ICIC Express Letters
Part B: Applications
Volume **7**, Number **12**, December **2016**

ICIC International ©2016 ISSN 2185-2766
pp. **2625–2632**

A NOVEL AND FAST CONNECTED COMPONENT-COUNTING ALGORITHM BASED ON GRAPH THEORY

Sihai Yang[1], Duansheng Chen[1], Xianhua Han[2] and Yenwei Chen[2]

[1]College of Computer Science and Technology
Huaqiao University
No. 668, Jimei Avenue, Xiamen 361021, P. R. China
{ shyang; dschen }@hqu.edu.cn

[2]College of Information Science and Engineering
Ritsumeikan University
1-1-1 Noji-Higashi, Kusatsu, Shiga 525-1000, Japan
hanxhua@fc.ritsumei.ac.jp; chen@is.ritsumei.ac.jp

Received June 2016; accepted September 2016

Abstract. *A fast component-counting algorithm is proposed based on graph theory in this paper. Vertices with degree no more than two are ineffective in counting components. From the intuitive viewpoint we theoretically analyzed the counting of faces in a graph based on Euler polyhedron formula and derived a formulation to count faces in a plane given only the vertices. With the derived formula, a component-counting algorithm is constructed based only on searching cross points whose degree is no less than three. When applied to a two-dimensional binary image, the proposed method divides an image into patches of the same size and decides which of them will be used in counting by searching the circumferential pixels of each patch. If the number of component edges within the circumferential pixels of a patch is not less than three, the patch will be used in counting. On the contrary, patches without matching the criteria will be discarded. After filtering all the vertices with degree no less than three, the number of components can be calculated by the formula. Because only a small number of pixels are investigated in the process, the computational speed is very fast. The disconnection of edges in an image is one of the main reasons which cause miscounts for scanning-based algorithms. The difficulty, however, can be naturally overcome by the proposed algorithm because disconnected points will be identified as futile pixels in the algorithm. Experimental results show the algorithm is more efficient than existing methods. When applied to images with disconnected edges, the counted number given by scanning-based algorithms is much smaller than the correct number whereas the proposed algorithm obtains satisfactory results.*
Keywords: Connected component labeling, Branch degree, Polyhedron formula, Connected component counting, Graph theory

1. **Introduction.** In image analysis and computer vision, one of the most fundamental tasks is to count the number of objects in an image and identify their shapes. As objects are normally characterized by connected pixels with different intensities from the background in an image, connected component labeling algorithms are often carried out to fulfill the task.

There have been many applications for counting the number of objects in an image. For example, hydrologists usually make use of the average grain size of sand taken from river beds to evaluate the corresponding parameters of hydrodynamics. Metallographists often use the average grain size of metal particles to judge the performance of the heat-treated metal. To determine the average grain size of particles, we only need to count the number of objects in a binary image which is derived from a photo by image preprocessing techniques given the size of lens or photos. Most of component counting algorithms are based on existing connected component labeling algorithms.

When counting isolated simple components, two primary morphological operators perform quite well [1]. However, when objects appear with graphic structures, usually it is needed to scan each pixel to count the number of components exactly. Researchers have proposed various algorithms to address the problem. In practice, these algorithms can be classified into three categories: (1) label-equivalence-resolving algorithms; (2) label-propagating algorithms; (3) contour-tracing algorithms.

The basic idea of label-equivalence-resolving algorithms is to scan each pixel in a binary image and label objects with different integers, and then merge equivalence integers based on connectivity. This type of algorithms is one of the most popular methods and there are various forms. For example, one method is the two-pass algorithm [2-4]. Haralick [5] and Suzuki et al. [6] used morphological operators and one dimensional table to conduct the operation. In addition, Samet applied quadtrees to merge the equivalent labels [7] and He et al. achieved the same goal by merging the label sets connected to a new run [8,9]. Usually, when applying these algorithms and determining the final labels for different objects, we need a label-revising step or completion.

