Decision Tree for 3-D Connected Components Labeling

Phaisarn Sutheebanjard
Chalermkarnchana College
Phetchabun 67000, Thailand
dr.phaisarn@gmail.com

***Abstract*—3-D connected components labeling is the fundamental technique in Magnetic Resonance Imaging (MRI). This research presents the decision tree for 3-D connected components labeling operation (26-connectivity) in 3-D binary images. This method will be used at the first round of the two-scan connected components labeling process. The distinctive characteristic of this method is its simplicity of the model by using the widely used fundamental algorithm which is the binary decision tree approach. This is very useful for developers to understand and implement the model rapidly. Moreover, this model can work correctly and rapidly with less time-consuming. It spends processing time just a little bit more than the fastest-processing model. Therefore, this method can be considered as one of the interesting techniques to perform 3-D connected components labeling algorithm.**

Keywords—connected components, decision tree, image processing, labeling algorithm, pattern recognition

I. INTRODUCTION

Connected components labeling in binary image is one of the most fundamental operations in many image processing. The labeling operation finds all connected components in an image and assigns a unique label to all points in the same component. There have been many algorithms proposed for addressing this operation. Generally, these algorithms are categorized into four classes as the following: (i) one-scan [1-2], (ii) two-scan [3-11], (iii) multi-scan [12] and (iv) contour tracing [13] algorithms.

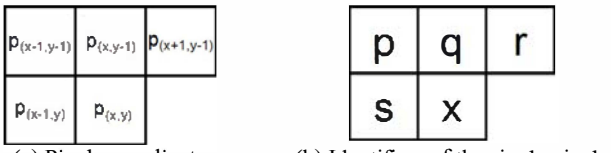
Accordingly to Grana *et al*. [3], two-scan is the fastest algorithm for labeling the 2-dimension connected components. Two-scan is a simple and efficient algorithm in computation time that was previously introduced by Rosenfeld and Pfaltz in 1966 [4]. It consists of three classical operations:

1. *First image scan*: (provisional label assignment and collection of label equivalences)
2. *Equivalences resolution*: (equivalence classes creation)
3. *Second image scan*: (final label assignment)

First image scan: This is an operation in the classical two-scan labeling algorithm which accesses the pixels sequentially in raster scan order to find the 8-connectivity in 2-D binary image by using the scan mask as shown in Fig. 1 [5], and find the 26-connectivity in 3-D binary image by using the scan mask as shown in Fig 2.

978-1-4673-2108-2/12/$31.00 ©2012 IEEE

The *equivalences resolution*: This is an operation that creates an equivalence table containing the information needed to assign unique labels to each connected component. In the first image scan, all those labels that belong to one component are declared equivalent. In the *second image scan*, one label from an equivalent class is selected to be assigned to all pixels of a component.



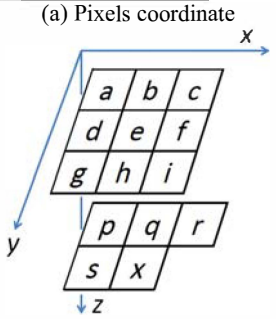
(a) Pixels coordinate (b) Identifiers of the single pixels

Figure 1. 2-D scan mask (8-connectivity)

a:$P_{(x-1,y-1,z-1)}$	b:$P_{(x,y-1,z-1)}$	c:$P_{(x+1,y-1,z-1)}$
d:$P_{(x-1,y,z-1)}$	e:$P_{(x,y,z-1)}$	f:$P_{(x+1,y,z-1)}$
g:$P_{(x-1,y+1,z-1)}$	h:$P_{(x,y+1,z-1)}$	i:$P_{(x+1,y+1,z-1)}$

p:$P_{(x-1,y-1,z)}$	q:$P_{(x,y-1,z)}$	r:$P_{(x+1,y-1,z)}$
s:$P_{(x-1,y,z)}$	x:$P_{(x,y,z)}$	

(a) Pixels coordinate



(b) Identifiers of the single pixels

Figure 2. 3-D scan mask (26-connectivity)

Recently, He *et al*. [16] presented the algorithm for 3-D connected component labeling (26-connectivity) which is considered to be the fastest approach. However, this approach might not be simple for developers to use this method for further developing. Therefore, this paper will propose the simpler method by using fundamental algorithm which is the binary decision tree model. The advantages of this model are it is not only easy for developers but also spends just a little processing time more than the fastest approach which is presented by He *et al*.

