Received 18 November 2022; accepted 25 December 2022. Date of publication 30 December 2022;
date of current version 13 January 2023. The review of this article was arranged by Associate Editor Shih-Chia Huang.

Digital Object Identifier 10.1109/OJCS.2022.3233088

An Efficient Connected-Component Labeling Algorithm for 3-D Binary Images

XIAO ZHAO [1], YUYAN CHAO [2], HUI ZHANG [1], BIN YAO [1], AND LIFENG HE [1,3] (Senior Member, IEEE)

[1] Artificial Intelligence Institute, School of Electronic Information and Artificial Intelligence, Shaanxi University of Science and Technology, Xi'an 710021, China
[2] Graduate School of Environment Management, Nagoya Sangyo University, Owariasa-shi 488-8711, Japan
[3] Graduate School of Information Science and Technology, Aichi Prefectural University, Nagakute 480-1198, Japan

CORRESPONDING AUTHOR: LIFENG HE (e-mail: helifeng@ist.aichi-pu.ac.jp)

This work was supported in part by the Grant-in-Aid for the National Natural Science Foundation of China under Grants 61971272 and 61601271, in part by the Doctoral Scientific Research Foundation of Shaanxi University of Science and Technology under Grant 2019BJ-27, and in part by the Nitto Foundation, Japan.

ABSTRACT Conventional voxel-based algorithms for labeling connected components in 3D binary images use the same mask to process all object voxels. To reduce the number of times that neighboring voxels are checked when object voxels are processed, we propose an algorithm that uses two different masks for processing two different types of object voxels. One type of mask is used when the voxel preceding the object voxel being processed is an object voxel, and the other type is used otherwise. In either case, an optimal order is used for checking the voxels in the corresponding mask. Experimental results demonstrate that our proposed algorithm checked fewer voxels, and was more efficient than conventional algorithms.

INDEX TERMS Computer vision, connected component, labeling algorithm, 3D binary image.

I. INTRODUCTION

Connected-Component labeling (CCL) in a binary image is indispensable for pattern recognition, pattern analysis, computer (robot) vision, and machine intelligence [1]. CCL can transform a binary image to a symbolic image in which all pixels belonging to a connected component (also called an object) are assigned a unique label. Subsequently, using the symbolic image, the features of these connected components (objects) can be extracted.

Many efficient CCL strategies have been proposed for 2D binary images [2]. Efficient two-scan CCL algorithms include pixel-based algorithms [3], [4], [5], run-based algorithms [6], [7], and block-based algorithms [8], [9].

As technologies for image acquisition and manipulation have advanced, 3D images have become widely used in various image-processing and analysis applications [10], such as medical image analysis and computer-aided diagnosis of medical images [11], [12], [13], [14], as well as computer graphics. More accurate and complete information is contained in 3D images than in 2D images. For example, doctors can quickly diagnose diseases using 3D images [15].

CCL for 3D binary images has attracted much attention since the 1980s, and many algorithms [3], [16], [17], [18], [19], [20], [21], [22] have been proposed.

He et al. [21] presented an efficient voxel-based CCL algorithm for 3D binary images (for convenience, we call this algorithm the VCL algorithm). By checking the voxels in the mask in an optimal order, the VCL algorithm can reduce the number of voxels to check when processing each object voxel. However, some voxels that were already checked earlier while processing another voxel may be checked again.

An effective strategy for improving 2D CCL algorithms is to reuse the information obtained while processing each object pixel to reduce the number of pixels that need to be checked when processing another object pixel later. This strategy was first presented in [24], further studied in [3], and has also been extended to block-based CCL algorithms [25], [26].

This article presents an improvement of the VCL algorithm that incorporates the above efficient strategy in the following manner: for an object voxel that follows another object voxel, the algorithm uses the mask M1, which contains much fewer voxels than the conventional mask. Conversely, for an object voxel that follows a background voxel, it uses another mask M2, which is like the mask used in conventional voxel-based

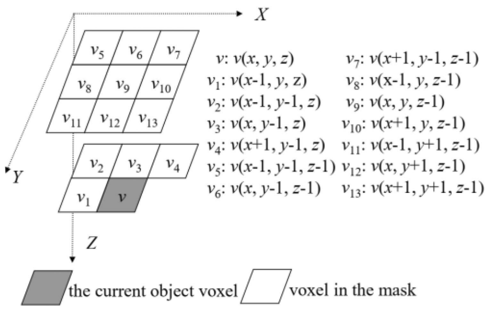


FIGURE 1. Mask M with 26-connectivity for 3-D binary images.

CCL algorithms. In either case, an optimal order is used for checking the voxels in the corresponding mask.

The remainder of the article is organized as follows. We present some preliminaries in Section II and introduce our proposed algorithm in Section III. In Section IV, we present experimental results to evaluate the performance of our proposed algorithm against conventional CCL algorithms. We discuss the results in Section V and present our concluding remarks in Section VI.

II. PRELIMINARIES

For a 3D binary image of size $U \times V \times W$, 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 $1 \leq z \leq W$. For convenience, we also use $v(x, y, z)$ to denote the value of the voxel. Without loss of generality, we use 0 as the value of a background voxel and 1 as the value of an object voxel. We only consider 26-connectivity in this article because the other cases (i.e., 6-connectivity and 18-connectivity) are sub-cases of 26-connectivity. Moreover, all border voxels of an image are assumed to be background voxels.

A. PRINCIPLE OF LABEL-EQUIVALENCE-BASED TWO-SCAN LABELING ALGORITHMS

For a voxel $v(x, y, z)$, the 26 voxels $v(x \pm i, y \pm j, z \pm k)$ such that i, j, k are 0 or 1, except $i = j = k = 0$, are said to be its neighbors. Two object voxels p and q in a binary image are said to be 26-connected if and only if there is a path that consists of object voxels $v_1, \ldots, v_n$ such that $v_1 = p$, $v_n = q$, and for $1 \leq i < n$, v_i and v_{i+1} are neighbors of each other. A connected component (an object) in a binary image is a maximum set of object voxels in the image such that all pairs of voxels in the set are 26-connected.

To label a binary image, a label-equivalence-based two-scan labeling algorithm [2] works as follows. In the first scan, for each voxel $v(x, y, z)$, if $v(x, y, z)$ is a background voxel, it does nothing; otherwise (i.e., $v(x, y, z)$ is an object voxel), it checks whether there is any object voxel in the mask M, which consists of the 13 neighbors of $v(x, y, z)$ that have been processed (Fig. 1). If there are none, the algorithm assigns $v(x, y, z)$ a new provisional label. Otherwise, there are some object voxels in the mask, each of which has been assigned a provisional label during the previous image processing. Because all object voxels in the mask are 26-connected to $v(x, y, z)$, all object voxels in the mask and $v(x, y, z)$ belong to the same object. Therefore, the algorithm assigns any label in the mask to $v(x, y, z)$, records all labels in the mask as equivalent labels, and finds a unique label as the representative of the equivalent labels by resolving equivalent labels.

In this manner, after the first scan, each object voxel has been assigned a provisional label, all provisional labels assigned to the voxels of each object have been recorded as equivalent labels, and a representative label for each group of equivalent labels has been found by resolving equivalent labels. In the second scan, by replacing the provisional label of each object voxel by its representative label, all voxels of an object are assigned a unique label.

B. EQUIVALENT-LABEL-SET STRATEGY FOR LABEL EQUIVALENCE RESOLUTION

The equivalent-label-set strategy is the most efficient strategy for resolving label equivalences [2]. This strategy combines all equivalent provisional labels in a set and uses the smallest label in the set as the representative label of all labels in the set [23]. For convenience, we use $R(l)$ to denote the representative label of the provisional label l, and $S(t)$ to denote the equivalent label set for which t is the representative label. Thus, for each provisional label $x \in S(t)$, $R(x) = t$ holds.

