxels in the mask should be $v_9 \rightarrow v_3$, $v_6 \rightarrow v_1$, $v_8 \rightarrow v_{10} \rightarrow v_2$, v_5, $v_{12} \rightarrow v_4$, $v_7 \rightarrow v_{11} \rightarrow v_{13}$.

By the aforementioned order, for processing the current foreground voxel v, we first check voxel v_9. If v_9 is a foreground voxel, we assign l_9 to $v(x, y, z)$. Because v_9 connects with all other voxels in the mask, no new label equivalence needs to be recorded. The assigning--finding task can be terminated here.

Thus, when v_9 is a foreground voxel, the procedure for processing the current foreground voxel, denoted as *process* (v_9), is as follows:

$$l(x, y, z) \leftarrow l_9.$$

On the other hand, if v_9 is a background voxel, we check voxel v_3. If v_3 is a foreground voxel, we assign provisional label l_3 to $v(x, y, z)$, and then check whether there is any label equivalence between the provisional labels assigned to the connected part consisting of v_3 as well as its neighbor foreground voxels and other foreground voxels in the mask caused by the existence of the current foreground voxel. In other words, we do not need to check the neighbor voxels of v_3, i.e., v_1, v_2, v_4, v_5, v_6, v_7, v_8, and v_{10}, but we need to check voxels v_{11}, v_{12}, and v_{13}.

There is also an order problem with checking v_{11}, v_{12}, and v_{13} (a group of voxels). For exactly the same reason given earlier, we decide the order based on the number of the neighbor voxels of each voxel in the group. Because the number of the neighbor voxels of both v_{11} and v_{13} is 1 and that of v_{12} is two, we check v_{12} first, and then v_{11} and v_{13}. As

shown in Fig. 5(b), the condition that υ_{12} becomes connected with υ_3 only due to the existence of the current foreground voxel is the situation, where υ_8 and υ_{10} are background voxels. Thus, if both υ_8 and υ_{10} are background voxels, we should record the label equivalence between l_3 and l_{12}. If υ_{12} is a background voxel, we need to check whether there is a label equivalence between l_3 and l_{11}, as well as between l_3 and l_{13}, respectively. If υ_{11} is a background voxel, nothing needs to be done. Otherwise, i.e., if υ_{11} is a foreground voxel, as shown in Fig. 5(c), the condition for recording a label equivalence between l_3 and l_{11} is that υ_8 is a background voxel. In a similar way, if υ_{13} is a background voxel, nothing needs to be done. Otherwise, i.e., if l_{13} is a foreground voxel, as shown in Fig. 5(d), the condition for recording label equivalence between l_3 and l_{13} is that υ_{10} is a background voxel.

Based on the aforementioned discussion, when υ_9 is a background voxel and υ_3 is a foreground voxel, the processing procedure, denoted as *process*(υ_3), can be summarized as follows:

```
l(x, y, z) ← l3;
if (l12 > 0 & l8 = 0 & l10 = 0)
   resolve(l3, l12);
else
   if (l11 > 0 & l8 = 0)
      resolve(l3, l11);
   end of if
   if (l13 > 0 & l10 = 0)
      resolve(l3, l13);
   end of if
end of if
```

where l_i is 0 if υ_i is a background voxel.

Notice that, by *process*(υ_3), all label equivalences in the mask will be found and resolved; therefore, after *process*(υ_3), the assigning--finding--resolving task can be terminated.

On the other hand, if υ_3 is also a background voxel, we check voxel υ_6 and make an analysis similar to that described above. For each υ_i being checked, if it is a foreground voxel, the processing procedure *process*(υ_i) can be derived in a similar way.

Lastly, if all voxels in the checking order are background voxels, this means that the current foreground voxel does not connect with any foreground voxel processed up to now, i.e., the current foreground voxel belongs to a new connected component consisting of itself only. The current foreground voxel is assigned a new provisional label, say, *NewLabel*, which is initialized by 0 at the beginning of the first scan. The equivalent-label set for the new connected component is established by $S(NewLabel) \leftarrow \{NewLabel\}$, and the representative label of *NewLabel* is set to itself, i.e., *r_label*[*NewLabel*] = *NewLabel*. Then, *NewLabel* increases by 1 for consecutive processing. The process, denoted to be *process*(0), can be summarized as follows:

$$NewLabel \leftarrow NewLabel+1;$$
$$l(x,y,z) \leftarrow NewLabel;$$
$$(NewLabel) \leftarrow \{NewLabel\};$$
$$r\_label[NewLabel] \leftarrow NewLabel.$$

When the first scan is finished, all label equivalences have been resolved, and all equivalent labels will have a unique representative label. Thus, similar to the label-equivalence-based labeling algorithms for 2-D binary images, by setting *r_label*[0] = 0 in advance, replacing each label with its representative label in our algorithm, denoted as *label-replacing*, can be completed in the second scan as follows:

```
r_label[0] ← 0;
for (u = 2; u < U; u ← u + 1)
   for (υ = 2; υ < V; υ ← υ + 1)
      for (w = 2; w < W; w ← w + 1)
         l(u, υ, w) ← r_label[l(u, υ, w)];
      end of for
   end of for
   end of for
```

In the premise that the probability of each voxel in the mask to be a foreground voxel is the same, the proposed order for checking voxels in the mask is optimal.

For any case such that υ_9 in the mask shown in Fig. 3 is a foreground voxel, checking υ_9 first as in our method makes us only need to check one voxel (υ_9 itself), while checking any other voxel first will make us check at least two voxels.

If υ_9 is a background voxel, our method will then check υ_6. We show that when υ_6 and some other voxels in the mask are foreground voxels simultaneously, checking υ_6 first will be more efficient than checking any of those voxels.

For example, we consider the case, where υ_6 and υ_8 are foreground voxels. Our method checks υ_6 first, then goes to check υ_{12}. If υ_{12} is a foreground voxel, it terminates. Here, the number of times for checking voxels is 2. Otherwise, i.e., υ_{12} is a background voxel, it will go to check υ_{11} and υ_{13}. The number of times for checking voxels is 4. Thus, the average number of times for checking voxels is (2 + 4)/2 = 3.

On the other hand, if we check υ_8 first, because υ_8 is a neighbor voxel of υ_1, υ_2, υ_3, υ_5, υ_6, υ_{11}, and υ_{12}, we need not check any of them. The remains of voxels for checking are υ_4, υ_7, υ_{10}, and υ_{13}. It is not difficult to find that among them; checking υ_{10} first will be most efficient. If υ_{10} is a foreground voxel, we terminate there. The number of times for checking voxels is 2. Otherwise, we should check υ_{13} and υ_7. The number of times for checking voxels is 4. Moreover, if υ_7 is a background voxel, we should further check υ_4. In this case, the number of times for checking voxels is 5. Thus, the average number of times for checking voxels is at least (2 + (4 + 5)/2)/2 = 3.25.

Therefore, in the case, where v_6 and v_8 are foreground voxels, checking v_6 first will be more efficient than checking v_8 first.

Other cases can be shown in a similar way. Thus, we showed that our proposed order for checking voxels in the mask is optimal.

B. Proposed Run- and Label-Equivalence-Based Labeling Algorithms

We extend and improve the run- and label-equivalence-based two-scan labeling algorithms for 2-D binary images proposed in [9] to label 3-D binary images.

For convenience, we use $r(s, e, y, z)$ to denote a run starting from $v(s, y, z)$ and ending at $v(e, y, z)$. Moreover, we use $row(u, v)$ to denote the row with y-coordinate u and z-coordinate v. For the current run $r(s, e, y, z)$, its processed 26-connected range is the area from $v(s - 1, y - 1, z)$ to $v(e + 1, y - 1, z)$, $v(s - 1, y - 1, z - 1)$ to $v(e + 1, y - 1, z - 1)$, $v(s - 1, y, z - 1)$ to $v(e + 1, y, z - 1)$, and $v(s - 1, y + 1, z - 1)$ to $v(e + 1, y + 1, z - 1)$, as shown in Fig. 6. A processed run $r(u, v, m, n)$ is a *26-connected run* of the current run if it contains at least a foreground voxel in the processed 26-connected range of the current run.

For the current run, if there is no processed 26-connected run, we assign a new provisional label to it. Otherwise, i.e., if there are some 26-connected runs, $r_1, r_2, \ldots, r_n$. Let $l_1, l_2, \ldots, l_n$ be the provisional labels corresponding to $r_1, r_2, \ldots, r_n$, respectively. Then, $r_1, r_2, \ldots, r_n$ and the current run belong to the same connected component; thus, $l_1, l_2, \ldots, l_n$ are equivalent labels. We assign l_1 to the current run, and we then resolve the label equivalences between l_1 and $l_2, \ldots, l_n$, respectively.