II. TWO-SCAN ALGORITHM

The two-scan algorithm is a method used for labeling the connected components in a binary image. There are three classical operations in the two-scan algorithm: first image scan, equivalences resolution and second image scan. This section presents the literature related to first image scan operations. The algorithms used in the first scan image operation are classified into 2-dimension and 3-dimension.

A. 2-dimension

This operation accesses the pixels sequentially in raster scan order for finding the 8-connectivity by using the pixel-based scan mask as shown in Fig. 1. The conditions outcomes are given by all possible combinations of five Boolean variables ("*p*", "*q*", "*r*", "*s*", "*x*"). The actions belong to four classes: no action, new label, assign, and merge [3].

1. *No action:* is performed if the current pixel belongs to the background.
2. *New label:* is created when the neighborhood is only composed of background pixels.
3. *Assign action:* current pixel can take on any existing provisional label in the mask without the need for consideration of label equivalences (either only one pixel is foreground or all pixels share the same label).
4. *Merge action:* is performed to solve equivalence between two or more classes and a representative is assigned to the current pixel.

In 2005, Wu *et al*. [6] proposed a decision tree as shown in Fig. 3 to examine the neighbors of the connected components. A decision tree is a binary tree whose non-terminal nodes are conditional variables and whose terminal nodes are actions to be performed. A decision tree will be defined as being optimal if it has a minimal number of non-terminal nodes and terminal nodes. Wu *et al*. [6] suggested the idea that every pixel in the scan mask is always the neighbor of "*q*" (see Fig. 1). If there is enough equivalence information to access the correct label of "*q*", there is no need to examine the rest of the neighbors. Therefore, their decision tree minimizes the number of scanned neighbors.

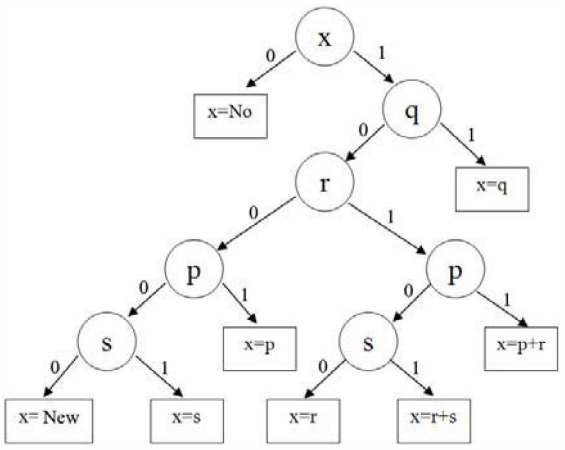


Figure 3. The decision tree used in scanning for 8-connectivity.

Instead of using the decision tree, He *et al*. [7], in 2009, analyzed the mask for 8-connectivity containing 16 possible cases (not including "*x*", which is the background), as shown in Fig. 4. Case 1 is the *new label action*, cases 2-9 and 13-16 are the *assign action* and cases 10-12 are the *merge action*. Based on these cases, they proposed the algorithm as shown in Fig. 5.

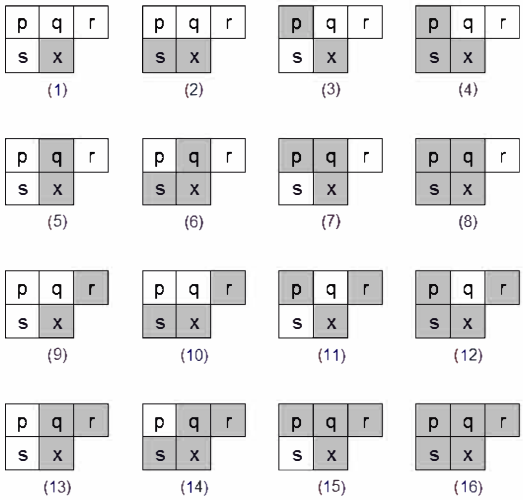


Figure 4. Sixteen possible cases for the current object pixel in the mask for eight-connectivity proposed by He *et al*.

```
 1: If (x==Foreground)
 2:    If (q==Foreground)
 3:       x=q
 4:    else if (s==Foreground)
 5:       x=s
 6:       if (r==Foreground)
 7:          x=r+s
 8:    else if (p==Foreground)
 9:       x=p
10:       if (r==Foreground)
11:          x=p+r
12:    else if (r==Foreground)
13:       x=r
14:    else
15:       x=new label
16: else
17:    no action
```

