A Component-Labeling Algorithm Using Contour Tracing Technique

Fu Chang and Chun-Jen Chen
Institute of Information Science, Academia Sinica, Taipei, Taiwan, ROC
E-mail: fchang@iis.sinica.edu.tw, dean@iis.sinica.edu.tw

Abstract

A new method for finding connected components from binary images is presented in this article. The main step of this method is to use a contour tracing technique to detect component contours and also to fill in interior areas. All the component points are traced by this algorithm in a single pass and are assigned either a new label or the same label as their neighboring pixels. Experimenting on various types of document images (characters, pictures, newspapers, etc.), we find that our method outperforms the other sequential methods using the equivalence technique. Our algorithm, moreover, is a method that not only labels components but also extracts component contours at the same time, which proves to be more useful than those algorithms that only label components.

1. Introduction

The objective of document image analysis is to detect and identify various types of objects, for example, halftone pictures, content characters, headline characters, and higher-order entities such as textlines, paragraphs, articles, etc. For this purpose, the detection and labeling of connected components (in short, components) is one of the most basic and commonly used techniques. The purpose of a component-labeling algorithm is to find all the components in the image and label them with proper indices [1-3]. For the rest of this article, we assume that each pixel in a binary image is represented by either 1 (black) or 0 (white).

Several sequential [4-6] and parallel [7-9] labeling algorithms are proposed in the literature. The parallel algorithms are designed for parallel processing devices and are therefore not comparable with ours that is designed for use in sequential processing devices. One of the sequential methods is the algorithm proposed by Rosenfeld and Pfaltz [4] that performs two passes over an input image. In the first pass, each pixel is scanned once. At each black pixel *P*, a further examination of its four neighboring (left, upper left, top, and upper right) pixels is also conducted. If none of these neighbors carries a nonzero label, *P* is assigned a new label. Otherwise, the nonzero labels carried by the neighbors of *P* are said to be equivalent, and *P* will be assigned the minimal equivalent label. In the first pass, a pair of arrays is also generated, one containing the current labels that need to be substituted and the other their minimal equivalent labels. In the second pass, label substitutions are made.

The method, proposed by Haralick [5], is designed for removing the extra storage requirement for the pair of arrays proposed in the first algorithm. In the beginning, each black pixel is given a unique label. Then, the labeled image is processed in two passes iteratively. During the first pass, which is a top to bottom pass, each labeled pixel is reassigned the smallest label from the label set including the pixel's own label and its neighboring nonzero labels. The second pass is a bottom to top pass that is similar to the first one. The two passes are conducted iteratively until no label change occurs. The memory storage of this method is small, but the overall processing time can vary widely according to the nature of the image being processed.

Lumia, Shapiro, and Zuniga [6] suggest another method that is a compromise between the two previous methods. In the first pass, which is a top to bottom pass, labels are assigned to black pixels as in the first method. At the end of each scan line, however, the nonzero labels on this line are changed to their minimal equivalent labels. The second pass works the same as the first pass, except that the working direction is from bottom to top. Note that the three methods just described have become the compared targets for other proposed methods.

In this paper we propose a very effective algorithm for detecting and labeling components in a binary image. Our method uses a contour tracing technique, which improves the methods [10,11], to detect and label the internal and external contours associated with each component and to label interior areas of components in the same pass of the image. The improved contour tracing technique not only labels component contours, but also provides additional information about the components such as area size, number of holes, and the positions of contour points. The extra information can be useful in computer vision and pattern recognition applications.

The detail of this method is described in Section 2. The experimental results of our method as compared with the other three algorithms and the applications of our method are discussed in Section 3. In Section 4, a brief conclusion is given.

2. The proposed algorithm

The idea of this algorithm is to scan the image from left to right and from top to bottom. When an *unlabeled* external contour point A is encountered, we make a complete trace of the contour until we get back to A. We also label A and all contour points with a new index (Figure 1a). On the other hand, when a *labeled* external contour point is encountered, we follow the scan line to find the subsequent black pixels and assign the same label to them until an internal contour point B is encountered the first time (Figure 1b). At B, we make a complete trace of the contour and assign the same label to all the points (Figure 1c). Later on, when an internal contour point is encountered again, we follow the scan line to find the subsequent black pixels and assign to them the same label as the contour point (Figure 1d).