Similar to the proposed voxel-based algorithm introduced earlier, recording and resolving label equivalences can be done in exactly the same way as in the algorithms proposed in [9] and [10].

Because all foreground voxels of a run belong to the same connected component, by labeling, they should be assigned the same label finally. Therefore, instead of assigning a provisional label to each voxel, as in the previous algorithm proposed in [9], we can assign a provisional label to each run. After all label equivalences between runs are resolved, all runs belonging to a connected component will have the same representative label. Then, by use of the recorded run data, we can assign to each foreground voxel in a run the representative label corresponding directly to the run. Thus, we improved the previous algorithm proposed in [9] in two ways: 1) the previous algorithm assigns a provisional label to each foreground pixel, while we assign a provisional label to each run. For images with large average length of runs, our algorithm will be very efficient. 2) In the second scan, the previous algorithm scans all pixels for relabeling foreground pixels, whereas our algorithm scans only foreground voxels (i.e., without scanning any background voxel). Thus, for images with low densities, our algorithm will be very efficient.

IV. Complexity of Our Algorithms

To complete labeling, our voxel-based algorithm performs the following procedures:

1. assigning provisional labels to foreground voxels during the first scan;
2. creating equivalent-label sets and setting representative labels for all new provisional labels;
3. resolving label equivalences;
4. replacing the provisional labels of foreground voxels with their representative labels during the second scan.

For a $U \times V \times W$-voxel image, both the maximum number of provisional labels and the maximum number of connected components have the order of $O(U \times V \times W)$. Accordingly, the order of the maximum number of label equivalences among provisional labels is also $O(U \times V \times W)$. By our voxel-based algorithm, procedures 1 and 4 are proportional to the number of foreground voxels, and procedure 2 is proportional to the number of provisional labels, either has the order $O(U \times V \times W)$.

For procedure 3, when resolving a label equivalence between two provisional labels, we need to combine an equivalent-label set, say, $S(u_1) = \{u_1,\ldots, u_m\}$, into another equivalent-label set, say, $S(\upsilon_1) = \{\upsilon_1,\ldots, \upsilon_n\}$, where $S(i)$ denotes the equivalent-label set with i as its representative label and $u_1 > \upsilon_1$. To realize this, for each $x \in \{u_1,\ldots, u_m\}$, we need to set its representative label to υ_1. The order of the operation is O (m).

For a Q-voxel connected component with R provisional labels, we consider the following two special cases: 1) when the maximum time of the operation happens; and 2) when the connected component has the maximum number of provisional labels, i.e., when R is a maximum.

In case 1, the maximum time of the operation should be $1 + 2 + 3 + \cdots + (R-1)$, and the order is $O(R^2)$. A typical connected component with five provisional labels for this case is shown in Fig. 10(a). In such cases, the number of voxels of a connected component has the order $O(R^2)$. Thus, $O(Q) = O(R^2)$.

In case 2, the maximum number of provisional labels for a Q-voxel connected component has the order $O(Q)$. A typical nine-voxel connected component for this case is shown in Fig. 7(b). In such cases, the time of the operation should be $1 + 1 + \cdots + 1 = (Q - 1)/2$, the order of which is $O(Q)$.

In both cases 1 and 2, the order of the operation for a Q-voxel connected component is $O(Q)$. Therefore, the order of the operation for labeling a $U \times V \times W$ image should be $O(U \times V \times W)$.

Because the order of every procedure for labeling a $U \times V \times W$ image is $O(U \times V \times W)$, the order of our voxel-based algorithm is $O(U \times V \times W)$.

Our run-based algorithm also consists of four procedures:

1. finding runs, recording run data, and assigning provisional labels to runs during the first scan;

2. creating equivalent-label sets and setting representative labels for all new provisional labels;
3. resolving label equivalences;
4. assigning each foreground voxel the representative label of the run containing the voxel.

For a $U \times V \times W$-voxel image, the number of runs will be smaller than or equal to the number of foreground voxels, and the number of provisional labels assigned by our run-based algorithm will be smaller than or equal to that assigned by our voxel-based algorithm; thus, the number of label equivalences in the case when we use our run-based algorithm will be smaller than or equal to that when we use our voxel-based algorithm.

Accordingly, each of the procedures 1--4 has the order $O(U \times V \times W)$. Thus, the order of our run-based algorithm should also be $O(U \times V \times W)$.

V. Experimental Results

Because recording and resolving label equivalences in our proposed two algorithms can be done in exactly the same way for those in the algorithm proposed in [9] and that proposed in [10], our voxel-based algorithm, denoted as *Ours-1* algorithm, can easily be implemented in a similar way, as is done for the two algorithms mentioned. Because the maximum provisional labels for a $U \times V \times W$-size 3-D binary image is $U \times V \times W/27$, the data structure for equivalent-label sets and the representative label table in our algorithm can be realized by use of three $U \times V \times W/27$-size arrays, i.e., *r_label*[ ], *next_label*[ ], and *last_label*[ ], where *next_label*[*i*] indicates the next label of *i* in the equivalent-label set *S*(*r_label*[*i*]), and *last_label*[*j*] means the last label in the equivalent-label set *S*(*j*). Moreover, *next_label*[*l*] = −1 means that there is no next label after label *l*, i.e., *l* is the last label in the corresponding equivalent-label set.

Thus, when a new label, *NewLabel*, is assigned to a foreground voxel, a new equivalent-label set *S*(*NewLabel*) = {*NewLabel*} is established, and the corresponding data in the data structure are set as follows:

$$r\_label[NewLabel] \leftarrow NewLabel;$$
$$next\_label[NewLabel] \leftarrow -1;$$
$$last\_label[NewLabel] \leftarrow NewLabel;$$

When an equivalent-label set *S*(*i*) is combined into another equivalent-label set *S*(*j*), where $i < j$, the corresponding data in the data structure, changes as follows:

```
m ← i;
while(m ≠ −1)
   r_label[m] ← j;
   m ← next_label[m];
end of while
next_label[last_label[j]] ← i;
```

last_label[*j*] ← *last_label*[*i*];

For implementing our proposed run-based algorithm, denoted as *Ours-2* algorithm, except for the three $U \times V \times W/27$-size arrays for *r_label*[ ], *next_label*[ ], and *last_label*[ ], respectively, we also need three $U \times V \times W/2$-size arrays for recording the starting points, end points, provisional labels of runs,[2] and a $U \times V/2$-size array for recording the number of runs.

Hu *et al*. proposed two iterative-recursion-based 3-D labeling methods [11]: one uses iterative recursion to label all foreground voxels directly, and the other uses iterative recursion to label the boundary foreground voxels and a one-pass process to label nonboundary foreground voxels, which is an extension of the contour-tracing 2-D labeling algorithm proposed in [4].

Although, for labeling connected components, recursion takes less time, it consumes more memory. Memory overflow often takes place when a recursion-based algorithm is used for labeling a 3-D binary image. Iteration is introduced to avoid memory overflow for large images. Iteration and recursion are combined as follows: Marking an unlabeled foreground voxel found by the raster scan as *selected* by changing the voxel's label from 1 to 2; for each *selected* foreground voxel υ, assigning to υ a label *l* (which is initialized to be 3); then iteration is executed for assigning label *l* to the foreground voxels connected with υ within a local cuboid with υ in the center; each foreground voxel on the border of the local cuboid is made as *selected*, and subsequent iterations on the *selected* foreground voxels are recursively called. After all *selected* foreground voxels are processed, *l* increases by 1 for consecutive processing, and then the raster scan continues to find the next unlabeled foreground voxel (if any), which is processed in the same way, and so on.

According to the experimental results shown in [11], the larger the size of a local cuboid, the lesser the memory consumed, but more execution time is needed. Moreover, with a local cuboid of the same size, the second method runs faster and uses less memory than does the first one. Therefore, we will only consider the second method in this section. For convenience, instead of using a local cuboid, we will use a local cube. We denote the algorithm with a size $i \times i \times i$ of the local cube as CTL_i. Thus, the CTL_0 is the fastest one. Because the experimental results also showed that the two algorithms were much more efficient than other conventional 3-D labeling algorithms, we will only compare our algorithms with the CTL_0 in this section.

All three algorithms were implemented with the *C* language on a PC-based workstation (Intel Pentium Duo 930 3.0 GHz + 3.0 GHz CPUs, 2-GB Memory, Mandriva Linux OS), and compiled by the GNU C complier (version 4.2.3) with the option −O3. All execution times shown in this section were obtained by use of one core.