Figure 5. He *et al*. first-scan algorithm

B. 3-dimension

Labeling of connected components in 3-D binary images has been studied from the Lumia *et al*. in 1983 [14] and Shirai in 1987 [15]. Recently, He *et al*. in 2010 [16] present an efficient method of deciding the order for checking voxels in the mask. They analyze number of the neighbor voxels that need to be checked. The number of the neighbor voxels of each voxel in the mask is shown in Table I.

TABLE I. THE NUMBER OF NEIGHBORING VOXELS TO A VOXEL IN THE SCAN MASK.

a	b	c	d	e	f	g	h	i	p	q	R	s
6	9	5	8	12	7	4	6	3	6	9	5	8

They can reduce number of voxel to be check, but in the worst case, their algorithm working on all neighbors.

III. PROPOSED DECISION TREE

When considering the pixel "x" in terms of provisional label, it can be seen that Suzuki's algorithm [12] scrutinize in all neighbors within the area of scan masks. In Fig 2, it can be seen that the area of scan masks contains all neighbors of "e". If the label of "e" can be accessed precisely, it is not necessary to consider other neighbors. By neglecting the accuracy aspect, the decision tree can be presented as a method for organizing scan operation in a particular order [6]. The proposed model was shown in Table II. Please note that dash ('-') indicate the indent.