In the first scan, when a new provisional label l is assigned to an object voxel v (i.e., $v = l$), this strategy sets the representative label of l to itself (i.e., $R(l) = l$) and establishes a new equivalent label set $S(l) = \{l\}$. Whenever two provisional labels m and n are found to be equivalent, where $m \in S(u)$ and $n \in S(v)$, all provisional labels in $S(u)$ and $S(v)$ are equivalent. Therefore, $S(u)$ and $S(v)$ are combined by the following procedure *resolve*(u, v).

```
resolve(u, v){
if (u < v)
S(u) = S(u) ∪ S(v);
 for all p ∈ S(v), R(p) = u;
else if (v < u)
S(v) = S(v) ∪ S(u);
 for all p ∈ S(u), R(p) = v;
 }
```

Using this strategy, at any time in the first scan, all provisional labels assigned to a connected component in the processed area of the image are combined in an equivalent label set whose smallest label is their representative label. Therefore, as soon as the first scan has finished, all equivalent labels assigned to an object are combined in an equivalent label set whose smallest label is the representative label of the object.

C. REVIEW OF VCL ALGORITHM

The VCL algorithm is a label-equivalence-based two-scan algorithm that uses the equivalent-label-set strategy for label equivalence resolution. In principle, when processing each object voxel, the voxels in the mask can be checked in any

TABLE 1. Number of Neighbors of 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

order. However, the order of checking may affect the efficiency. Because there are 13 voxels in the mask, the number of possible orders is 13! therefore, it is important to find an optimal order for checking the voxels in the mask.

According to the principle of the equivalent-label-set strategy for label equivalence resolution, when processing an object voxel, each object voxel in the mask has been assigned a provisional label, and all provisional labels assigned to the voxels of each connected part in the mask have been combined in the same equivalent label set during the previous image processing. That is, all label equivalences between the provisional labels of each connected part in the mask have been resolved. Therefore, only the label equivalences between different connected parts in the mask need to be considered.

When checking a voxel in the mask, if the voxel is an object voxel, that voxel and all its neighbors in the mask certainly belong to the same connected part; therefore, it is not necessary to check any of its neighbors further. For this reason, when the VCL algorithm processes an object voxel, it checks voxels in the mask in the order of the numbers of their neighboring voxels in the mask. The number of neighbors of each voxel in the mask shown in Fig. 1 is shown in Table 1. According to Table 1, to process an object voxel, the VCL algorithm checks voxels in the mask in the order (named *Order*-I) $v_9 \rightarrow (v_3, v_6) \rightarrow (v_1, v_8) \rightarrow v_{10} \rightarrow (v_2, v_5, v_{12}) \rightarrow (v_4, v_7) \rightarrow v_{11} \rightarrow v_{13}$.

The VCL algorithm labels a 3D binary image as follows. In the first scan, if the current voxel is a background voxel, it does nothing. Otherwise (i.e., if the current voxel is an object voxel), it checks its neighbors in the mask in *Order*-I. If v_9 is an object voxel, the provisional label of v_9 is assigned to the current voxel. Because v_9 connects with all other object voxels in the mask (if any), no label equivalence needs to be considered. Conversely, if v_9 is a background voxel, then v_3 is checked. If v_3 is an object voxel, the provisional label of v_3 is assigned to the current voxel. Because there may be label equivalences between the provisional label of v_3 and other object voxels in the mask, caused by the existence of the current object voxel, the other voxels in the mask except the neighbors of v_3 (i.e., v_{11}, v_{12}, and v_{13}) should be checked further.

In the case in which v_3 is also a background voxel, v_6 is the next voxel in *Order*-I. The procedure for processing v_6 is like that described above. Other cases can be processed in a similar manner.

Experimental results have demonstrated that the VCL algorithm is much more efficient than previous labeling algorithms for 3D binary images [21]. However, some voxels in the mask that have been checked when processing another object voxel earlier may be checked again. Therefore, the VCL algorithm may perform substantial redundant work, which should be avoided.

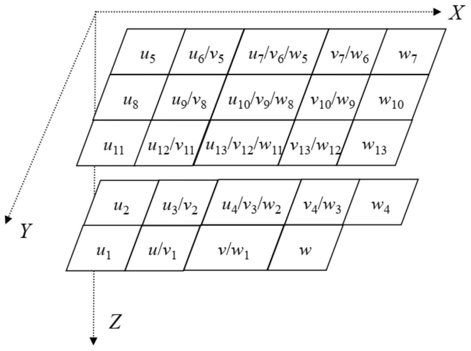


FIGURE 2. Processed neighbors of voxels *v* and *w*.

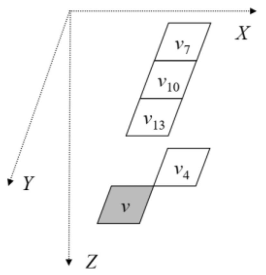


FIGURE 3. Mask M1 for processing an object voxel *v* following another object voxel.

III. PROPOSED ALGORITHM

As noted above, the VCL algorithm uses the same order to check the voxels in the mask when it processes all object voxels. When processing an object voxel v that follows another object voxel u, some voxels in the mask that were checked when processing u may be checked again when processing v. This increases the time for checking voxels and therefore decreases the efficiency of CCL. To solve this problem, we propose an improved VCL algorithm that uses the information obtained while processing an earlier object voxel to label the current object voxel.

A. PROCESSING AN OBJECT VOXEL FOLLOWING ANOTHER OBJECT VOXEL

When the algorithm processes an object voxel v that follows another object voxel u, it can assign the provisional label of v_1 (u) to v. To resolve the related label equivalences, of the 13 processed neighbor voxels of v, the nine voxels v_1 (u), v_2 (u_3), v_3 (u_4), v_5 (u_6), v_6 (u_7), v_8 (u_9), v_9 (u_{10}), v_{11} (u_{12}), and v_{12} (u_{13}) are also the neighbors of u, as shown in Fig. 2. Therefore, all label equivalences (if any) related to these nine voxels have already been resolved while processing u. To process v, the provisional label of u needs to be assigned to v, and the label equivalences (if any) caused by the remaining neighbors of v (i.e., v_4, v_7, v_{10}, and v_{13}) need to be resolved. Accordingly, the mask M1 for processing v consists of v_4, v_7, v_{10}, and v_{13}, as shown in Fig. 3.

The voxels u_3 (v_2), u_4 (v_3), u_6 (v_5), u_7 (v_6), u_9 (v_8), u_{10} (v_9), u_{12} (v_{11}), and u_{13} (v_{12}) may have been checked while processing u. Therefore, when processing v, the information known about these voxels can be used to resolve the related label equivalences. According to how the information to be used, there are the following five cases, where resolve(u, v) is the label equivalence resolution procedure (between u and v) defined in Section II-B.

a) In the case in which $u_{10} = 1$ (i.e., $v_9 = 1$; see Fig. 2), because v_9 connects to v and to each of v_4, v_7, v_{10}, and v_{13}, all related label equivalences (if any) in the mask M1 have been resolved; therefore, no new label equivalence needs to be resolved.

b) In the case in which $[(u_{10} = 0) \wedge \{(u_4 = 1) \vee (u_7 = 1)\} \wedge (u_{13} = 1)]$ (i.e., $[(v_9 = 0) \wedge \{(v_3 = 1) \vee (v_6 = 1)\} \wedge (v_{12} = 1)]$; see Fig. 2), v_4, v_7, v_{10}, and v_{13} all connect to u and v; therefore, no new label equivalence needs to be resolved.

c) In the case in which $[(u_{10} = 0) \wedge \{(u_4 = 1) \vee (u_7 = 1)\} \wedge (u_{13} = 0)]$ (i.e., $[(v_9 = 0) \wedge \{(v_3 = 1) \vee (v_6 = 1)\} \wedge (v_{12} = 0)]$; see Fig. 2), v_4, v_7, and v_{10} connect to u and v, but v_{13} does not. Therefore, only v_{13} needs to be checked. If $v_{13} = 1$, *resolve*(v, v_{13}) is executed.

d) In the case in which $[(u_{10} = 0) \wedge (u_4 = 0) \wedge (u_7 = 0) \wedge (u_{13} = 1)]$ (i.e., $[(v_9 = 0) \wedge (v_3 = 0) \wedge (v_6 = 0) \wedge (v_{12} = 1)]$; see Fig. 2), v_{13} and v_{10} connect to u and v, but v_4 and v_7 do not. Therefore, if $v_4 = 1$, *resolve*(v, v_4) is executed, otherwise, if $v_7 = 1$, *resolve*(v, v_7) is executed.

e) In the other cases, because there is less available information, it is less helpful and can be ignored. In such cases, if $v_9 = 1$, because v_9 connects to v_1 (u) and all v_4, v_7, v_{10}, and v_{13}, no new label equivalence needs to be resolved. Therefore, checking v_9 first is efficient. Conversely, if $v_9 = 0$, the voxels in mask M1 are checked in the order decided by the strategy in the VCL algorithm. Because the numbers of neighbors of v_4, v_7, v_{10}, and v_{13} in the mask M1 are 2, 2, 3, and 1, respectively, the optimal order (named *Order*-II) for checking these voxels is $v_{10} \rightarrow (v_4, v_7) \rightarrow v_{13}$. The procedure for resolving label equivalences in these cases will be given latter.