We first compared the algorithms with uniform noise images. Five sets of 41 uniform noise images with five different sizes (128 × 128 × 128, 327 × 327 × 327, 408 × 408 × 408, 465 × 465 × 465, and 512 × 512 × 512 voxels) were generated by thresholding of the images

[2]The maximum number of runs in an $U \times V \times W$-size image is $U \times V \times W/2$.

containing uniform random noise ranging from 0 to 1000, with 41 different threshold values from 0 to 1000 in steps of 25. Because connected components in such noise images have complicated geometrical shapes and complex connectivity, severe evaluations of labeling algorithms can be performed with these images.

We used all noise images for testing the linearity of the computation of the algorithms. For each size of the noise images, the maximum and average execution times are shown in Fig. 8(a) and (b), respectively. As we can see from Fig. 8, all three algorithms have the ideal linear characteristic against image sizes.

512 × 512 × 512-sized noise images were used for testing the execution time versus the density of images. The results are shown in Fig. 9.

We also used 512×512×512 noise images to compare voxel-based algorithm with two natural extensions of the algorithm proposed in [10], denoted as *Ours*-1* *algorithm* and *Ours*-1** *algorithm*, respectively, and our run-based algorithm with the natural extension of the one proposed in [9], denoted as *Ours*-2* *algorithm*.

The Ours-1* algorithm checks voxels in the mask with a randomly selected order $v_7 \rightarrow v_3 \rightarrow v_{11} \rightarrow v_9 \rightarrow v_2 \rightarrow v_6 \rightarrow v_{13} \rightarrow v_{10} \rightarrow v_4 \rightarrow v_1 \rightarrow v_8 \rightarrow v_5 \rightarrow v_{12}$, the Ours-1** algorithm does with another randomly selected order $v_{12} \rightarrow v_1 \rightarrow v_8 \rightarrow v_5 \rightarrow v_{13} \rightarrow v_7 \rightarrow v_{11} \rightarrow v_2 \rightarrow v_9 \rightarrow v_{10} \rightarrow v_4 \rightarrow v_6$.

The results are shown in Fig. 10.

From Fig. 10, we can find that, for all noise images, both of our proposed algorithms are more efficient than the algorithms naturally extended from the previous algorithms. On an average, for all images, the Ours-1 algorithm is 24.2% faster than the Ours-1* algorithm and 37.3% faster than the Ours-1** algorithm, and the Ours-2 algorithm is 14.0% faster than the Ours-2* algorithm. Especially, for the last ten high-density images, the Ours-1 algorithm is 39.8% faster than the Ours-1* algorithm and 91.0% faster than the Ours-1** algorithm, and for the first five and the last five images, Ours-2 algorithm is 37.3% faster than the Ours-2* algorithm.

The reason that the Ours-1 algorithm is much better than the Ours-1* algorithm and the Ours-1** algorithm for high-density images is that, for a high-density image, the possibility that the voxel v_9 is a foreground voxel will be large. In this case, the Ours-1 algorithm only need to check the voxel v_9, but the Ours-1* algorithm might need to check v_7, v_{11}, v_2, and v_{13}, and the Ours-1** algorithm might need to check v_{12}, v_1, v_8, v_5, v_{13}, v_7 etc, where the Ours-1** algorithm might check more voxels than does the Ours-1* algorithm.

On the other hand, for high-density images, the average length of runs in the image will be large, i.e., the number of runs will be much smaller than the number of foreground voxels. Because the Ours-2 algorithm assigns to each run a provisional label, whereas the Ours-2* algorithm assigns to each foreground voxel a provisional label, the number of operations for assigning provisional labels by the Ours-2 algorithm will be much smaller than that by the Ours-2* algorithm. For low-density images, the number of foreground voxels will be small.

Because, in the second scan, the Ours-2 algorithm only scans foreground voxels, whereas the Ours-2* algorithm scans all voxels, the number of operations for scanning voxels in the second scan by the Ours-2 algorithm will be much smaller than that by the Ours-2* algorithm. Therefore, the Ours-2 algorithm is much better than the Ours-2* algorithm for high-density and low-density images.

Second, we compared the algorithms on overlapped-cube image set. This set is composed of images with a random distribution of 50 square cubes of foreground voxels, where overlap of cubes is allowed, with cube size ranging from 5 × 5 × 5 to 100 × 100 × 100 in steps of 5, 10 000 different images for each cube size. The densities of these images range from 0.46% to 80.01%.

The overlapped-cube images were used for testing the execution time versus the cube size. The results are shown in Fig. 11, where the running time for each cube size is the average of the running times on the 10 000 different images corresponding to that size. Notice that there are two vertical axes in Fig. 11: the left one is for the Ours-1 algorithm and the Ours-2 algorithm, and the right one for the CTL_0 algorithm.

Moreover, similarly, as in [11], a 3-D magnetic resonance (MR) image was downloaded from the Montreal Neurological Institute website (http://www.bic.mni.mcgill.ca/brainweb) with a noise level of 5%, and an intensity inhomogeneity level of 20%. The size of the image was 181 × 217 × 181, and the gray-level range from 0 to 255. Twenty-four binary images were derived by thresholding the image with a threshold changing from 10 to 120 in steps of 10. A sagittal slice of the image and its binary image with a threshold of 45 is shown in Fig. 12(a) and (b), respectively. The testing results are shown in Table II.

Lastly, a 256 × 256 × 552 size 3-D binary image of abdominal CT was used for testing. An axial slice of the 3-D binary image is shown in Fig. 13. The running times of the Ours-1 algorithm, the Ours-2 algorithm, and the Hu-algorithm were 0.344, 0.571, and 2.893 s, respectively.

VI. Comparison of Ours-1 Algorithm, Ours-2 Algorithm, and CTL_0 Algorithm

The original *CTL* algorithm proposed in [4] for the labeling of 2-D binary images is a one-scan algorithm. However, the CTL_0 algorithm for 3-D binary images is a two-scan algorithm.

1. In the first scan, the CTL_0 algorithm differentiates contour voxels from other foreground voxels (i.e., inner foreground voxels) by setting each contour voxel to 2.
2. In the second scan, for each unlabeled foreground voxel $\upsilon(x, y, z)$, if $\upsilon(x, y, z) = 2$ and $label(x - 1, y, z) = 0$, i.e., $\upsilon(x, y, z)$ is an outside contour voxel, it assigns $\upsilon(x, y, z)$ a new label, i.e., $label(x, y, z) \leftarrow NewLabel$, where *NewLabel* is initialized to be 3, and then propagates the label to all outside contour voxels that are connected to $\upsilon(x, y, z)$ by use of recursive operation; else, if $\upsilon(x, y, z) = 2$ and $label(x - 1, y, z) > 2$, i.e., $\upsilon(x, y, z)$ is an inside contour voxel, it assigns to $\upsilon(x, y, z)$ the label of $label(x - 1, y, z)$, i.e., $label(x, y, z) \leftarrow label(x - 1, y, z)$, and then propagates the label to all

inside contour voxels that are connected to $v(x, y, z)$ by using the recursive operation; else, if $v(x, y, z) = 1$, i.e., $v(x, y, z)$ is an inner foreground voxel, it merely assigns to $v(x, y, z)$ the label of $label(x - 1, y, z)$, i.e., $label(x, y, z) \leftarrow label(x - 1, y, z)$.

Because all labeling algorithms do nothing for background voxels, the performance of an algorithm will depend on the methods for processing foreground voxels.

For Ours-1 algorithm, for a foreground voxel with all of its neighbor voxels being background voxels, it checks 13 voxels in the mask. For a foreground voxel with all of its neighbour voxels being foreground voxels, it checks only one voxel (v_9 in the mask shown in Fig. 3). In both cases, it does not need to resolve any label equivalence. For the other cases, the number of voxels checked by Ours-1 algorithm will vary between 2 and 12, and some label equivalences will need to be resolve, depending on the configuration of its neighbor foreground voxels. Therefore, the efficiency of the Ours-1 algorithm will depend on the complexity of connected components in images.

For Ours-2 algorithm, for any run, it needs to check the four row in the corresponding 26-connected range. Because it accesses background voxels only once, it will be efficient for low-density images. Moreover, because it assigns provisional labels to runs, it will be also efficient for images with large average lengths of runs (in such cases, the number of runs will be much smaller than that of foreground voxels). On the other hand, because Ours-2 algorithm needs to record the run data, for images with small average lengths of runs (in such cases, the number of runs is near to that of foreground voxels), it will take much time than Ours-1 algorithm for labeling.

On the other hand, for any foreground voxel, the CTL_0 algorithm needs to check all 26 neighbor voxels. Therefore, the total number of times for it to check neighbor voxels will increase with the density of an image.

The aforementioned analyses are consistent with the experimental results given in Section V. From Fig. 9, we can find that the execution times of the CTL_0 algorithm are almost proportional to the density of an image, whereas the execution times of the Ours-1 algorithm depend on the complexity of the connected components in an image.[3] On the other hand, the Ours-2 algorithm is very efficient for low-density images (where the number of background voxels is large) and high-density images (where the average length of runs is large). However, for images with complex connected components, it takes much more time for labeling than the Ours-1 algorithm.