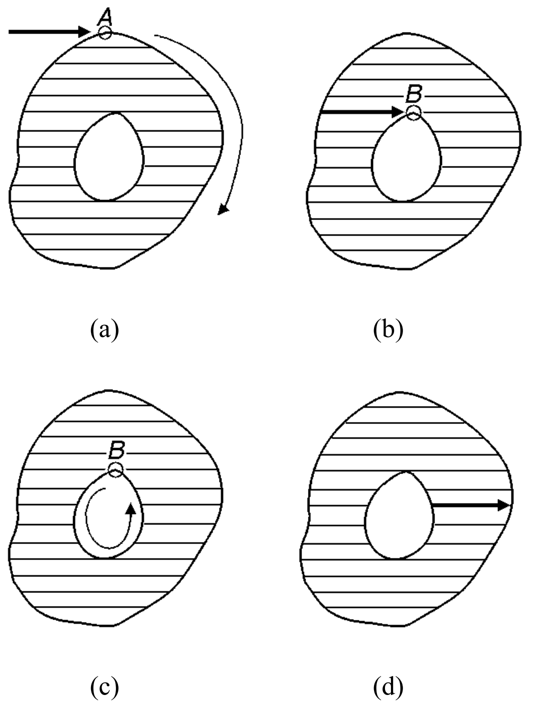

Figure 1. The four major steps in tracing the points in a component.

In doing so, we visit all the component points in a single pass over the image and label them according to a new index or according to the one that has already been assigned to its left neighbor. The detail of this algorithm is now given as follows.

For a given document image I, we associate with it an accompanying image L, in which we are going to store the labels assigned to all the components. At the beginning of the algorithm, all pixels of L are initialized to 0. We then start to scan I from left to right and from top to bottom to find any black pixel, denoted as P henceforth. Let C be the labeling index for components. At the beginning, C is set to be 1.

We now examine the following three cases in the order as stated.

Case 1: P is unlabeled and its left neighbor is a white pixel (Figure 2). In this case, P must be an external contour point, and we execute *Contour Tracing* to find the contour containing P. We then assign label C to all the contour pixels and increase the value of C by 1.

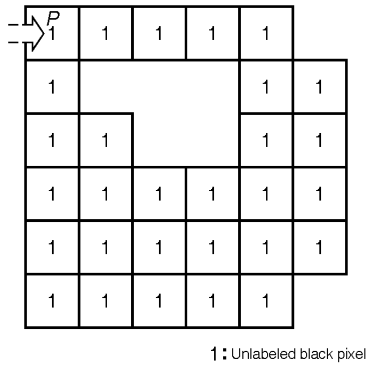

Figure 2. P is the starting point of an external contour.

Case 2: The right neighbor of P is an unlabeled white pixel. In this case, P must be an internal contour pixel, and we also execute *Contour Tracing* to find the contour containing P.

There are two possibilities in this case. First, P is not labeled (Figure 3). In this situation, the left neighbor of P must have been labeled. We then assign to P and all contour points the same label as the left neighbor of P.

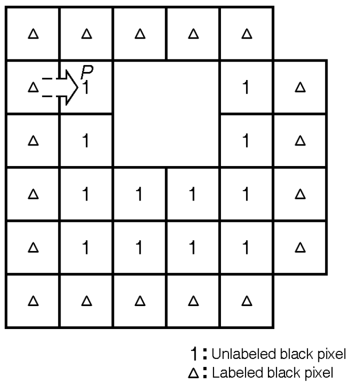

Figure 3. P is the starting point of an internal contour.

The second possibility is that P is already labeled (Figure 4). In this situation, we assign to all contour points the same label as P.

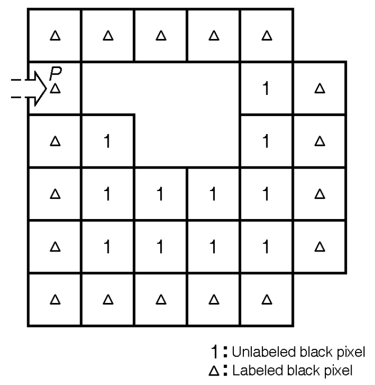

Figure 4. *P* is the starting point of an internal contour and is also a point in an external contour that has already been labeled.