The information of checked voxels in each of the above cases will be recursively used to process the following object voxel w (if it is). The case for processing w will be also one of Case (a)–Case (e).

In Case (a), we only know $v_9 = 1$, i.e., $w_8 = 1$ (see Fig. 2, similarly hereinafter). Thus, it is Case (e) for processing w. In Case (b), we know that $[(v_9 = 0) \wedge \{(v_3 = 1) \vee (v_6 = 1)\} \wedge (v_{12} = 1)]$, i.e., $[(w_8 = 0) \wedge \{(w_2 = 1) \vee (w_5 = 1)\} \wedge (w_{11} = 1)]$. Thus, it is also Case (e) for processing w. In Case (c), we know that $[(v_9 = 0) \wedge \{(v_3 = 1) \vee (v_6 = 1)\} \wedge (v_{12} = 0)]$, i.e., $[(w_8 = 0) \wedge \{(w_2 = 1) \vee (w_5 = 1)\} \wedge (w_{11} = 0)]$. Thus, for processing w if $v_{13} = 1$, i.e., $w_{12} = 1$, it is Case (d), else Case (e). In Case (d), we know that $[(v_9 = 0) \wedge (v_3 = 0) \wedge (v_6 = 0) \wedge (v_{12} = 1)]$, i.e., $[(w_8 = 0) \wedge (w_2 = 0) \wedge (w_5 = 0) \wedge (w_{11} = 1)]$. Thus, for processing w if $\{(v_4 = 1) \vee (v_7 = 1)\}$, i.e., $\{(w_3 = 1) \vee (w_6 = 1)\}$, it is Case (c), else Case (e).

Now, we consider the procedure for processing v in Case (e). As introduced above, v_9 (i.e., u_{10}/w_8) will be checked first. If $v_9 = 1$, the same as in Case (a), nothing needs to be done. Moreover, if $w = 1$, for processing w, it will be Case (e). On the other hand, if $v_9 = 0$, v_{10} (i.e., u_{11}/w_9) will be checked. If $v_{10} = 1$, *resolve*(v, v_{10}) will be executed. If $w = 1$, for processing w, it will be Case (a). Other situations can be analyzed in a similar way. The procedure for Case (e) (named *Procedure* I) can be described as follows.

```
Procedure I:
    if ($v_9$) {
        if($w$) goto Case (e);
    }
    else if ($v_{10}$) {
        resolve($v$, $v_{10}$);
        if($w$) goto Case (a);
    }
    else if ($v_4$) {
        resolve($v$, $v_4$);
        if ($v_{13}$){
            resolve($v$, $v_{13}$);
            if($w$) goto Case (b);
        }
        else if($w$) goto Case (c);
    }
    else if ($v_7$) {
        resolve($v$, $v_7$);
        if ($v_{13}$){
            resolve($v$, $v_{13}$);
            if($w$) goto Case (b);
        }
        else if($w$) goto Case (c);
    }
    else if ($v_{13}$){
        resolve($v$, $v_{13}$);
        if($w$) goto Case (d);
    }
    else if($w$) goto Case (e);
```

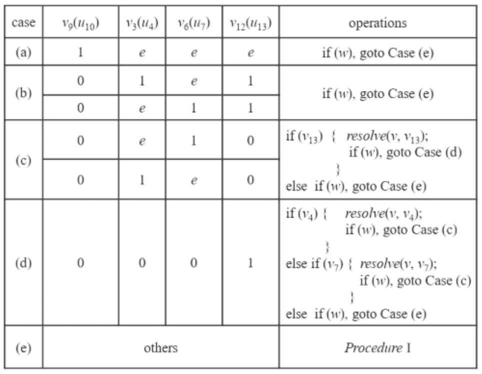
The operations for processing v in the various cases are summarized in Table 2, where "e" means either a background voxel or an object voxel, and w is the next voxel of v. The case transition is shown in Fig. 4. More details are given in Appendix.

B. PROCESSING AN OBJECT VOXEL FOLLOWING A BACKGROUND VOXEL

When processing an object voxel u following a background voxel, all processed neighbor voxels of u except for u_1 (i.e., u_2, …, u_{13}), as shown in Fig. 2, are unknown voxels. Therefore, the mask M2 for processing u in this case consists of u_2, …, u_{13}, as shown in Fig. 5.

Our proposed algorithm decides the order in which to check the voxels in M2 according to the following two strategies.

TABLE 2. Operations in Various Cases



case	$v_9(u_{10})$	$v_3(u_4)$	$v_6(u_7)$	$v_{12}(u_{13})$	operations
(a)	1	*e*	*e*	*e*	if (w), goto Case (e)
(b)	0	1	*e*	1	if (w), goto Case (e)
	0	*e*	1	1	
(c)	0	*e*	1	0	if (v_{13}) { *resolve*(v, v_{13}); if (w), goto Case (d) } else if (w), goto Case (e)
	0	1	*e*	0	
(d)	0	0	0	1	if (v_4) { *resolve*(v, v_4); if (w), goto Case (c) } else if (v_7) { *resolve*(v, v_7); if (w), goto Case (c) } else if (w), goto Case (e)
(e)	others				*Procedure* I

"*e*" means either a background voxel or an object voxel and "*w*" indicates the next voxel of *v*.

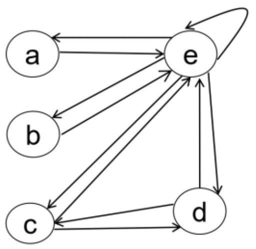


FIGURE 4. Case transition for processing consecutive object voxels by use of M1.

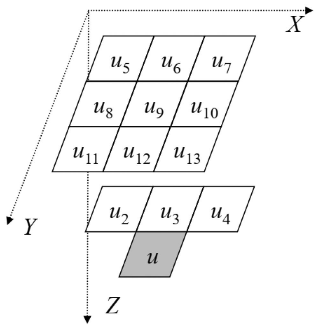


FIGURE 5. Mask M2 for processing an object voxel *u* following a background voxel.

The first strategy is the same as that in the VCL algorithm: a voxel with more neighbors should be checked earlier. The number m of neighbors of each voxel in the mask for processing u is shown in column 2 of Table 3.

The second strategy is based on the following consideration. In the case in which the next voxel v to be processed is also an object voxel, as described in Section III-A, the information on u_4, u_7, u_{10}, and u_{13} obtained while processing u can be used to reduce the number of voxels to check when processing v. Therefore, these voxels should be checked preferentially. Obviously, a voxel with more neighbors from the set $\{v_4, v_7, v_{10}, v_{13}\}$ should be checked earlier. The number of neighbors from $\{v_4, v_7, v_{10}, v_{13}\}$ of each voxel in the mask for processing u is shown in column 3 of Table 3.

TABLE 3. Number of Neighbors of a Voxel in the Mask M2

Voxel	m	n	s
u_2	5	0	5
u_3	8	0	8
u_4	5	3	7
u_5	5	0	5
u_6	8	0	8
u_7	5	3	7
u_8	7	0	7
u_9	11	0	11
u_{10}	7	4	9
u_{11}	3	0	3
u_{12}	5	0	5
u_{13}	3	2	4

Combining the above two strategies, the voxels in the mask for processing v should be checked in descending order of the value $s = m + 0.5 \times n$, as shown in column 4 of Table 3. Here, 0.5 is a weighting coefficient inspired by the observation that the next voxel v may be a background voxel. Thus, the order for checking the voxels in the mask for processing u (named *Order*-III) is $u_9 \rightarrow u_{10} \rightarrow (u_3, u_6) \rightarrow (u_4, u_7, u_8) \rightarrow (u_2, u_5, u_{12}) \rightarrow u_{13} \rightarrow u_{11}$.