From Fig. 11, for all overlapped-cube images, we can find that both the Ours-1 and the Ours-2 algorithms are much more efficient than the CTL_0 algorithm. Moreover, the Ours-2 algorithm is more efficient than the Ours-1 algorithm, where the average length of runs is larger than or equal to 5, especially when the length of the sides of cubes becomes large.

[3]For noise images, with the increase of the density of an image from 0 to 0.5, the complexity of connected components also increases, and when the density of an image exceeds 0.5, the complexity of connected components decreases with the increase in density, i.e., the maximum complexity of connected components occurs in images with densities around 0.5.

For the 3-D magnetic resonance MR head images, from Table II, the execution time of the CTL_0 algorithm is positively correlated with the density of an image, whereas the execution times of the Ours-1 and Ours-2 algorithms vary only little for different densities. Moreover, because, in such images, when the average length of the runs is small, the density is small, and when the density is large, the average length of the runs is large; the Ours-2 algorithm is more efficient than the Ours-1 algorithm.

Another main problem of the CTL_0 algorithm is the overflow of stack [4]. Although the CTL_0 algorithm can reduce the requirement on stack memory by increasing the local cube size, this will decrease its efficiency. In other words, there is a tradeoff between execution time and memory. On the other hand, the memory space necessary for the Ours-1 and Ours-2 algorithms is $U \times V \times W/9$ and $29 \times U \times V \times W/18$, respectively.

VII. Concluding Remarks

In this paper, we proposed two label-equivalence-based labeling algorithms for 3-D binary images. One is a voxel-based algorithm, which checks the neighbors of the current voxel in optical order and is efficient for images with complicated connected components. The other is a run-based algorithm, which assigns provisional labels to runs rather than voxels, as in conventional label-equivalence-based labeling algorithms; it processes background voxels only once, and it is efficient for images with low density or a large average length of runs. Both of the proposed algorithms are linearity to image sizes, and much more efficient than those naturally extended from the related algorithms for 2-D binary images. Experimental results demonstrated that the proposed two algorithms were much more efficient than conventional labeling algorithms for 3-D binary images.

Even with our proposed algorithms, it takes seconds for labeling $512 \times 512 \times 512$ 3D binary images. Therefore, the hardware implementations [26] and/or parallel implementations [1], [13] of our algorithms should be considered. For example, we can divide a large 3-D binary image into some small subimages, and process all subimages in parallel; then, we combine the results for all subimages together by resolving the label equivalences on their interfaces.

Acknowledgments

The authors would like to thank the anonymous referees for their valuable comments that improved this paper greatly. They are grateful to Prof. S. Acton, the Editor, for his kind cooperation and help, and also to E. F. Lanzl for proofreading this paper.

This work was supported by the Kayamori Foundation of Informational Science Advancement, Japan, by the Hibi Research Grant, Japan, by the Aritficial Intelligence Research Promotion Foundation, Japan, and by the National Institutes of Health, Bethesda, MD, under Grant R01CA120549, Grant S10 RR021039, and Grant P30 CA14599. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. S. T. Acton.

References

1. Alnuweiri HM, Prasanna VK. Parallel architectures and algorithms for image component labeling. IEEE Trans Pattern Anal Mach Intell. Oct; 1992 14(10):1014–1034.
2. Ballard, DH. Computer Vision. Englewood Cliff, NJ: Prentice-Hall; 1982.

3. Borgefors, G.; Nystrom, I.; Baja, GSD. Connected components in 3D neighbourhoods. Proc 10th Scand Conf Image Anal; 1997. p. 567-572.
4. Chang F, Chen CJ, Lu CJ. A linear-time component-labeling algorithm using contour tracing technique. Comput Vis Image Understand. 2004; 93:206–220.
5. Crum WR, Camara O, Hill DLG. Generalized overlap measures for evaluation and validation in medical image analysis. IEEE Trans Med Imag. Nov; 2006 25(11):1451–1461.
6. Finnis KW, Starreveld YP, Parrent AG, Sadikot AF, Peters TM. Three-dimensional database of subcortical electrophysiology for image-guided stereotactic functional neurosurgery. IEEE Trans Med Imag. Jan; 2003 22(1):93–104.
7. Gonzalez, RC.; Woods, RE. Digital Image Processing. Reading MA: Addison-Wesley; 1992.
8. Haralick, RM. Real Time/Parallel Computing Image Analysis. New York: Plenum; 1981. Some neighborhood operations; p. 11-35.
9. He L, Chao Y, Suzuki K. A run-based two-scan labeling algorithm. IEEE Trans Image Process. May; 2008 17(5):749–756. [PubMed: 18390379]
10. He L, Chao Y, Suzuki K, Wu K. Fast connected-component labeling. Pattern Recognit. 2009; 42(9):1977–1987.
11. Hu Q, Qian G, Nowinski WL. Fast connected-component labeling in three-dimensional binary images based on iterative recursion. Comput Vis Image Understand. 2005; 99:414–434.
12. Lumia R, Shapiro L, Zungia O. A new connected components algorithm for virtual memory computers. Comput Vis Graph Image Process. 1983; 22(2):287–300.
13. Manohar M, Ramapriyan HK. Connected component labeling of binary images on a mesh connected massively parallel processor. Comput Vis Graph Image Process. 1989; 45(2):133–149.
14. Mehlhorn, K. Data Structures and Algorithm 1: Sorting and Searching. Berlin, Germany: Springer-Verlag; 1984.
15. Queirolo C, Silva L, Bellon O, Segundo M. 3D face recognition using simulated annealing and the surface interpenetration measure. IEEE Trans Pattern Anal Mach Intell. Feb; 2010 32(2):206–219. [PubMed: 20075453]
16. Rosenfeld A, Pfalts JL. Sequential operations in digital picture processing. J ACM. Oct; 1966 13(4):471–494.
17. Rosenfeld A. Connectivity in digital pictures. J ACM. Jan; 1970 17(1):146–160.
18. Rosenhahn B, Brox T, Weickert J. Three-dimensional shape knowledge for joint image segmentation and pose tracking. Int J Comput Vis. 2007; 73(3):243–262.
19. Shirai, Y. Three-Dimensional Computer Vision. New York: Springer-Verlag; 1987. Labeling connected regions; p. 86-89.
20. Suzuki K, Horiba I, Sugie N. Linear-time connected-component labeling based on sequential local operations. Comput Vis Image Understand. 2003; 89:1–23.
21. Suzuki K, Yoshida H, Nappi J, Dachman AH. Massive-training artificial neural network (MTANN) for reduction of false positives in computer-aided detection of polyps: Suppression of rectal tubes. Med Phys. 2006; 33:3814–3824. [PubMed: 17089846]
22. Suzuki K, Yoshida H, Nappi J, Armato SG III, Dachman AH. Mixture of expert 3D massive-training ANNs for reduction of multiple types of false positives in CAD for detection of polyps in CT colonography. Med Phys. 2008; 35:694–703. [PubMed: 18383691]
23. Thurfjell L, Bengtsson E, Nordin B. A new three-dimensional connected components labeling algorithm with simultaneous object feature extraction capability. J CVGIP: Graph Models Image Process. 54(4):357–364.
24. Tarjan RE. Efficiency of a good but not linear set union algorithm. J ACM. 1975; 22(2):215–225.
25. Udupa J, Ajjanagadde VG. Boundary and object labelling in three-dimensional images. Comput Vis Graph Image Process. 1990; 51(3):355–369.
26. Yang, XD. Design of fast connected components hardware. Proc IEEE Conf Comput Vis Pattern Recognit; Ann Arbor, MI. Jun. 1988; p. 937-944.
27. Yoshida H, Nappi J. Three-dimensional computer-aided diagnosis scheme for detection of colonic polyps. IEEE Trans Med Imag. Dec; 2001 20(12):1261–1274.

28. Young, TY. Handbook of Pattern Recognition and Image Processing. Vol. 2. Orlando, FL: Academic; 1994.

Biographies

Lifeng He received the B.E. degree from Northwest Institute of Light Industry, Shaanxi, China, in 1982, the second B.E. degree from Xian Jiaotong University, Shaanxi, in 1986, and the M.S. and the Ph.D. degrees in artificial intelligence and computer science from Nagoya Institute of Technology, Aichi, Japan, in 1994 and 1997, respectively.

He is currently an Associate Professor at Aichi Prefectural University, Aichi, and a Guest Professor at Shaanxi University of Science and Technology, Shaanxi. From September 2006 to May 2007, he was a Research Associate at The University of Chicago. His research interests include intelligent image processing, computer vision, medical image processing, automated reasoning, and artificial intelligence.