Note that we have to differentiate pixel *P* from other black pixels, for example, *Q* in Figure 5, whose right neighbor is also a white pixel. In order to avoid executing *Contour Tracing* at *Q*, we label the surrounding pixels of a component with a negative integer, whenever we have traced a contour of it. Thus, at the time *Q* is swept by the scan line, the right neighbor of *Q* is no longer an unlabeled white pixel. The right neighbor of *P*, on the other hand, is still unlabeled, since the contour containing *P* has not been traced at the time *P* is encountered.

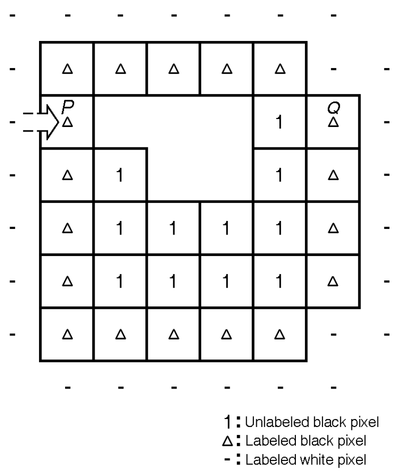

Figure 5. All exterior surrounding pixels are labeled with a native integer when the external contour has been traced.

By labeling the surrounding pixels of a component, we can also avoid tracing the same internal contour from any contour point that has already been traced. Thus, as illustrated in Figure 6, when *R* is traced, the surrounding pixels of *R* are now labeled with a negative integer and the right neighbor *R* is no longer an unlabeled white pixel. Therefore, case 2 does not apply to *R*. We thus avoid tracing the internal contour from *R* again.

The operation of labeling all the surrounding pixels with a negative integer is included in *Contour Tracing*, as will be explained later.

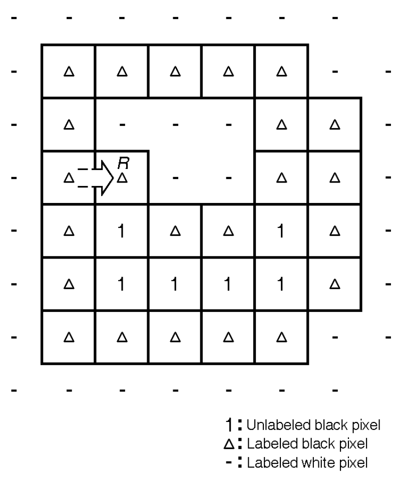

Figure 6. All the interior surrounding pixels are labeled with a negative integer when the internal contour has been traced.

Case 3: None of the above. In this case, the left neighbor of *P* must have been labeled already (Figure 7). We shall then assign to *P* the same label as its left neighbor.

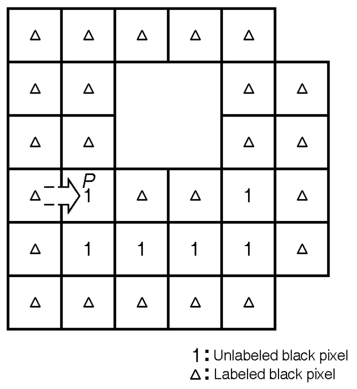

Figure 7. *P* is an unlabeled pixel but its left neighbor has already been labeled.

The following two sub-procedures, *Contour Tracing* and *Tracer*, are used in the main procedure described in the above.

2.1 Contour tracing

The goal of *Contour Tracing* is to find an external or internal contour at a given pixel, named *S*. At this pixel, we first execute *Tracer.* If *Tracer* identifies *S* as an isolated pixel, we reach the end of *Contour Tracing*. Otherwise, *Tracer* will output the next contour point of *S*. Let us name this point *T*. We then continue to execute *Tracer* to find the next contour point of *T*, then its next point, etc. until the following two conditions hold: (i) *Tracer* outputs *S* again, and (ii) the next contour point of *S* is *T* again.

Note that the procedure would stop only when both of the above conditions hold. If *S* is hit again while the next contour point of *S* is not *T* but *V* (Figure 8), we shall continue the procedure. Thus, as shown in Figure 8, when *S* is the starting point and *T* is the next contour point, the path traced by *Tracer* is *STUTSVWVS.*

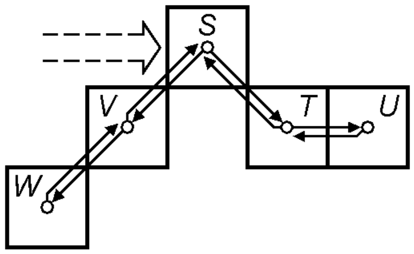

Figure 8. Tracing the contour of a stripe-shaped component.