The procedure for labeling an object voxel can be defined in a similar way to that of the VCL algorithm.

C. PROCEDURE OF PROPOSED ALGORITHM

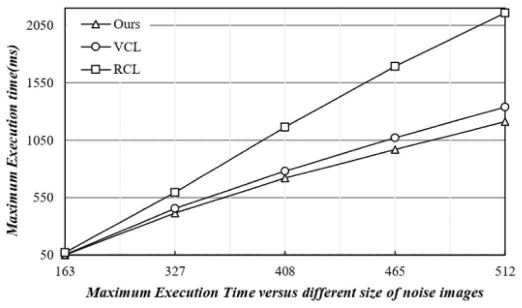
In the first scan, the algorithm does nothing for any of the background voxels. For an object voxel following a background voxel, the voxels in mask M2 (shown in Fig. 5) are checked in the order *Order*-III, and the related label equivalences (if any) are resolved. Then, each following object voxel is processed according to the information about the voxels checked while processing the previous object voxel, as introduced in Section III-A. An illustration of the first scan of our proposed algorithm, *LabelProcessing*, is presented in Appendix.

In the second scan, the provisional label of each object voxel is replaced by its representative label, similarly to the VCL algorithm.

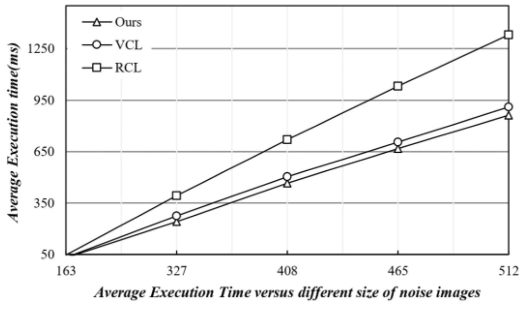
IV. EXPERIMENTAL RESULTS

In this section, we compare our proposed algorithm with VCL and RCL, which is a run-based algorithm for labeling 3D images [21]. The code of the VCL and RCL algorithms was provided by their authors.

All three algorithms were implemented in the C language. All experiments were executed on a PC-based workstation (Intel Xeon CPU E5-1650 v4 @ 3.60 GHz, 4 GB RAM, Ubuntu Linux OS), and were compiled by the GNU C compiler (version 5.16) with the -O3 option. All execution times reported in this section were obtained with the use of one core.



(a)



(b)

FIGURE 6. Maximum and average execution time (ms) versus size of noise images: (a) Maximum execution time, (b) Average execution time.

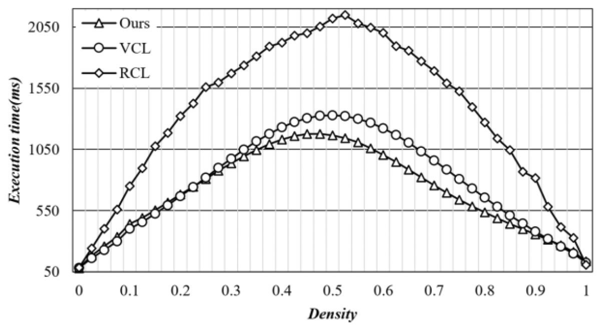


FIGURE 7. Execution time (ms) versus density on 512 × 512 × 512 noise images.

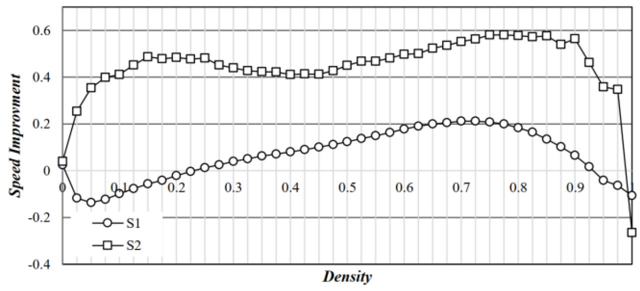


FIGURE 8. Speed improvement versus density on 512 × 512 × 512 noise images (where S1 = (T_{VCL} - T_{Ours})/T_{VCL}, S2 = (T_{RCL} - T_{Ours})/T_{RCL}).

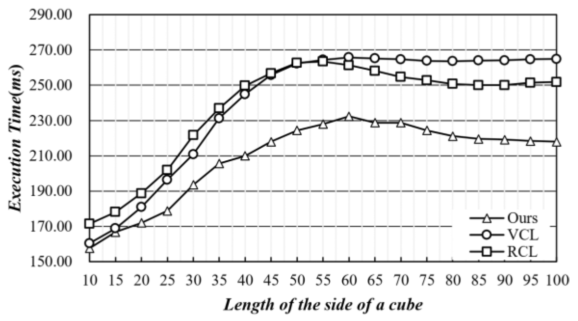


FIGURE 9. Execution time (ms) on 512 × 512 × 512 overlapped-cube images.

All reported results were obtained by averaging the results of 1000 runs.

A. EXPERIMENTAL RESULTS ON NOISE IMAGE SETS

Five sets of 41 uniform noise images, of five different sizes (163 × 163 × 163, 327 × 327 × 327, 408 × 408 × 408, 465 × 465 × 465, and 512 × 512 × 512 voxels) were generated by thresholding images containing uniform random noise with 41 different threshold values (from 0 to 1000 in steps of 25). Because such noise images contain connected components with complicated geometric shapes and complex connectivity, they pose a severe challenge to CCL algorithms.

For the noise images of each size, the maximum and average execution times are shown in Fig. 6. As Fig. 6 shows, as the image size increased, the execution time of all three algorithms increased similarly, and our proposed algorithm was the most efficient of the algorithms with respect to both maximum and average execution time.

We used all noise images of size 512 × 512 × 512 to evaluate how the execution time varied with the density of the object voxels in an image. The execution times are shown in Fig. 7. The results show that our proposed algorithm was more efficient than the VCL algorithm on noise images with densities between 0.3 and 0.9, and was more efficient than the RCL algorithm on all tested images except those with the lowest and highest densities. Fig. 8 shows the speed improvement of our proposed algorithm compared with the VCL and RCL algorithms. The speed improvement compared with VCL is plotted as S1 (= (T_{VCL} - T_{Ours}) / T_{VCL}), and that compared with RCL is plotted as S2 (= (T_{RCL} -T_{Ours}) / T_{RCL}).

B. EXPERIMENTAL RESULTS ON OVERLAPPED-CUBE IMAGE SET

The overlapped-cube image set is composed of images of size 512 × 512 × 512 containing a random distribution of 50 cubes of object voxels, where overlapping cubes are allowed, with cube sizes ranging from 10 × 10 × 10 to 100 × 100 × 100 in steps of 5. The densities of these images range from 0.59% to 85.5%.

The execution times of the algorithms on the overlapped-cube image set are shown in Fig. 9. For almost all the overlapped-cube images, our proposed algorithm was much more efficient than both the VCL and RCL algorithms, particularly for larger cubes. The average speed improvements of our proposed algorithm compared with the VCL and RCL algorithms were 15% and 12%, respectively.

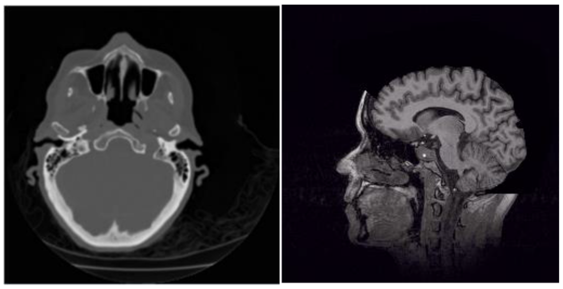

(a) (b)

FIGURE 10. **3D image samples: (a) CThead, (b) MRbrain.**

TABLE 4. **Execution Time (ms) on Medical Images**

Image	VCL	RCL	**Ours**
CThead	9.88	11.08	**7.45**
MRbrain	10.35	12.76	**9.47**

C. EXPERIMENTAL RESULTS ON MEDICAL IMAGES

Two 3D medical images, which were downloaded from a dataset in the University of North Carolina Volume Rendering Test Data Set archive, were used in our experiments. One is a CT head image (named CThead), and the other is an MR brain image (named MRbrain). The size of both images is 256 × 256 × 99. Slices of CThead and MRbrain are shown in Fig. 10(a) and (b), respectively. The two images were binarized by using the optimal threshold obtained by the MATLAB *graythresh* function.