Yuyan Chao received the B.E. degree from North-west Institute of Light Industry, Shaanxi, China, in 1984, and the M.S. and the Ph.D. degrees from Nagoya University, Nagoya, Japan, in 1997 and 2000, respectively.

From 2000 to 2002, she was a special foreign Researcher of Japan Society for the Promotion of Science at Nagoya Institute of Technology, Aichi, Japan. She is currently a Professor at Nagoya Sangyo University, Aichi, and a Guest Professor at Shaanxi University of Science and Technology, Shaanxi. Her research interests include image processing, graphic understanding, computer-aided design, and automated reasoning.

Kenji Suzuki received the B.S. (*magna cum laude*) and M.S. (*summa cum laude*) degrees in electrical and electronic engineering from Meijo University, Nagoya, Japan, in 1991 and 1993, respectively, and the Ph.D. degree (by published work) in information engineering from Nagoya University, Nagoya, Japan, in 2001.

From 1993 to 1997, he was a Researcher with the Research and Development Center, Hitachi Medical Corporation. From 1997 to 2001, he was a faculty member with the Faculty of Information Science and Technology, Aichi Prefectural University, Aichi, Japan. In 2001, he was a Research Associate with the Kurt Rossmann Laboratories for Radiologic Image Research, Department of Radiology, Division of the Biological Sciences, The University of Chicago, where he became a Research Associate (Instructor) in 2003, and Research Associate (Assistant Professor) in 2004, and since 2006, has been an Assistant Professor in the Department of Radiology, the Committee of Medical Physics, and the Cancer Research Center. He is the author or coauthor of more than 100 scientific papers (including 45 peer-reviewed journal papers) in the field of medical image analysis, machine learning, computer vision, and pattern recognition.

Dr. Suzuki has been a referee for more than 15 journals, including IEEE Transactions on Medical Imaging, IEEE Transactions on Biomedical Engineering, IEEE Transactions on Information Technology in Biomedicine, IEEE Transaction on Image Processing, IEEE Transaction on Signal Processing, IEEE Transaction on Systems, Man and Cybernetics, and *Image and Vision Computing*. He has received awards for his research, including the Paul C. Hodges Award from The University of Chicago, in 2002, a Certificate of Merit Award from the Radiological Society of North America (RSNA), in 2003, a Research Trainee Prize from the RSNA, in 2004, a Young Investigator Award from the Cancer Research Foundation, in 2005, Honorable Mention Poster Award at the SPIE International Symposium on Medical Imaging, in 2006, and a Certificate of Merit Award from RSNA, in 2006. He was elected as a Senior Member of the IEEE in 2004. He is a member of the Institute of Electronics, Information and Communication Engineers, Institute of Electrical Engineers of Japan, Information Processing Society of Japan, Japanese Neural Network Society, and Japanese Circulation Society.

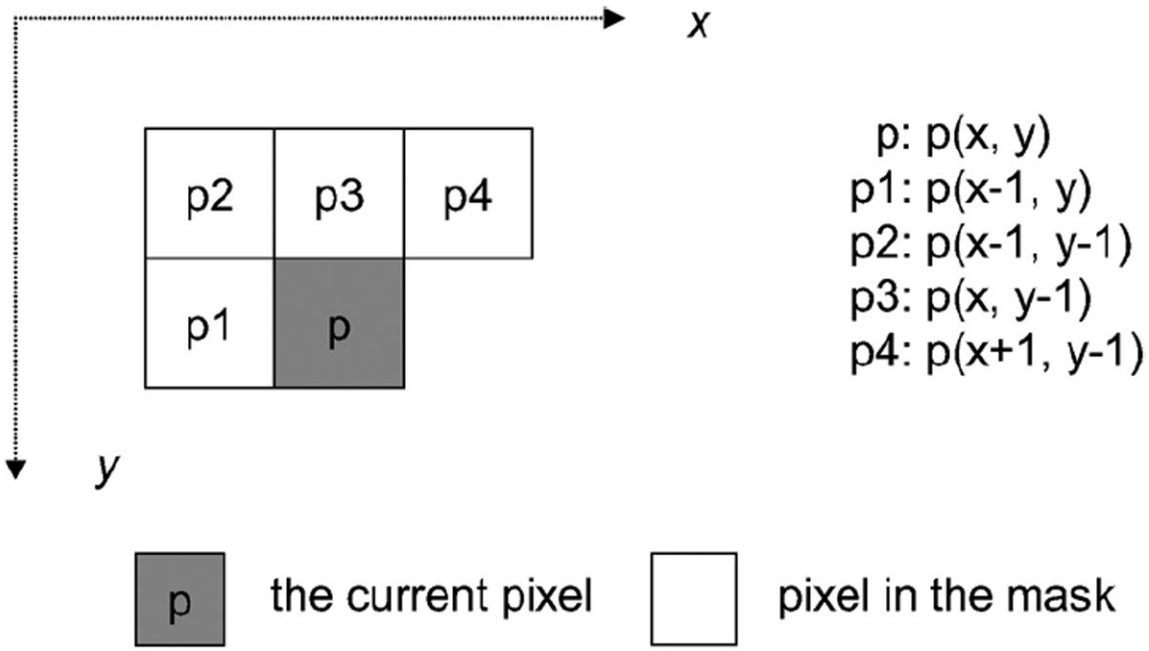


Fig. 1.
Mask for labeling 2-D binary images with 8-connectivity.

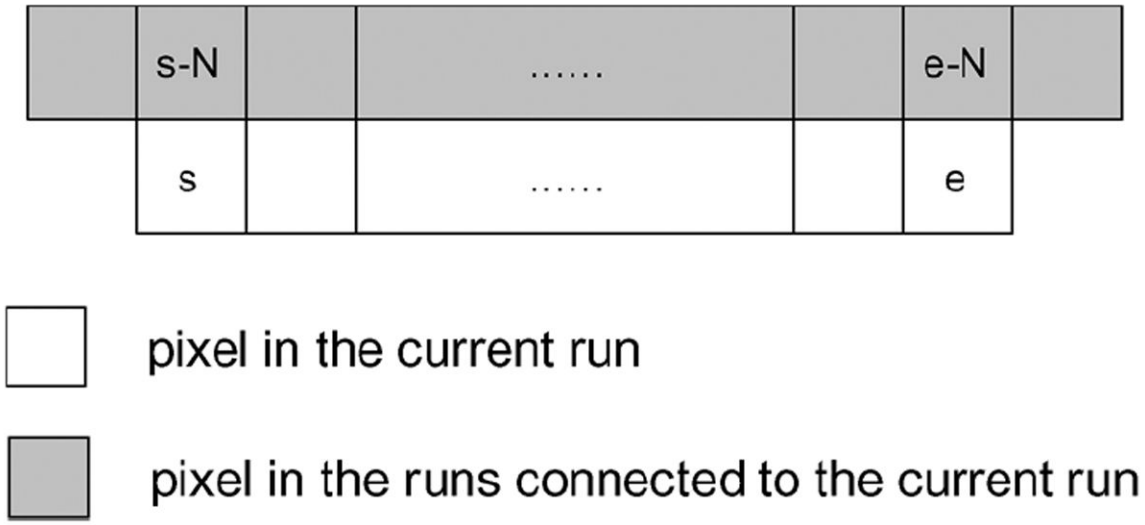


Fig. 2.
Range for checking the processed eight-connected runs of the current run $r(s, e)$.

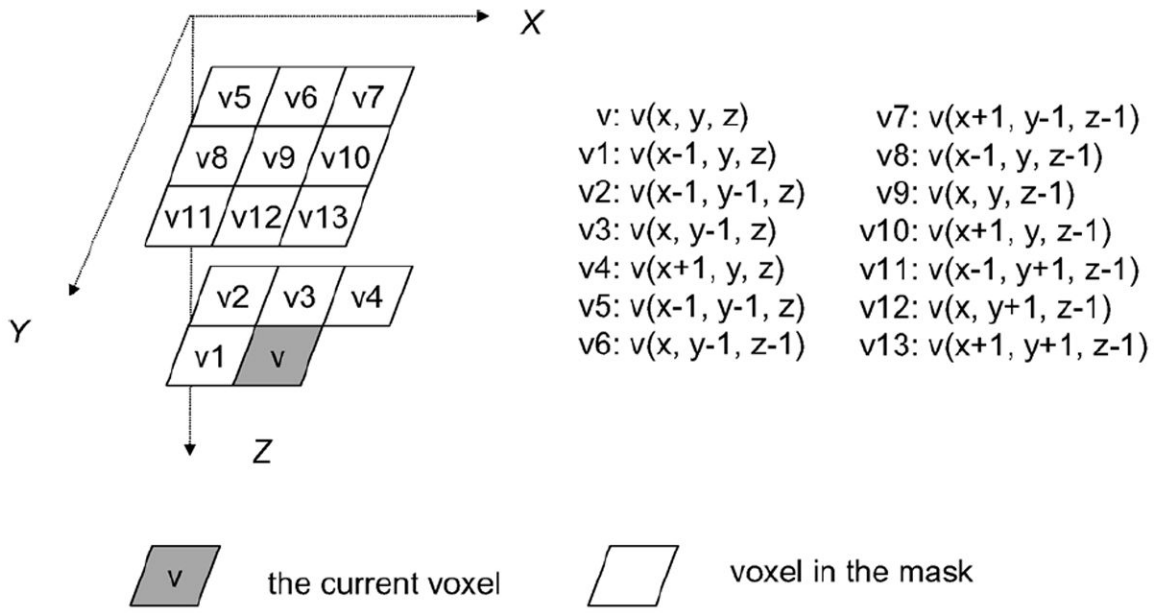


Fig. 3.
Mask with 26-connectivity for 3-D binary images.