2.2 Tracer

The goal of *Tracer* is to search for the next contour point from the current point *P*. The tracing is controlled by the initial search direction, which is determined by the position of a previous contour point.

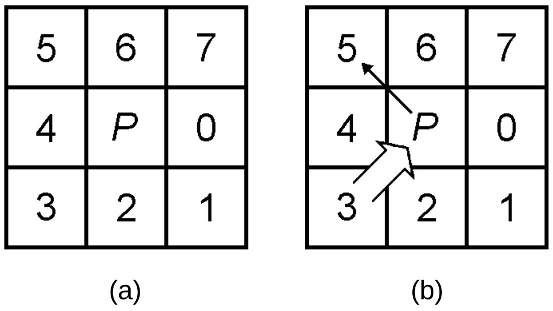

Figure 9. (a) The neighboring pixels of *P* are indexed by their direction relative to *P*. (b) If the previous contour point lies at 3, the next search direction is set at 5.

For a given *P*, the positions relative to *P* are labeled by indices from 0 to 7, as shown in Figure 9a. If *P* is the starting point of an external contour, the initial search position is set to be 0 (right), since all the pixels left to *P* or above *P* does not belong to this contour. If, however, *P* is the starting point of an internal contour, the initial search position is set to be 1 (lower right), since its right neighbor is known to be white.

On the other hand, when there is a previous contour point and it lies at position 3 (lower left), the initial search position is set to be 5, since the pixel at position 4 must have been visited already (Figure 9b), irrespective whether the contour is internal or external. In general, when *P* is not a starting point of a contour, its initial search position is set to be *d*+2 (mod 8), where *d* is the position of the previous contour point.

Once the initial search position is determined, we proceed in a clockwise direction to look for the first black pixel. This pixel is taken as the output of *Tracer*. If no black pixel is found in the whole circle, *P* is identified as an isolated pixel.

When finding the next contour pixel from a given contour point, we can also label (with a negative index) the surrounding pixels of the component containing *P*. As illustrated in Figure 10, *A* is the current point and *C* is the next contour point. Along the pathway from *A* to *C*, we can label the encountered *B* with the negative index.

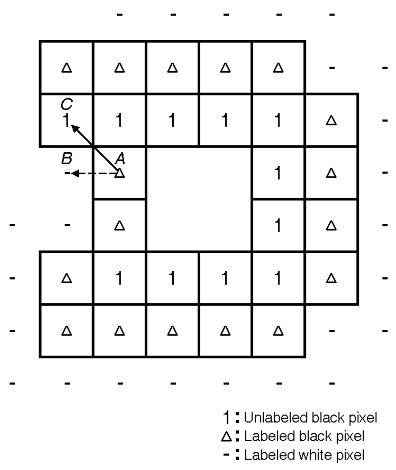

Figure 10. While tracing the external or internal contour, we are also labeling the surrounding pixels.

3. Experimental results

3.1 Processing time

We made a comparison between our method and three other component-labeling algorithms listed in Table 1. Several types of documents or excerpts from them, including legacy documents, newspapers, halftone pictures, headlines, and textual contents, were used for the testing purpose, while the testing environment is an Intel Pentium III 1GHz personal computer with 384MB SDRAM. The comparison results are listed in Table 2.

The results show that algorithm #2, algorithm #3, and our method grow linearly with respect to the number of components. However, the processing time of the first two algorithms are multiples of our processing time.

Table 1. Four types of algorithms, including ours, that are being compared.

Algorithm #1	Rosenfeld and Pfaltz [4]
Algorithm #2	Haralick [5]
Algorithm #3	Lumia, Shapiro, and Zuniga [6]
Algorithm #4	Our algorithm

Table 2. Comparisons among all the four algorithms.