TABLE II. THE PROPOSED DECISION TREE ALGORITHM.

```
if (condition_x) {
-if (condition_e) {
--goto Assign E;
-} else {
--if (condition_q) {
---if (condition_h) {
----if (condition_d) {
-----goto Assign Q;
----} else {
-----if (condition_f) {
------goto Assign Q;
-----} else {
------goto Merge HQ;
-----} //else f
----} //else d
---} else {
----if (condition_g) {
-----if (condition_d) {
------if (condition_i) {
-------goto Merge QI;
------} else {
-------goto Assign Q;
------} //else i
-----} else {
------if (condition_i) {
-------goto Merge GIQ;
------} else {
-------goto Merge GQ;
------} //else i
-----} //else g
----} else {
-----if (condition_i) {
------if (condition_f) {
-------goto Assign Q;
------} else {
-------goto Merge IQ;
------} //else f
-----} else {
------goto Assign Q;
-----} //else i
----} //else g
---} //else h
--} else {
---if (condition_b) {
----if (condition_h) {
-----if (condition_d) {
------goto Assign B;
-----} else {
------if (condition_f) {
-------goto Assign B;
------} else {
-------goto Merge BH;
------} //else f
-----} //else d
----} else {
-----if (condition_g) {
------if (condition_d) {
-------if (condition_i) {
--------if (condition_f) {
---------goto Assign B;
--------} else {
---------goto Merge BI;
--------} //else f
-------} else {
--------goto Assign B;
-------} //else i
------} else {
-------if (condition_i) {
--------if (condition_f) {
---------goto Merge BG;
--------} else {
---------goto Merge BGI;
--------} //else f
-------} else {
--------goto Merge BG;
-------} //else i
------} //else d
-----} else {
------if (condition_i) {
-------if (condition_f) {
--------goto Assign B;
-------} else {
--------goto Merge BI;
-------} //else f
------} else {
-------goto Assign B;
------} //else i
-----} //else g
----} //else h
---} else {
----if (condition_d) {
-----if (condition_f) {
------if (condition_h) {
-------goto Assign D;
------} else {
-------goto Merge DF;
------} //else h
-----} else {
------if (condition_i) {
-------if (condition_h) {
--------if (condition_c) {
---------goto Merge CD;
--------} else {
---------if (condition_r) {
----------goto Merge DR;
---------} else {
----------goto Assign D;
---------} //else r
--------} //else c
-------} else {
--------if (condition_c) {
---------goto Merge CDI;
--------} else {
---------if (condition_r) {
----------goto Merge DIR;
---------} else {
----------goto Merge DI;
---------} //else r
--------} //else c
-------} //else h
------} else {
-------if (condition_c) {
--------goto Merge CD;
-------} else {
--------if (condition_r) {
---------goto Merge DR;
--------} else {
---------goto Assign D;
--------} //else r
-------} //else c
------} //else i
-----} //else f
----} else {
-----if (condition_h) {
------if (condition_a) {
-------if (condition_s) {
--------if (condition_c) {
---------if (condition_f) {
----------goto Assign H;
---------} else {
----------goto Merge CH;
---------} //else h
--------} else {
---------if (condition_r) {
----------if (condition_f) {
-----------goto Assign H;
----------} else {
-----------goto Merge HR;
----------} //else f
---------} //else f
--------} else {
---------if (condition_r) {
----------if (condition_f) {
-----------goto Assign H;
----------} else {
-----------goto Merge HR;
----------} //else f
---------} else {
----------goto Assign H;
---------} //else r
--------} //else c
-------} else {
--------if (condition_c) {
---------if (condition_f) {
----------goto Merge AH;
---------} else {
----------goto Merge ACH;
---------} //else f
--------} else {
---------if (condition_r) {
----------if (condition_f) {
-----------goto Merge AH;
----------} else {
-----------goto Merge AHR;
----------} //else f
---------} else {
----------goto Merge AH;
---------} //else r
--------} //else c
-------} //else s
------} else {
-------if (condition_p) {
--------if (condition_s) {
---------if (condition_c) {
----------if (condition_f) {
-----------goto Assign H;
----------} else {
-----------goto Merge CH;
----------} //else f
---------} else {
----------if (condition_r) {
-----------if (condition_f) {
------------goto Assign H;
-----------} else {
------------goto Merge HR;
-----------} //else f
```

```
---------} else {
----------goto Assign H;
---------} //else r
---------} //else c
--------} else {
---------if (condition_c) {
----------if (condition_f) {
----------goto Merge HP;
----------} else {
----------goto Merge CHP;
----------} //else f
---------} else {
----------if (condition_r) {
----------if (condition_f) {
----------goto Merge HP;
----------} else {
----------goto Merge HPR;
----------} //else f
----------} else {
----------goto Merge HP;
----------} //else r
---------} //else c
--------} //else s
-------} else {
--------if (condition_c) {
---------if (condition_f) {
----------goto Assign H;
---------} else {
----------goto Merge CH;
---------} //else f
--------} else {
---------if (condition_r) {
----------if (condition_f) {