The execution times of the algorithms on the two medical images are presented in Table 4. The table shows that our proposed algorithm is more efficient than both the VCL and RCL algorithms, on both images.

D. EXPERIMENTAL RESULTS COMPARED WITH PRED++_3D ON YACCLAB

An efficient algorithm for labeling 3D binary images, named PRED++_3D, has recently been proposed [16]. This algorithm is also voxel-based and uses the same mask as that used in the VCL algorithm. By using the optimal decision tree, state prediction, and code compression, PRED++_3D performs better than LED_3D, a variant of the VCL algorithm.

We compared our proposed algorithm with the LED_3D and PRED++_3D algorithms on the Yet Another Connected Components Labeling Benchmark (YACCLAB) [27], [28], which is a widely used open-source C++ benchmarking framework for CCL algorithms. The code of the PRED++_3D and LED_3D algorithms was downloaded from https://github.com/prittt/YACCLAB. The code of our proposed algorithm was modified to adapt it to YACCLAB.

All experiments on YACCLAB were executed on a PC-based workstation (Intel Core i7-6700 CPU @ 3.40 GHz, 8 GB RAM, Windows 7 Pro 64-bit).

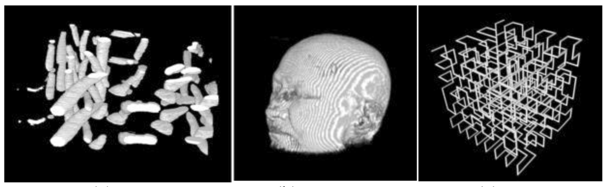
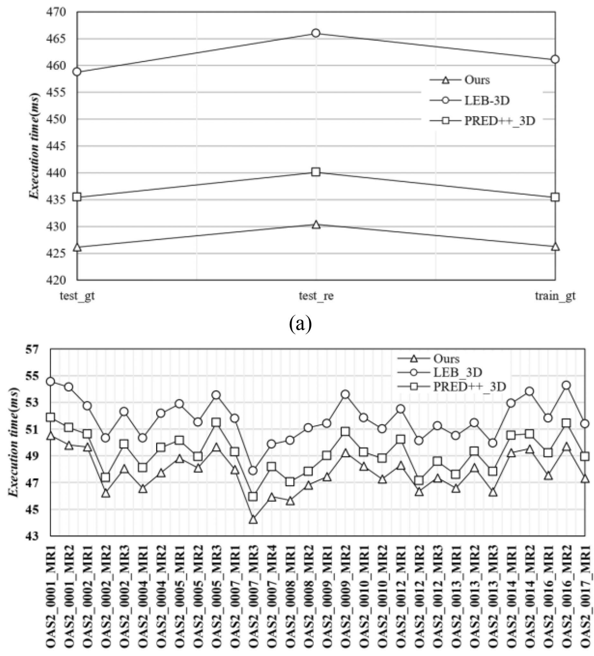
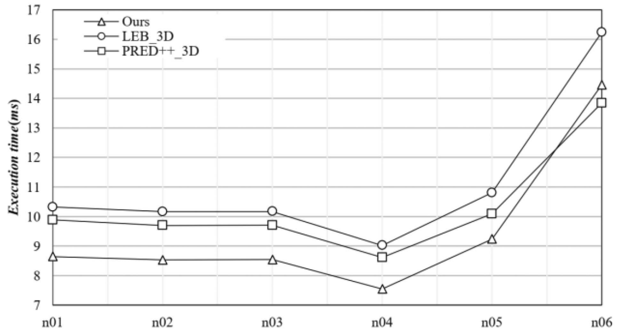

(a) (b) (c)

FIGURE 11. **Image samples: (a) Mitochondria, (b) OASIS, (c) Hilbert.**





(c)

FIGURE 12. **Execution time (ms) on the YACCLAB datasets: (a) Mitochondria, (b) OASIS, (c) Hilbert.**

Our experiments used three images (test_gt, test_re, and train_gt) in Mitochondria, 107 images in OASIS, and six images (n01, n02, n03, n04, n05, and n06) in Hilbert, which are included in the YACCLAB dataset. Image samples are shown in Fig. 11.

The execution times (which were obtained by averaging 50, 10, and 100 runs on Mitochondria, OASIS, and Hilbert, respectively) are shown in Fig. 12. For OASIS, Fig. 12 shows only the execution times on the 30 images from

TABLE 5. **Number of Neighbors to be Checked for Each Configuration in the VCL Algorithm**

case	v_9	v_6	v_3	v_1	v_8	v_{10}	v_5	v_2	v_{12}	v_7	v_4	v_{11}	v_{13}	Number
1	*o*	-	-	-	-	-	-	-	-	-	-	-	-	1
2	*b*	*o*	-	-	-	-	-	-	*o*	-	-	-	-	3
3	*b*	*o*	-	-	-	-	-	-	*b*	-	-	*e*	*e*	5
4	*b*	*b*	*o*	-	-	-	-	-	*o*	-	-	*e*	*e*	6
5	*b*	*b*	*o*	-	-	-	-	-	*b*	-	-	*e*	*e*	6
6	*b*	*b*	*b*	*o*	-	*o*	-	-	-	-	-	-	-	5
7	*b*	*b*	*b*	*o*	-	*b*	-	-	-	*o*	-	-	*e*	7
8	*b*	*b*	*b*	*o*	-	*b*	-	-	-	*b*	*e*	-	*e*	8
9	*b*	*b*	*b*	*b*	*o*	*o*	-	-	-	-	-	-	-	6
10	*b*	*b*	*b*	*b*	*o*	*b*	-	-	-	*o*	-	-	*e*	8
11	*b*	*b*	*b*	*b*	*o*	*b*	-	-	-	*b*	*e*	-	*e*	9
12	*b*	*b*	*b*	*b*	*b*	*o*	*o*	-	-	-	-	*e*	-	8
13	*b*	*b*	*b*	*b*	*b*	*o*	*b*	*e*	-	-	-	*e*	-	9
14	*b*	*b*	*b*	*b*	*b*	*b*	*o*	-	*o*	*o*	-	-	-	9
15	*b*	*b*	*b*	*b*	*b*	*b*	*o*	-	*o*	*b*	*e*	-	-	10
16	*b*	*b*	*b*	*b*	*b*	*b*	*o*	-	*b*	*o*	-	*e*	*e*	11
17	*b*	*b*	*b*	*b*	*b*	*b*	*o*	-	*b*	*b*	*e*	*e*	*e*	12
18	*b*	*b*	*b*	*b*	*b*	*b*	*b*	*o*	*o*	*o*	-	-	-	10
19	*b*	*b*	*b*	*b*	*b*	*b*	*b*	*o*	*o*	*b*	*e*	-	-	11
20	*b*	*b*	*b*	*b*	*b*	*b*	*b*	*o*	*b*	*o*	-	*e*	*e*	12
21	*b*	*b*	*b*	*b*	*b*	*b*	*b*	*o*	*b*	*b*	*e*	*e*	*e*	13
22	*b*	*b*	*b*	*b*	*b*	*b*	*b*	*b*	*o*	*o*	-	-	*e*	11
23	*b*	*b*	*b*	*b*	*b*	*b*	*b*	*b*	*o*	*b*	*e*	-	-	11
24	*b*	*b*	*b*	*b*	*b*	*b*	*b*	*b*	*b*	*o*	-	*e*	*e*	12
25	*b*	*b*	*b*	*b*	*b*	*b*	*b*	*b*	*b*	*b*	*e*	*e*	*e*	13

"*b*," "*o*," "*e*," and "-" mean "background voxel," "object voxel," "either," and "irrelevant voxel," respectively.