Type	Size (pixels)	CC#	Algorithm #1	#2	#3	#4
			Processing time (sec)			
Legacy	15.26M	2903	39.26	4.63	3.52	0.39
	15.27M	3430	53.55	5.13	3.70	0.38
Headline	11.57M	1235	25.62	3.38	2.82	0.32
	15.44M	1595	29.72	4.26	6.12	0.47
Content	25.31M	15511	1348.39	7.87	19.90	0.93
	24.76M	16141	1801.95	9.31	25.41	0.97
Halftone	22.65M	17536	3590.49	19.83	11.65	0.60
	23.06M	86171	6437.57	25.46	42.58	1.10
News	27.99M	22777	4412.76	63.79	43.32	1.16
	33.48M	31249	3073.32	56.59	36.52	1.21
CC#: The number of the connected components						

3.2 Application of component labeling using contour tracing technique

A method that extracts contour and labels components at the same time can find potential applications in any areas in which we have to detect components and also to recognize components or collections of components by means of certain contour features. Document analysis and recognition, in particular, is a place to which this method applies. Components are the objects that compose all high-order textual objects such as characters, textlines, and text regions [12]. When those objects have been identified, there is also a need to recognize characters within them. For this purpose, there are many methods that employ certain contour features for identifying characters [13-15]. In one particular application, we use contour information to generate skeletons for characters [13]. The skeletons can be used in the recognition tasks that rely upon the analysis of the skeleton structure [16].

4. Conclusion

We have described a new component-labeling algorithm that employs the contour tracing technique. This technique is very useful in labeling the components and extracting the contour features of components on the image. We experiment with our method and compare it with three other sequential algorithms. The results show that our algorithm is much more efficient in terms of the computational speed.

5. References

[1] C. Ronse and P. A. Devijver, *Connected components in binary images: the detection problem*, Research Studies Press, NY: Wiley, 1984.

[2] R. Kasturi and M. M. Trivedi, Mohan, *Image analysis applications*, NY: M. Dekker, 1990.

[3] R. C. Gonzalez and R. E. Woods, *Digital Image Processing, Reading*, MA: Addison-Wesley, 1992.

[4] A. Rosenfeld and P. Pfaltz. "Sequential Operations in Digital Picture Processing," *Journal of the Association for Computing Machinery*, vol. 12, 1966, pp. 471-494.

[5] R. M. Haralick, *Some neighborhood operations, in Real Time/Parallel Computing Image Analysis* (M. Onoe, K. Preston, and A. Rosenfeld, Eds.), Plenum Press, New York, 1981.

[6] R. Lumia, L. Shapiro, and O. Zuniga, "A New Connected Components Algorithm for Virtual Memory Computers," *Computer Vision, Graphics, and Image Processing*, vol. 22, 1983, pp. 287-300.

[7] D. Nassimi and S. Sahni, "Finding Connected Components and Connected Ones on a Mesh-Connected Parallel Computer," *SIAM Journal of Computing*, vol. 9, no. 4, 1980, pp. 744-757.

[8] Y. Shiloach and U. Vishkin, "An O(logN) Parallel Connectivity Algorithm," *Journal of Algorithms*, vol. 3, 1982, pp. 57-67.

[9] M. Manohar and H. K. Ramapriyan, "Connected Component Labeling of Binary Images on a Mesh Connected Massively Parallel Processor," *Computer Vision, Graphics, and Image Processing*, vol. 45, 1989, pp. 133-149.

[10] H. Freeman, "Techniques for the Digital Computer Analysis of Chain-Encoded Arbitrary Plane Curves," *Proc. Nat. Electronics Conf.*, vol. 17, 1961, pp. 412–432.

[11] T. Pavlidis, *Algorithms for graphics and image processing, Computer Science Press*, MD: Rockville, 1982.

[12] F. Chang, "Retrieving Information from Document Images: Problems and Solutions," *International Journal on Document Analysis and Recognition, Special Issues on Document Analysis for Office Systems*, vol. 4, no. 1, 2001, pp. 46-55.

[13] F. Chang, Y. C. Lu, and T. Pavlidis, "Feature Analysis Using Line Sweep Thinning Algorithm," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 21, no. 2, 1999, pp. 145-158.

[14] A. K. Jain, R. P. W. Duin, and J. Mao, "Statistical Pattern Recognition: A Review," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 1, 2000, pp. 4-37.

[15] Ø. D. Trier, A. K. Jain, and T. Taxt, "Feature Extraction Methods for Character Recognition – A Survey," *Pattern Recognition*, vol. 29, no. 4, 1996, pp. 641-662.

[16] L. Lam, S. W. Lee, and C. Y. Suen, "Thinning Methodology - A Comprehensive Survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, 1992, pp. 869-885.