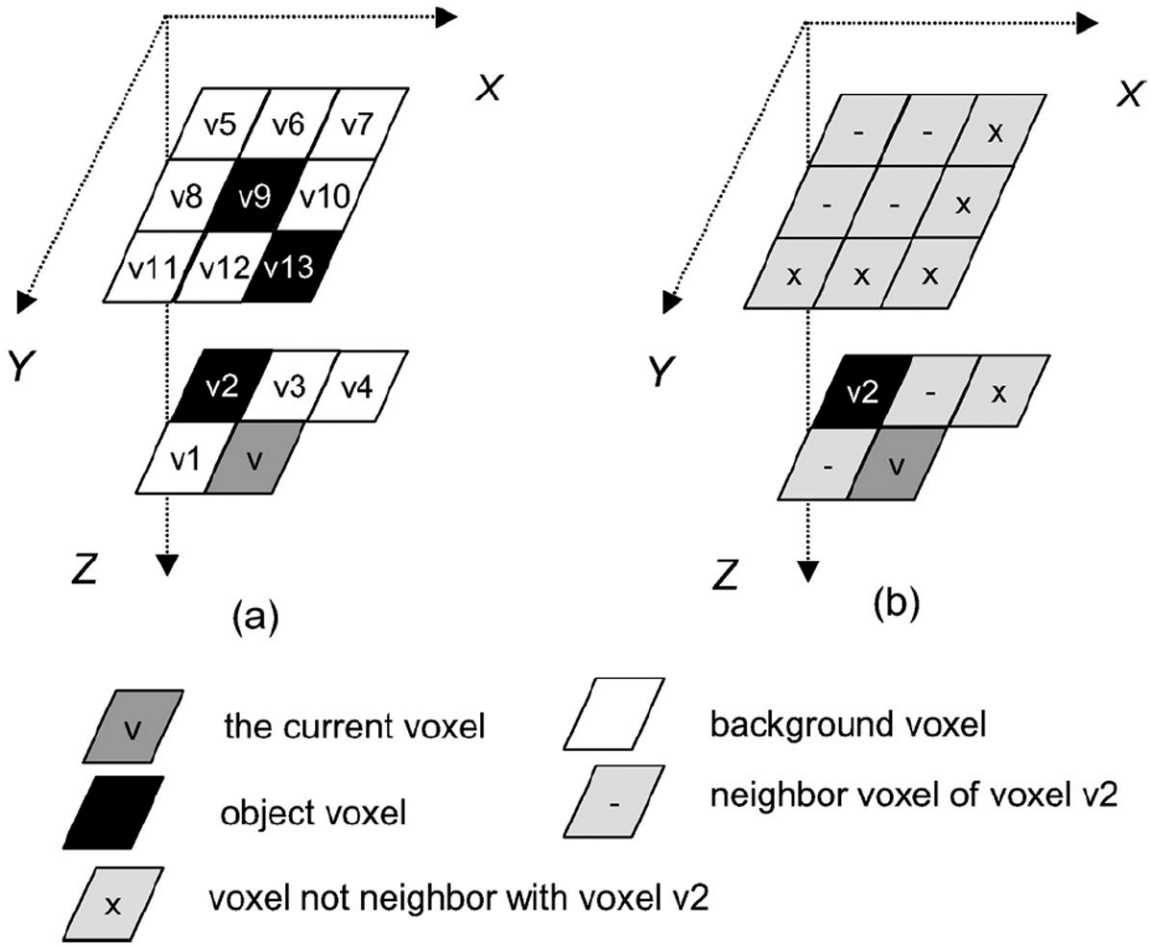


Fig. 4.
Configuration in the mask for a foreground voxel.

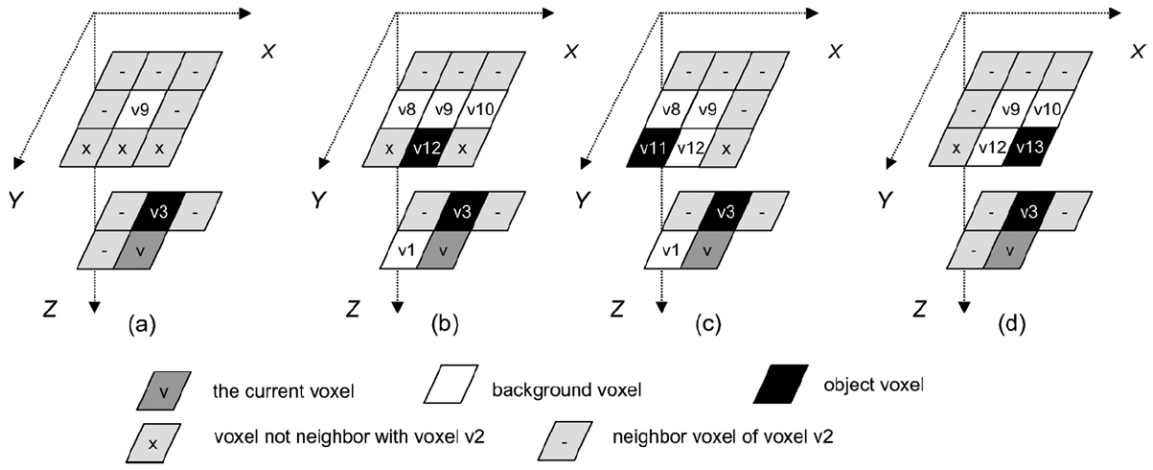


Fig. 5.
Status where v_3 is a foreground voxel.

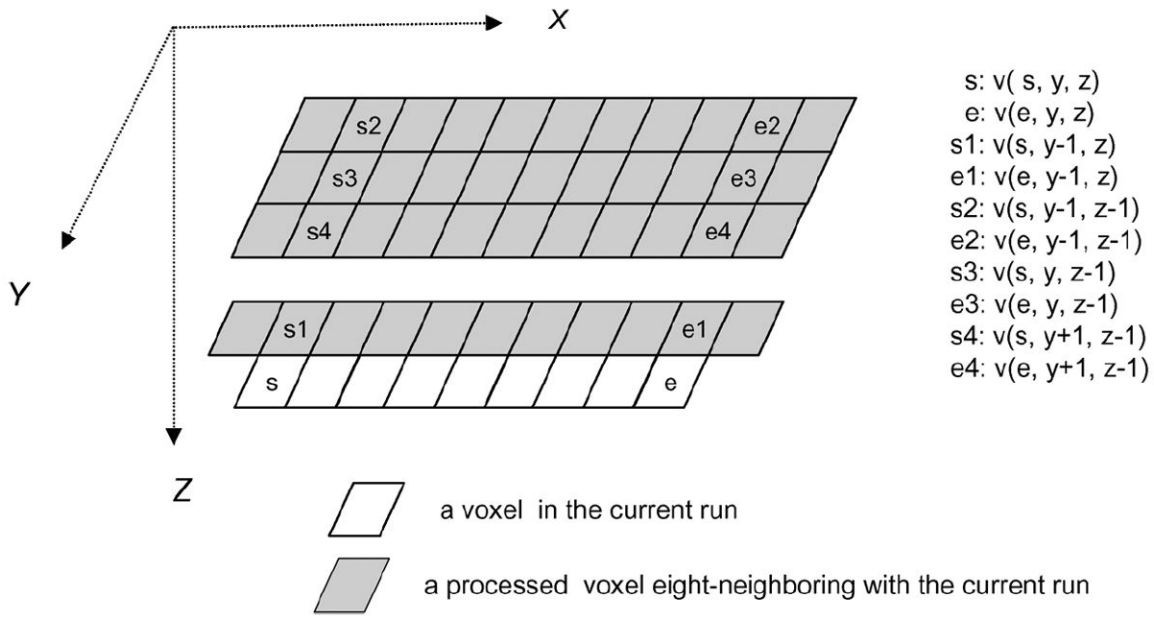


Fig. 6.
Processed 26-connected range of the current run $r(s, e, y, z)$.