TABLE 6. **Number of Neighbors to be Checked for Each Configuration in Mask M2 in Our Proposed Algorithm**

case	u_9	u_{10}	u_6	u_3	u_7	u_4	u_8	u_{12}	u_5	u_2	u_{13}	u_{11}	Number
1	*o*	-	-	-	-	-	-	-	-	-	-	-	1
2	*b*	*o*	-	-	-	-	*o*	-	-	-	-	-	3
3	*b*	*o*	-	-	-	-	*b*	-	*o*	-	-	*e*	5
4	*b*	*o*	-	-	-	-	*b*	-	*b*	*e*	-	*e*	6
5	*b*	*b*	*o*	-	-	-	-	*o*	-	-	-	-	4
6	*b*	*b*	*o*	-	-	-	-	*b*	-	-	*e*	*e*	6
7	*b*	*b*	*b*	*o*	-	-	-	*o*	-	-	-	-	5
8	*b*	*b*	*b*	*o*	-	-	-	*b*	-	-	*e*	*e*	7
9	*b*	*b*	*b*	*b*	*o*	-	*o*	-	-	-	*e*	-	7
10	*b*	*b*	*b*	*b*	*o*	-	*b*	*o*	*o*	-	-	-	8
11	*b*	*b*	*b*	*b*	*o*	-	*b*	*o*	*b*	*e*	-	-	9
12	*b*	*b*	*b*	*b*	*o*	-	*b*	*b*	*o*	-	*e*	*e*	10
13	*b*	*b*	*b*	*b*	*o*	-	*b*	*b*	*b*	*e*	*e*	*e*	11
14	*b*	*b*	*b*	*b*	*b*	*o*	*o*	-	-	-	*e*	-	8
15	*b*	*b*	*b*	*o*	*b*	*o*	*b*	*o*	*o*	-	-	-	9
16	*b*	*b*	*b*	*o*	*b*	*o*	*b*	*o*	*b*	*e*	-	-	10
17	*b*	*b*	*b*	*b*	*b*	*o*	*b*	*b*	*o*	-	*e*	*e*	11
18	*b*	*b*	*b*	*b*	*b*	*o*	*b*	*b*	*b*	*e*	*e*	*e*	12
19	*b*	*b*	*b*	*b*	*b*	*b*	*o*	-	-	-	*e*	-	8
20	*b*	*b*	*b*	*b*	*b*	*b*	*b*	*o*	*o*	-	-	-	9
21	*b*	*b*	*b*	*b*	*b*	*b*	*b*	*o*	*b*	*e*	-	-	10
22	*b*	*b*	*b*	*b*	*b*	*b*	*b*	*b*	*o*	-	*e*	*e*	11
23	*b*	*b*	*b*	*b*	*b*	*b*	*b*	*b*	*b*	*e*	*e*	*e*	12

"*b*," "*o*," "*e*," and "-" mean "background voxel," "object voxel," "either," and "irrelevant voxel," respectively.

TABLE 7. **Number of Neighbors to be Checked for Each Configuration in Mask M1 in Our Proposed Algorithm**

case	$v_9(u_{10})$	$v_6(u_7)$	$v_3(u_4)$	$v_{12}(u_{13})$	v_9	v_{10}	v_7	v_4	v_{13}	Number
1	*o*	-	-	-	*c*	-	-	-	-	0
2	*b*	*o*	-	*o*	*c*	-	-	-	-	0
3	*b*	*o*	-	*b*	*c*	-	-	-	*e*	1
4	*b*	*b*	*o*	*o*	*c*	-	-	-	-	0
5	*b*	*b*	*o*	*b*	*c*	-	-	-	*e*	1
6	*b*	*b*	*b*	*o*	*c*	-	*o*	-	-	1
7	*b*	*b*	*b*	*o*	*c*	-	*b*	*e*	-	2
8	*b*	*b*	*b*	*b*	*c*	*o*	-	-	-	1
9	*b*	*b*	*b*	*b*	*c*	*b*	*o*	-	*e*	3
10	*b*	*b*	*b*	*b*	*c*	*b*	*b*	*e*	*e*	4
11	*n*	-	-	-	*o*	-	-	-	-	1
12	*n*	-	-	-	*b*	*o*	-	-	-	2
13	*n*	-	-	-	*b*	*b*	*o*	-	*e*	4
14	*n*	-	-	-	*b*	*b*	*b*	*e*	*e*	5

"*b*," "*o*," "*e*," "*n*," "-," and "*c*" mean "background voxel," "object voxel," "either," "not checked," "irrelevant voxel," and "checked voxel," respectively.

OAS2_0001_MR1 to OAS2_0017_MR1, instead of all 107 images. This is because the execution times on the remaining images are similar to those shown in Fig. 12(b). Fig. 12 shows that, for the first two datasets, our proposed algorithm was faster than both the LED_3D and PRED++_3D algorithms, particularly on the Mitochondria dataset. On the third dataset, our proposed algorithm is faster than the other two algorithms on all images except n06.

V. DISCUSSION

A. COST EFFECTIVENESS ANALYSIS

The VCL algorithm and our proposed algorithm perform CCL in a similar manner. In the first scan, for each object voxel, both algorithms assign the voxel a provisional label, record all provisional labels in the mask as equivalent, and resolve label equivalences among these provisional labels.

In the second scan, both algorithms replace each provisional label by its representative label. Therefore, the two algorithms have the same time complexity. The reason that our proposed algorithm performs better than the VCL algorithm is that our proposed algorithm checks fewer voxels than the VCL algorithm.

For each object voxel being processed, the VCL algorithm checks the voxels in mask M (shown in Fig. 1) in *Order*-I (defined in Section II-C). As explained in [21], when processing an object voxel, it considers 25 configurations of the mask. For each configuration, the number of times that a voxel in the mask is checked is shown in Table 5, in which "*b*," "*o*," "*e*," and "-" mean "background voxel," "object voxel," "either," and "irrelevant voxel," respectively. An irrelevant voxel is one that does not need to be checked because whether it is an object voxel does not affect the result. An "*e*" voxel does need to be checked, and it could be either an object voxel or a background voxel.

The number of times that neighbors are checked in case i, denoted by N_i, is shown in the rightmost column of Table 5, where $1 \leq i \leq 25$. The probability of being case i, denoted by p_i, is k/W, where W is the number of all configurations of the mask and k is the number of configurations in case i. For example, $p_1 = 2^{12}/2^{13} = 0.5$ and $p_5 = 2^9/2^{13} = 0.0625$. Thus, for the VCL algorithm, the average number of times that it checks voxels when it processes an object voxel is $\sum_{i=1}^{25} p_i N_i = 3.17$.

When our proposed algorithm processes an object voxel v, if the previous voxel is a background voxel, it processes v by using the mask M2 (shown in Fig. 5) and checks the voxels in the mask in *Order*-III (defined in Section III-B). As shown in Table 6, 23 configurations of the mask need to be considered. For each configuration, the number of times that voxels in the mask are checked is shown in the rightmost column of Table 6. The average number of times that voxels are checked is $\sum_{i=1}^{23} p_i N_i = 3.11$.

If the voxel preceding v is an object voxel u, our proposed algorithm processes v by using the mask M1 (shown in Fig. 3) and checking the voxels in the mask in *Order*-II (defined in Section III-A). The cases to be considered are listed in Table 7. Here, u_i and v_j are as defined in Fig. 2, "*c*" means "checked

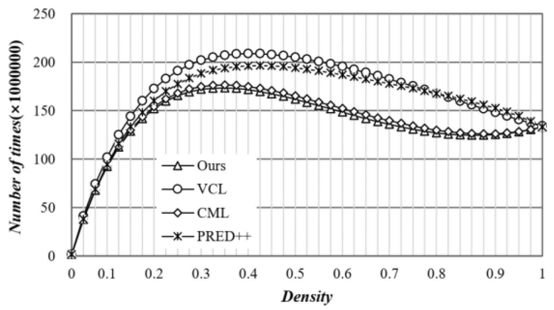


FIGURE 13. Number of times that neighbors are checked on 512 × 512 × 512 noise images.

voxel" (i.e., the voxel has been checked while processing the previous object voxel), and "n" means "not checked" (i.e., the voxel has not been checked while processing the previous object voxel). For each case, the number of times that voxels in the mask are checked is shown in the rightmost column of Table 7.