----------goto Assign H;
----------} else {
----------goto Merge HR;
----------} //else f
---------} else {
----------goto Assign H;
---------} //else r
--------} //else c
-------} //else p
------} //else a
-----} else {
------if (condition_f) {
-------if (condition_s) {
--------goto Merge FS;
-------} else {
--------if (condition_g) {
---------if (condition_a) {
----------goto Merge AFG;
---------} else {
----------if (condition_p) {
----------goto Merge FGP;
----------} else {
----------goto Merge FG;
----------} //else p
---------} //else a
--------} else {
---------if (condition_a) {
----------goto Merge AF;
---------} else {
----------if (condition_p) {
----------goto Merge FP;
```

```
---------} else {
----------goto Assign F;
---------} //else p
---------} //else a
--------} //else g
-------} //else s
------} else {
-------if (condition_s) {
--------if (condition_i) {
---------if (condition_c) {
----------goto Merge CIS;
---------} else {
----------if (condition_r) {
----------goto Merge IRS;
----------} else {
----------goto Merge IS;
----------} //else r
---------} //else c
--------} else {
---------if (condition_c) {
----------goto Merge CS;
---------} else {
----------if (condition_r) {
----------goto Merge RS;
----------} else {
----------goto Assign S;
----------} //else r
---------} //else c
--------} //else i
-------} else {
--------if (condition_a) {
---------if (condition_c) {
----------if (condition_g) {
----------if (condition_i) {
----------goto Merge ACGI;
----------} else {
----------goto Merge ACG;
----------} //else i
---------} else {
----------if (condition_i) {
----------goto Merge ACI;
----------} else {
----------goto Merge AC;
----------} //else i
----------} //else g
---------} else {
----------if (condition_r) {
----------if (condition_g) {
----------if (condition_i) {
----------goto Merge AGIR;
----------} else {
----------goto Merge AGR;
----------} //else i
----------} else {
----------if (condition_i) {
----------goto Merge AIR;
----------} else {
----------goto Merge AR;
----------} //else i
----------} //else g
---------} else {
----------if (condition_g) {
----------if (condition_i) {
----------goto Merge AGI;
```

```
----------} else {
----------goto Merge AG;
----------} //else i
----------} else {
----------if (condition_i) {
----------goto Merge AI;
----------} else {
----------goto Assign A;
----------} //else i
----------} //else g
----------} //else r
---------} //else c
--------} else {
---------if (condition_c) {
----------if (condition_p) {
-----------if (condition_g) {
----------if (condition_i) {
----------goto Merge CGIP;
----------} else {
----------goto Merge CGP;
----------} //else i
----------} else {
----------if (condition_i) {
----------goto Merge CIP;
----------} else {
----------goto Merge CP;
----------} //else i
----------} //else g
----------} else {
----------if (condition_g) {
----------if (condition_i) {
----------goto Merge CGI;
----------} else {
----------goto Merge CG;
----------} //else i
----------} else {
----------if (condition_i) {
----------goto Merge CI;
----------} else {
----------goto Assign C;
----------} //else i
----------} //else g
---------} else {
----------if (condition_g) {
----------if (condition_i) {
----------if (condition_p) {
----------if (condition_r) {
----------goto Merge GIPR;
----------} else {
----------goto Merge GIP;
----------} //else r
----------} else {
----------if (condition_r) {
----------goto Merge GIR;
----------} else {
----------goto Merge GI;
----------} //else r
----------} //else p
----------} else {
----------if (condition_p) {
----------if (condition_r) {
----------goto Merge GPR;
----------} else {
```

```
----------goto Merge GP;
----------} //else r
----------} else {
----------if (condition_r) {
----------goto Merge GR;
----------} else {
----------goto Assign G;
----------} //else r
----------} //else p
----------} //else i
----------} else {
----------if (condition_i) {
----------if (condition_p) {
----------if (condition_r) {
----------goto Merge IPR;
----------} else {
----------goto Merge IP;
----------} //else r
----------} else {
----------if (condition_r) {
----------goto Merge IR;
----------} else {
----------goto Assign I;
----------} //else r
----------} //else p
----------} else {
----------if (condition_p) {
----------if (condition_r) {
----------goto Merge PR;
----------} else {
----------goto Assign P;
----------} //else condition_r
----------} else {
----------if (condition_r) {
----------goto Assign R;
----------} else {
----------goto action_NLB;
----------} //else condition_r
----------} //else condition_p
----------} //else condition_i
---------} //else condition_g
---------} //else condition_c
--------} //else condition_a
-------} //else condition_s
------} //else condition_f
-----} //else condition_h
----} //else condition_d
---} //else condition_b
--} //else condition_q
-} //else condition_e
} else {
-goto action_NO;
}
```