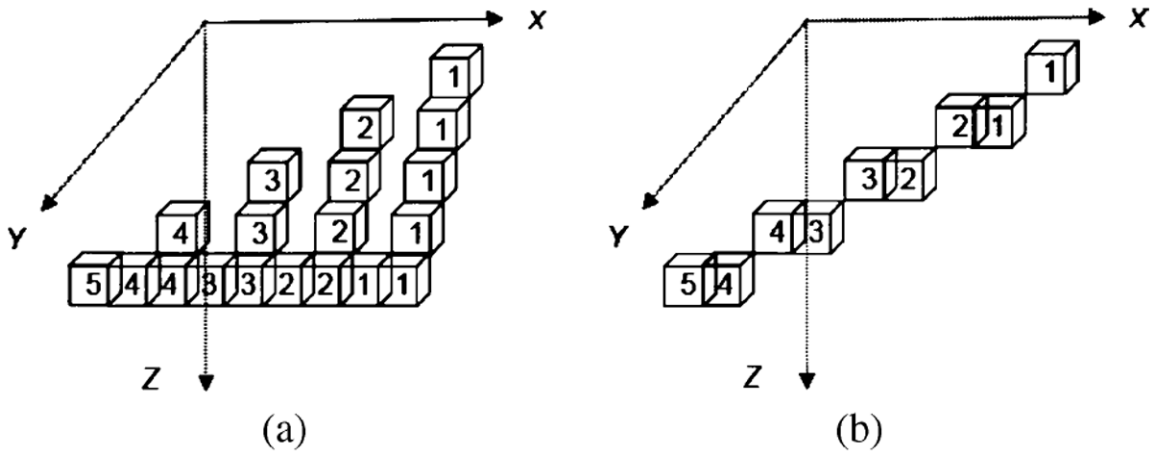


Fig. 7.
Two typical connected components.

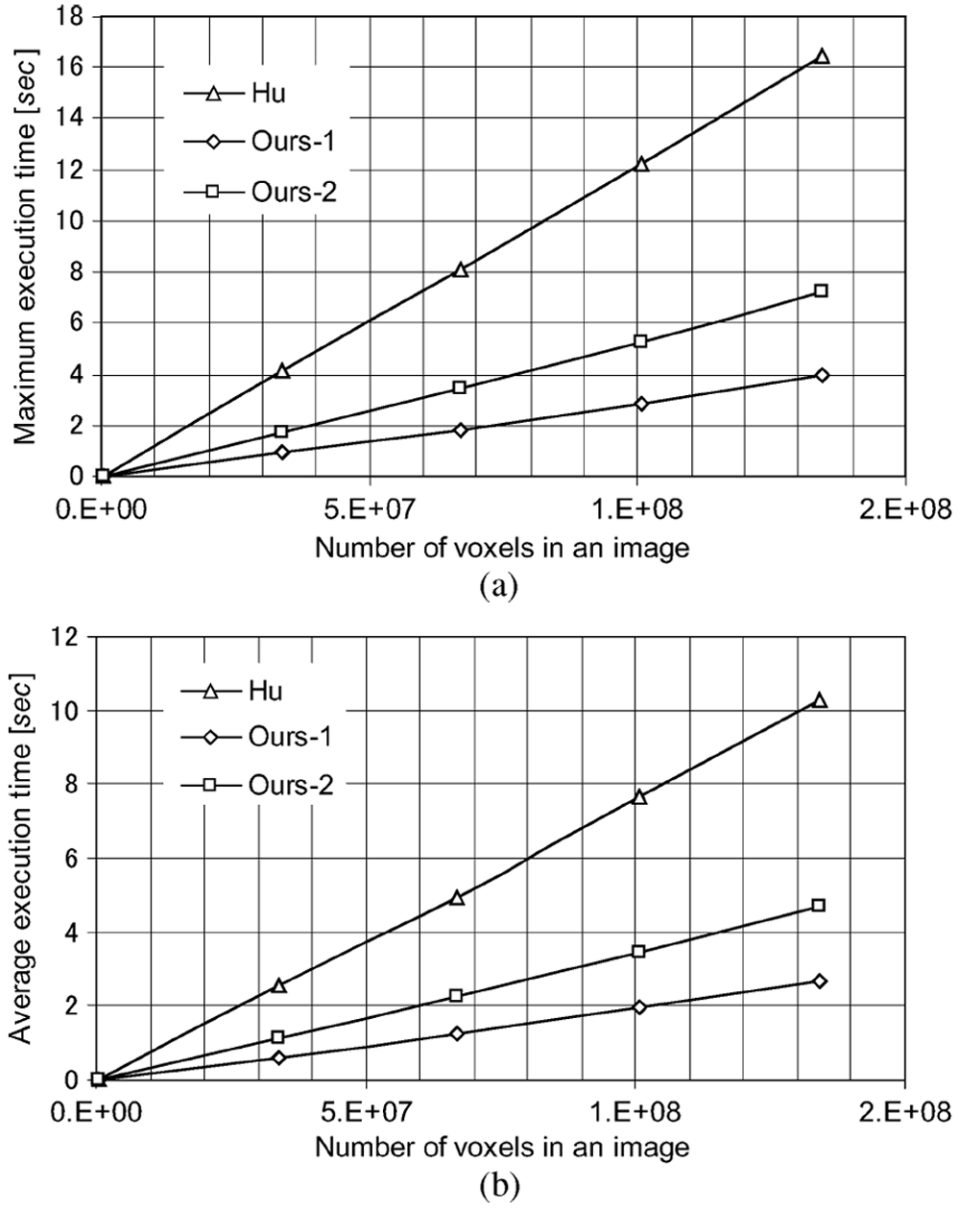


Fig. 8.

Execution time on different-sized noise images: (a) Maximum execution time. (b) Average execution time.

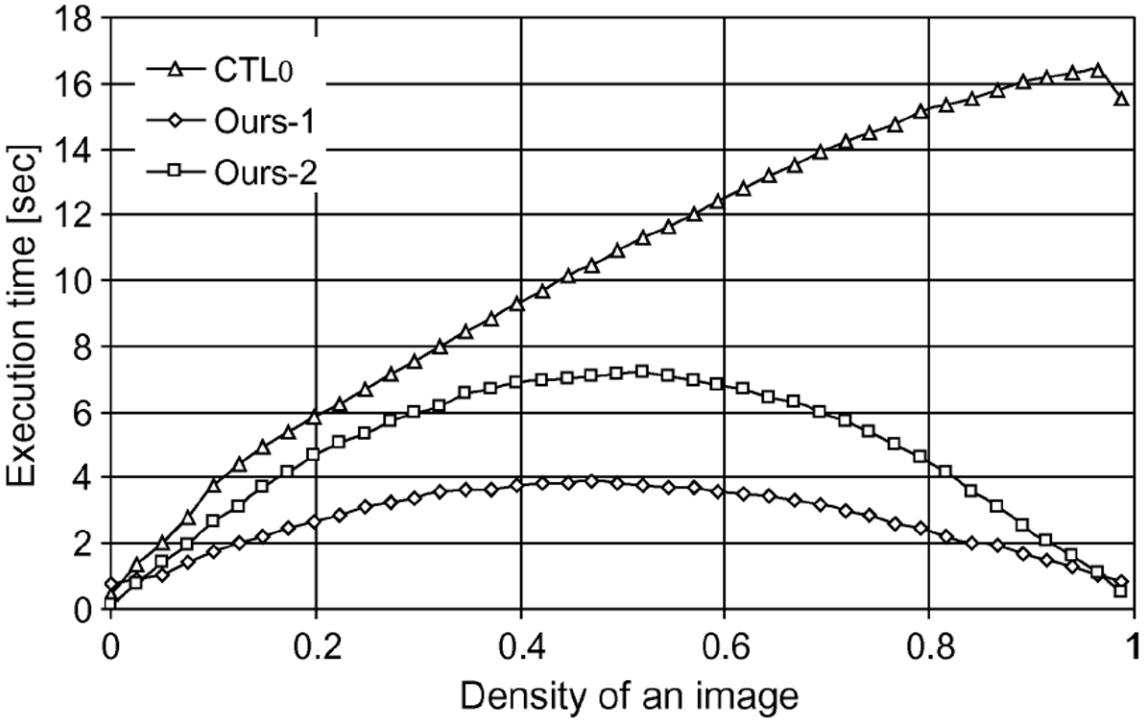


Fig. 9.
Execution time versus the density of images.