If u_{10} (v_9) has been checked while processing the previous object voxel u (v_1), one of the cases from case 1 to case 10 holds. The probability of being case i, p_i, can be calculated as explained above. In this situation, the average number of times that voxels are checked is $\sum_{i=1}^{10} p_i N_i = 0.42$.

If u_{10} (v_9) has not been checked while processing the previous object voxel u (v_1), one of the cases from case 11 to case 14 holds. In this situation, the algorithm checks some or all of v_9, v_{10}, v_7, v_4, and v_{13}.

The probability of being case i, p_i, can also be calculated as explained above. The average number of times that voxels are checked is $\sum_{i=11}^{14} p_i N_i = 2.13$.

According to Table 7, the probability that u_{10} (v_9) is checked while processing u is 0.5. Therefore, when processing an object voxel following another object voxel by using the mask M1, the average number of times that voxels are checked is $0.5 \times 0.42 + 0.5 \times 2.13 = 1.27$.

Suppose that M1 and M2 are used equivalently; then, to process an object voxel, the average number of times that voxels are checked by our proposed algorithm is $0.5 \times 3.11 + 0.5 \times 1.27 = 2.19$. Because 2.19 is less than 3.17 (the average number of times that voxels are checked when the VCL algorithm processes an object voxel), our proposed algorithm checks fewer voxels than the VCL algorithm. Therefore, our proposed algorithm is more efficient than the VCL algorithm.

We used all the 512 × 512 × 512 noise images to count the numbers of voxels checked by the VCL algorithm, the CML algorithm (which is a variant of our proposed algorithm that uses only m, shown in Table 3, to decide the order for checking the voxels in the mask M2), the PRED++ algorithm, and our proposed algorithm. The results are shown in Fig. 13, which shows that our proposed algorithm checks fewer voxels than the other three algorithms for all images. This is the reason that our proposed algorithm is more efficient than the conventional algorithms.

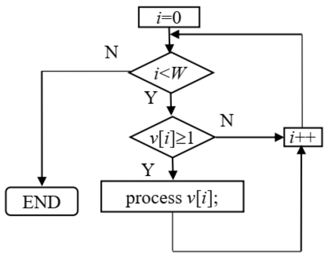


FIGURE 14. Processing flow of the first scan of the VCL algorithm.

B. COMPARISON OF IMPLEMENTATION ARCHITECTURE AND FLOW

The first scan of our proposed algorithm is an improvement of that of the VCL algorithm. The VCL algorithm processes all object voxels in the same manner, and therefore its implementation architecture is very simple. The processing flow is shown in Fig. 14, where W is the size of the input image. In contrast, our proposed algorithm processes object voxels differently, depending on whether they follow a background voxel or another object voxel. Moreover, when our proposed algorithm processes an object voxel following another object voxel, it uses the information obtained while processing the previous object voxel, and the information gathered while processing this object voxel is used to process the following object voxel (if it is one). The cases (a)-(e) shown in Table 2 demonstrates how to process an object voxel based on the information of already checked pixels when Mask M1 is used, and the case transition diagram shown in Fig. 4 shows the case transition when successive object voxels are processing. Because recursion is necessary to implement our proposed algorithm, the implementation architecture of our proposed algorithm is much more complex than that of the VCL algorithm (see Appendix.).

C. COMPARISON WITH PRED++_3D ALGORITHM

Similar to our proposed algorithm, when processing object voxels, the PRED++_3D algorithm also uses known information for state prediction. However, PRED++_3D uses the same mask as that used in the VCL algorithm to process all object voxels, whereas our proposed algorithm uses two different masks for processing two different types of object voxels.

In addition to the two masks, our proposed algorithm introduces two new improved strategies. The first strategy is a method to decide the order in which to process voxels in the mask when processing the object voxels that follow a background voxel. The second strategy is to ignore voxels that do not provide information for processing an object voxel that follows another object voxel. The latter improvement reduces the number of cases to be considered; moreover, in such cases, an efficient procedure (i.e., *Procedure* I, presented in Section III-A) is used, which checks the voxel above the current object voxel first.

Because of these improvements, our proposed algorithm performed better than PRED++_3D.

D. LIMITATION OF PROPOSED ALGORITHM

According to the analysis in Section V-A, when processing an object voxel following a background voxel, the average numbers of times that voxels in the corresponding mask are checked by the VCL algorithm and our proposed algorithm are 3.17 and 2.19, respectively; these numbers are very similar. For a low-density noise image, the voxel preceding an object voxel is very likely to be a background one. Therefore, the efficiencies of the VCL algorithm and our proposed algorithm are almost the same for low-density noise images (with densities between 0.0 and 0.3).

Conversely, when processing an object voxel $v(x, y, z)$, both the VCL algorithm and our proposed algorithm first check the voxel above $v(x, y, z)$ (i.e., $v(x, y, z\text{-}1)$), and do nothing to resolve label equivalences if $v(x, y, z\text{-}1)$ is an object voxel. Therefore, for high-density noise images (with densities greater than 0.9), in which the voxel above the object voxel being processed is very likely to be an object voxel, the efficiencies of the two algorithms are also almost the same.

This analysis indicates that our proposed algorithm has no advantage for processing very low-density or very high-density images. This is the main limitation of our proposed algorithm.

Additionally, as noted in Section V-B, the source code of our proposed algorithm is much longer than that of the VCL algorithm, and our proposed algorithm contains recursive procedures, whereas the VCL algorithm does not. Usually, a longer and recursive program requires more execution time. This may be the reason that the execution time on noise images is not completely proportional to the number of times that voxels are checked.

VI. CONCLUSION

In this article we proposed an efficient CCL algorithm for 3D binary images. Our proposed algorithm uses two different masks for processing object voxels, depending on whether the voxel preceding the object voxel being processed is an object voxel. In either case, the algorithm checks the voxels in the corresponding mask in an optimal order. Experimental results demonstrate that our proposed algorithm checked fewer voxels than the VCL algorithm for all 3D binary images tested, and that our proposed algorithm was more efficient than the VCL and RCL algorithms for almost all noise images, the overlapped-cube image set, and the medical images. Experimental results on YACCLAB also demonstrated that our proposed algorithm was more efficient than the PRED++_3D algorithm.

Our research results can play an important role in many applications, such as detecting the number and volume of blood clots in blood vessels, detecting the number and volume of crystals in material experiments, and determining the volume of materials required for 3D printers by calculating the volumes of parts. In future work, we will construct a hardware implementation and parallel implementation of the algorithm. We will also try to use the strategies proposed in [16] to automatically generate and compress the code of our proposed algorithm, which may improve its speed further.

APPENDIX

```
LabelProcessing:
 l = 1;
 W = Xsize×Ysize×Zsize;
 for(i = 0; i < W; i++) {
     if (v[i]){
       • check the voxels in the mask M2 of v[i] by the order Order-III;
       • resolving each label equivalence of u and v by calling resolving(u, v);
       • if (there is an object voxel v[j] in the mask)
               v[i] = v[j];
         else {
               v[i] = l; l = l + 1;
            }
       • when the status of checked voxels is Case (a) or Case (b)
            i++;
            if(v[i]) {
                v[i] = v[i-1];
                goto processing_e;
              }
       • when the status of checked voxels is Case (c)
       processing_c:
                  i++;
                  if(v[i]) {
                      v[i] = v[i-1];
                      if (v13) {
                            resolve(v[i], v13);
                            goto processing_d;
                         }
                      else {
                           goto processing_e;
                         }
                    }
       • when the status of checked voxels is Case (d)
         processing_d:
                    i++;
                    if(v[i]) {
                        v[i] = v[i-1];
                        if (v4) {
                              resolve(v[i], v4);
                              goto processing_c;
                           }
                       else if (v7){
                              resolve(v[i], v7);
                              goto processing_c;
                           }
                            else goto processing_e;
                      }
       • when the status of checked voxels is Case (e)
           processing_e:
                     i++;
                     if(v[i]) {
                         v[i] = v[i-1];
                         Procedure I;
                       }
             }
   }
```

ACKNOWLEDGMENTS

We thank the anonymous referees for their valuable comments, which greatly improved this article. We are grateful to the associate editor, Prof. Song Guo, for his kind cooperation.