IV. EXPERIMENTAL RESULTS

The experiment was performed on Ubuntu 12.04 OS with an Intel® Core™ i5-2450M Processor, 2.50 GHz, 4 cores, using a single core for the processing. All algorithms used for our comparison were implemented in C++ using OpenCV library, the compiler is g++. All experimental results presented in this section were obtained by averaging the execution time for 100 runs. To prevent one run from filling the cache to make subsequent runs faster, we deallocated the image header and the image data at the end of each run by calling the standard function (cvReleaseImage()) of OpenCV. On the other hand, all algorithms produced the same number of labels and the same labeling on all images.

I used the synthetic dataset of black and white random noise square images with seven different image sizes from a low resolution of 32x32x10 pixels to a maximum resolution of 2048x2048x10 pixels collected from [3]. The experimental results show that the proposed method consumes nearly the same computation time as He *et al*. [16] for all image sizes as shown in Table III.

TABLE III. PERFORMANCE OF EACH ALGORITHM WITH VARYING SIZE OF THE IMAGE.

Image size	Time (ms)	
	He et al.	DT
32x32x10	0.465	0.503
64x64x10	1.583	1.619
128x128x10	5.144	5.227
256x256x10	19.615	19.658
512x512x10	77.636	78.014
1024x1024x10	306.139	307.581
2048x2048x10	1226.077	1230.73

V. CONCLUSION

Decision trees are commonly used as a predictive model in statistics, data mining and machine learning. In digital image processing, this technique is also used in 2-D connected components labeling algorithm [6]. Therefore, from the popular and simple aspect of this method, this paper presents the decision tree model for 3-D connected components labeling algorithm of binary image. The result indicates that this model spends a short processing time just a little bit more than the fastest method [16]. Although this approach does not provide the fastest processing time, it can be considered as an interesting technique

REFERENCES

[1] A. AbuBaker, R. Qahwaji, S. Ipson, and M. Saleh, "One scan connected component labeling technique," *IEEE International Conference on Signal Processing and Communications (ICSPC 2007)*, Dubai, United Arab Emirates, 2007, pp. 1283-1286.

[2] J. Trein, A. T. Schwarzbacher, and B. Hoppe, "FPGA implementation of a single pass real-time blob analysis using run length encoding," *MPC-Workshop, Ravensburg-Weingarten*, Germany, 2008, pp. 71-77.

[3] C. Grana, D. Borghesani and R. Cucchiara, "Optimized block-based connected components labeling with decision trees," *IEEE Transactions on Image Processing*, vol. 19, issue 6, 2010.

[4] A. Rosenfeld and J. L. Pfaltz, "Sequential operations in digital picture processing," *J. ACM*, vol. 13, no. 4, pp. 471–494, 1966

[5] A. Rosenfeld, A.C. Kak, "Digital Picture Processing," *Academic Press*, second ed., vol. 2, San Diego, CA, 1982.

[6] K. Wu, E. Otoo and A. Shoshani, "Optimizing Connected Component Labeling Algorithms," *Proceedings of the SPIE*, vol. 5747, pp. 1965-1976, 2005.

[7] L. He, Y. Chao, K. Suzuki, and K. Wu, "Fast connected-component labeling," *Pattern Recog*, vol. 42, no. 9, Sep. 2009.

[8] L. He, Y. Chao, and K. Suzuki, "An Efficient First-Scan Method for Label-Equivalence-Based Labeling Algorithms," *Pattern Recognition Letters*, vol. 31, 2010, pp. 28-35

[9] L. He, Y. Chao, and K. Suzuki, "A Run-based Two-Scan Labeling Algorithm," *IEEE Transactions on Image Processing*, vol.17, No. 5, pp. 749-756, 2008

[10] L. He, Y. Chao, and K. Suzuki, "A Run-Based One-and-a-Half-Scan Connected-Component Labeling Algorithm," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 24, No. 4, 2010, pp. 557-579.

[11] L. He, Y. Chao, and K. Suzuki, "A linear-time two-scan labeling algorithm," *2007 IEEE International Conference on Image Processing (ICIP)*, San Antonio, Texas, USA, September 2007, pp. V-241-V-244.

[12] K. Suzuki, I. Horiba, and N. Sugie, "Linear-time connected-component labeling based on sequential local operations," *Comput Vision Image Understand*, vol. 89, pp. 1–23, 2003.

[13] F. Chang, C.J. Chen and C.J. Lu, "A linear-time component-labeling algorithm using contour tracing technique," *Computer Vision and Image Understanding*, vol. 93, 2004.

[14] 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, 1983.

[15] Y. Shirai, "Labeling connected regions," in *Three-Dimensional Computer Vision*. New York: Springer-Verlag, pp. 86–89, 1987.

[16] He, L. Chao, Y. Suzuki, K. "Two Efficient Label-Equivalence-Based Connected-Component Labeling Algorithms for 3-D Binary Images," *IEEE Transactions on Image Processing,* Vol. 20 No. 8. August, 2011.