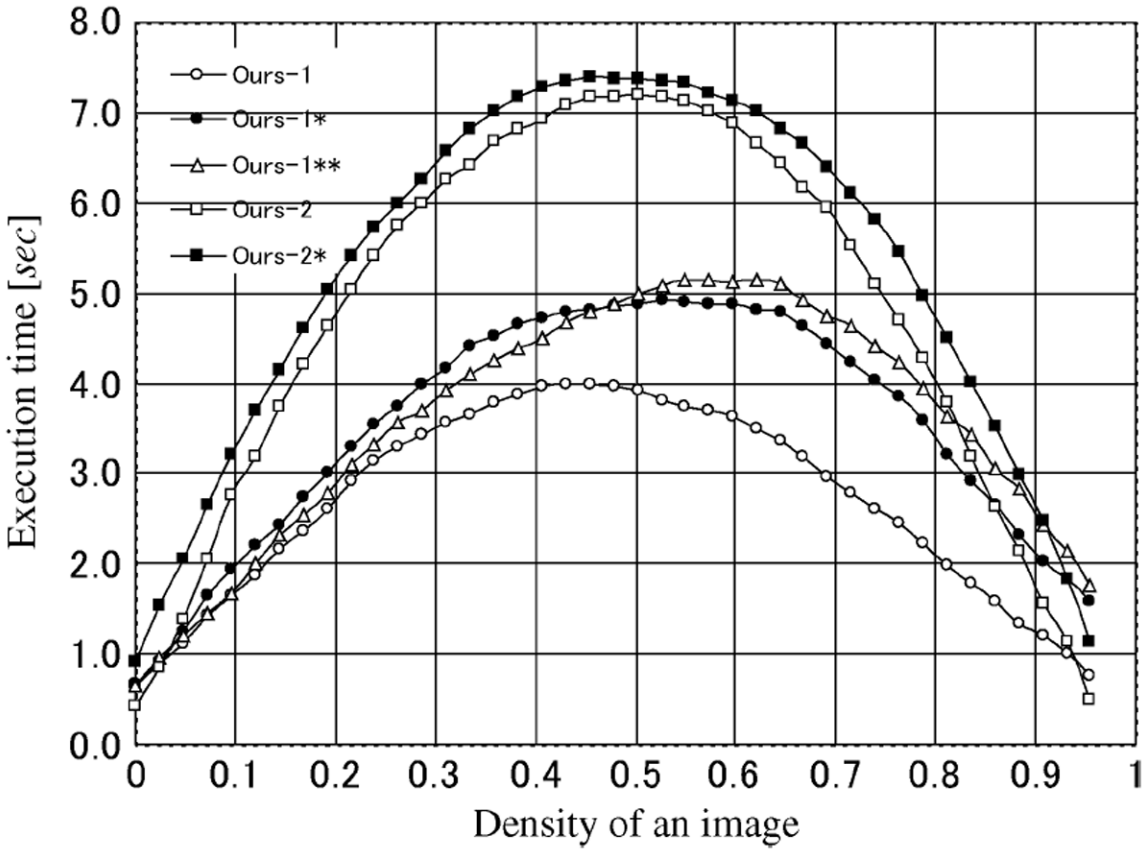


Fig. 10.
Execution time on 512 × 512 × 512 noise images.

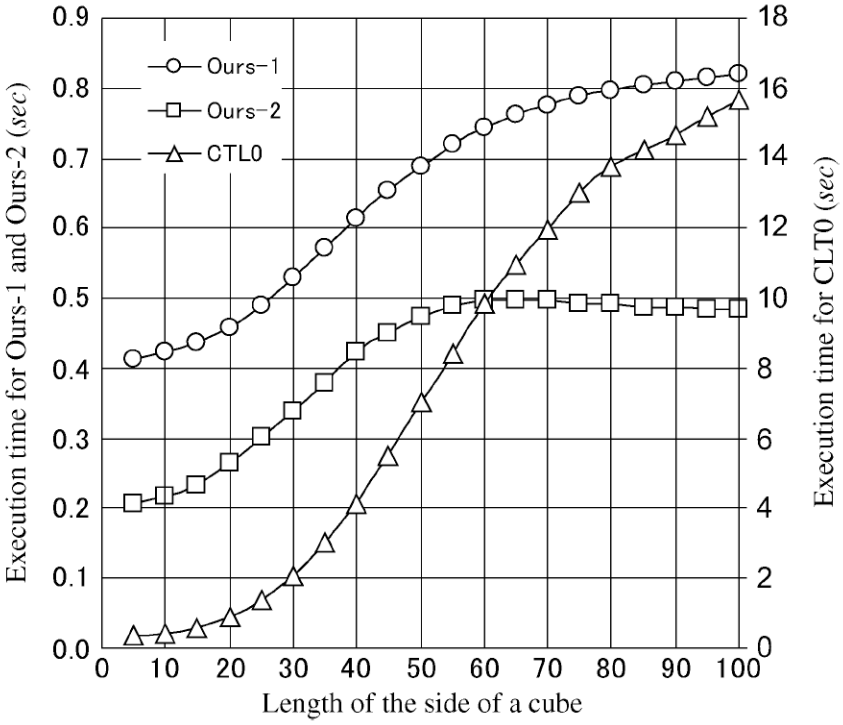
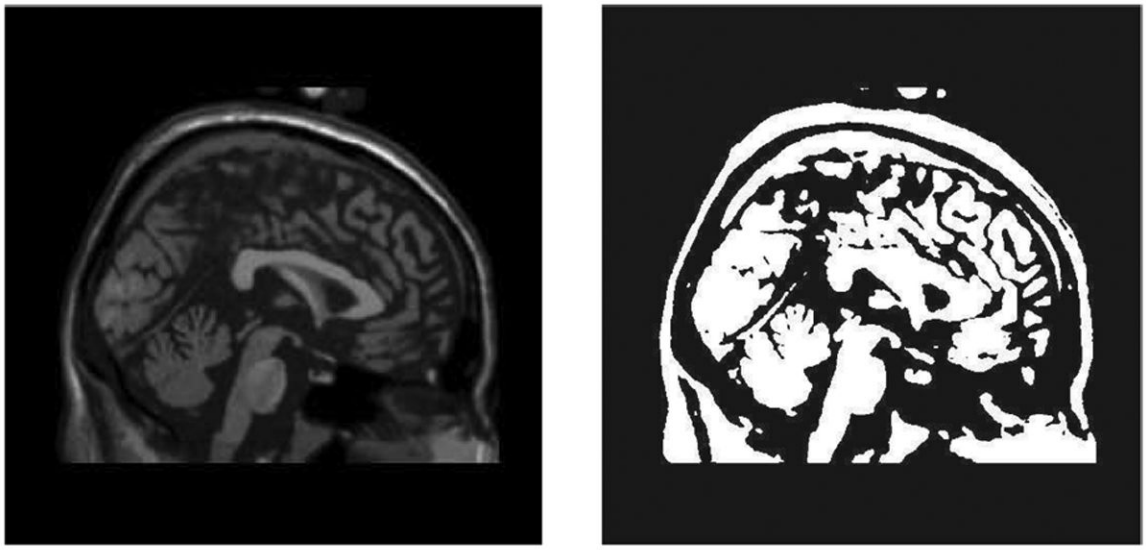


Fig. 11.
Execution time versus the cube size.



(a) (b)

Fig. 12.
Sagittal slice of a 3-D MR image (a) and its corresponding binary image (b).

Fig. 13.
Axial slice of a 3-D binary image of abdominal CT.

TABLE I

Number of Neighboring Voxels to a Voxel in the Mask

v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}	v_{12}	v_{13}
8	6	9	5	6	9	5	8	12	7	4	6	3

TABLE II

Execution Times (in Seconds) of Different Methods on 3-D MR Images

T	D	AL	#CC	Hu	Ours-1	Ours-2
120	2.414	3.6	2343	0.363	0.188	0.102
110	3.463	4.4	1430	0.438	0.197	0.108
100	4.586	5.1	1321	0.517	0.215	0.109
90	5.900	6.1	1058	0.613	0.213	0.110
80	7.223	7.4	737	0.698	0.215	0.118
70	8.333	9.0	536	0.762	0.217	0.110
60	9.232	10.6	286	0.812	0.217	0.110
50	10.152	12.0	201	0.867	0.217	0.109
40	11.256	13.9	270	0.935	0.208	0.109
30	12.600	15.4	3672	1.004	0.202	0.110
20	15.272	11.7	9405	1.219	0.207	0.116
10	20.652	15.8	4	1.543	0.221	0.116

T: threshold; D: density; AL: average length of runs; #CC: number of connected components.