We thank Edanz (https://jp.edanz.com/ac) for editing a draft of this manuscript.

REFERENCES

[1] R. C. Gonzalez, R. E. Woods, and B. R. Masters, "Digital image processing, third edition," *J. Biomed. Opt.*, vol. 14, no. 2, pp. 257–262, Feb. 2009.

[2] L. He, X. Ren, Q. Gao, X. Zhao, B. Yao, and Y. Chao, "The connected-component labeling problem: A review of state-of-the-art algorithms," *Pattern Recognit.*, vol. 70, no. 10, pp. 25–43, Oct. 2017.

[3] L. He, X. Zhao, Y. Chao, and K. Suzuki, "Configuration-transition-based connected-component labeling," *IEEE Trans. Image Process.*, vol. 23, no. 2, pp. 943–951, Feb. 2014.

[4] X. Zhao et al., "A new connected-component labeling algorithm," *IEICE Trans. Inf. Syst.*, vol. 98, no. 11, pp. 2013–2016, Nov. 2015.

[5] L. He, Y. Chao, K. Suzuki, and K. Wu, "Fast connected-component labeling," *Pattern Recognit.*, vol. 42, no. 9, pp. 1977–1987, Sep. 2009.

[6] L. He, Y. Chao, and K. Suzuki, "A run-based two-scan labeling algorithm," *IEEE Trans. Image Process.*, vol. 17, no. 5, pp. 749–756, May 2008.

[7] L. He, Y. Chao, and K. Suzuki, "A run-based one-and-a-half-scan connected-component labeling algorithm," *Int. J. Pattern Recognit. Artif. Intell.*, vol. 24, no. 4, pp. 557–579, Apr. 2010.

[8] C. Grana, D. Borghesani, and R. Cucchiara, "Optimized block-based connected components labeling with decision trees," *IEEE Trans. Image Process.*, vol. 19, no. 6, pp. 1596–1609, Jun. 2010.

[9] W. Chang, C. Chiu, and J. Yang, "Block-based connected-component labeling algorithm using binary decision trees," *Sensors*, vol. 15, no. 9, pp. 23763–23787, Feb. 2015.

[10] C. C. Queirolo, L. Silva, O. R. P. Bellon, and M. Pamplona Segundo, "3D face recognition using simulated annealing and the surface interpenetration measure," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 32, no. 2, pp. 206–219, Feb. 2010.

[11] W. R. Crum, O. Camara, and D. L. G. Hill, "Generalized overlap measures for evaluation and validation in medical image analysis," *IEEE Trans. Med. Imag.*, vol. 25, no. 11, pp. 1451–1461, Nov. 2006.

[12] K. W. Finnis, Y. P. Starreveld, A. G. Parrent, A. F. Sadikot, and T. M. Peters, "Three-dimensional database of subcortical electrophysiology for image-guided stereotactic functional neurosurgery," *IEEE Trans. Med. Imag.*, vol. 22, no. 1, pp. 93–104, Jan. 2003.

[13] B. Rosenhahn, T. Brox, and J. Weickert, "Three-dimensional shape knowledge for joint image segmentation and pose tracking," *Int. J. Comput. Vis.*, vol. 73, no. 3, pp. 243–262, Mar. 2007.

[14] K. Suzuki, H. Yoshida, and J. Nappi, "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.*, vol. 35, pp. 694–703, 2008.

[15] X. Y. Chen, X. W. Song, and H. Xiao, "The diagnostic value of 128 slice spiral CT 3D imaging in cerebral infarction and aneurysms," *Chin. J. Ct Mri*, vol. 13, no. 7, pp. 7–10, Jul. 2015.

[16] F. Bolelli, S. Allegretti, and C. Grana, "One DAG to rule them all," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 44, no. 7, pp. 3647–3658, Jul. 2022.

[17] R. Lumia, L. Shapiro, and O. Zungia, "A new connected components algorithm for virtual memory computers," *Comput. Vis. Graph. Image Process.*, vol. 22, no. 2, pp. 287–300, Feb. 1983.

[18] L. Thurfjell, E. Bengtsson, and B. Nordin, "A new three-dimensional connected components labeling algorithm with simultaneous object feature extraction capability," *CVGIP Graphical Model Image Process.*, vol. 54, no. 4, pp. 357–364, Apr. 1992.

[19] J. Udupa and V. G. Ajjanagadde, "Boundary and object labelling in three-dimensional images," *Comput. Vis. Graph. Image Process.*, vol. 51, no. 3, pp. 355–369, Mar. 1990.

[20] Q. Hu, G. Qian, and W. L. Nowinski, "Fast connected-component labeling in three-dimensional binary images based on iterative recursion," *Comput. Vis. Image Understanding*, vol. 99, pp. 414–434, 2005.

[21] L. He, Y. Chao, and K. Suzuki, "Two efficient label-equivalence-based connected-component labeling algorithms for 3-D binary images," *IEEE Trans. Image Process.*, vol. 20, no. 8, pp. 2122–2134, Aug. 2011.

[22] N. Maurice, F. Lemaitre, J. Sopena, and L. Lacassagne, "LSL3D: A run-based connected component labeling algorithm for 3D volumes," in *Proc. Int. Conf. Image Anal. Process.*, 2022, pp. 132–142.

[23] L. He, Y. Chao, and K. Suzuki, "A linear-time two-scan labeling algorithm," in *Proc. IEEE Int. Conf. Image, Process.*, 2007, pp. 241–244.

[24] L. He et al., "An efficient first-scan method for label-equivalence-based labeling algorithms," *Pattern Recognit. Lett.*, vol. 31, no. 1, pp. 28–35, 2010.

[25] C. Grana et al., "Optimized connected components labeling with pixel prediction," in *Proc. Int. Conf. Adv. Concepts Intell. Vis. Syst.*, 2016, pp. 431–440.

[26] F. Bolelli, S. Allegretti, L. Baraldi, and C. Grana, "Spaghetti labeling: Directed acyclic graphs for block-based connected components labeling," *IEEE Trans. Image Process.*, vol. 29, pp. 1999–2012, 2020.

[27] C. Grana, F. Bolelli, L. Baraldi, and R. Vezzani, "YACCLAB - yet another connected components labeling benchmark," in *Proc. IEEE 23rd Int. Conf. Pattern Recognit.*, 2016, pp. 3109–3114.

[28] F. Bolelli, M. Cancilla, L. Baraldi, and C. Grana, "Toward reliable experiments on the performance of connected components labeling algorithms," *J. Real-Time Image Process.*, vol. 17, pp. 229–244, 2020.

XIAO ZHAO received the B.E., M.S., and Ph.D. degrees from the Shaanxi University of Science and Technology, Xi'an, China, in 2001, 2006, and 2019, respectively. She is currently an Associate Professor with the Shaanxi University of Science and Technology. From 2017 to 2018, she was with Aichi Prefectural University, Nagakute, Japan, as a Research Associate. Her research interests include image processing, artificial intelligence, pattern recognition, and string searching.

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, Nagoya, 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, Nagoya Institute of Technology, Nagoya. She is currently a Professor with Nagoya Sangyo University, Owariasahi, Japan, and a Guest Professor with the Shaanxi University of Science and Technology, Xi'an, China. Her research interests include image processing, graphic understanding, CAD, pattern recognition, and automated reasoning.

HUI ZHANG received the B.E. degree from the Xi'an University of Technology, Xi'an, China, in 2020. He is currently working toward the graduation degree with the School of Electronic Information and Artificial Intelligence, Shaanxi University of Science and Technology, Xi'an. His research interests include image processing and pattern recognition.

BIN YAO received the B.E., M.S., and Ph.D. degrees from the Shaanxi University of Science and Technology, Xi'an, China, in 2003, 2006, and 2019, respectively. He is currently an Associate Professor with the Shaanxi University of Science and Technology. His research interests include image processing and artificial intelligence.

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