It seems that label-propagating algorithms are similar to label-equivalence-resolving algorithms. However, these algorithms emphasize on the propagation of a label among the pixels belonging to an identical object. Recursive searching algorithm [1] is a typical one in this category. Some algorithms based on depth-first or greedy strategy can also be classified into this category. Intuitively, these algorithms can be easily understood as it is similar to human's counting process. Nevertheless, they require intensive computation when objects have intricately geometric structure, which makes it difficult to apply in practice.

Contour-tracing algorithms can be viewed as a hybrid of the first two categories. In the paper given by Chang et al. [10], they designed a scheme to determine the label of objects through two alternating steps. In one step, the algorithm traces the contour of a newly found object; while in the other step, it scans the image and labels pixels based on traced contours.

In this paper, we focus on component counting algorithms which are based on graph theory and are different from the aforementioned algorithms. For simplicity, we only consider binary images which can be viewed as pre-processed images, but the proposed algorithm can also be applied to gray images. The proposed algorithm has the approximate solution although theoretically it has the potential to obtain exact counting numbers.

There are three main contributions of our paper. (1) We focus on a rarely studied feature in literature-branch, and introduce the corresponding transformation and use a global indicator to quantify it. (2) A branch-detecting method based on patches is presented. (3) Two component counting algorithms are proposed based on branch-counting.

The rest of this paper is organized as follows. First, we explain the derivation of the counting formula and illustrate the proof in Section 2. Second, in Section 3, we describe the two proposed algorithms in detail and analyze their efficiency. Section 4 contains the experiment results. In the final section, we summarize key points.

2. **Problem Statement and Preliminaries.**

Counting the number of objects in an image is not an easy task. The reason is that to identify an object, an algorithm has to scan each pixel within the contour to make sure the entire object is identified. In other words, this characteristic of an object is global. However, as we know, ordinary operators when applied to processing images are in local scale, which means we need extra subsequent operations to achieve the final goal by combining the results derived from the local operators. Therefore, to skip over the pixel-scanning paradigm and maintain a fairly low level of algorithmic complexity, we must find a local characteristic which is related to the number of objects.

Euler proved a renowned formula:

$$F + V - E = 2. \tag{1}$$

For a convex polyhedron, F, V, E are the number of faces, vertices and edges respectively. When the object is a planer graph, the formula will be changed to

$$F + V - E = 1. \tag{2}$$

Figure 1 illustrates how the transformation is set up. For a convex polyhedron, if we choose a face as the under-surface, then press vertices above the under-surface down to it till these vertices coincide with the under-surface. As F, V, E do not change under the transformation, once we discard the under-surface, we can see how the equation is derived.

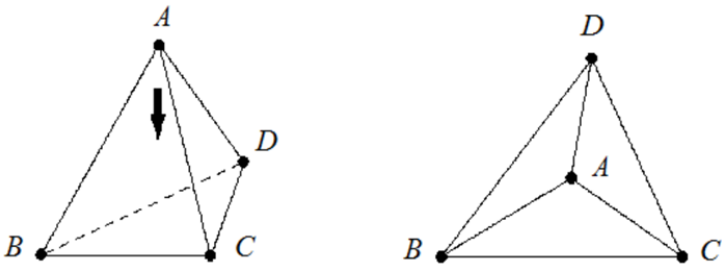

FIGURE 1. The convex polyhedron and the transformed planer graph

Although (2) is derived from (1) which is correct only for convex polyhedrons, it is valid for arbitrary connected planer graphs [11]. There are mainly two obstacles when applying (2) to transferring the study from F to V and E directly. First, edges are also global objects which cannot be identified by local operators straightforwardly. Second, the vertices with degree 2 are hard to identify in a binary image, especially when considering the possible deformation of objects (consider a shape-shifting curve). To overcome the two difficulties, we need to transfer the study the second time from vertices and edges to branches.

Classic graph theory chooses vertex and edge as elements to represent objects and their relationships in the real world. However, in image processing, when analyzing the invariant properties under deformations, the two concepts cannot meet our requirements. For example, the left graph in Figure 2 has 6 vertices and 7 edges. After a topological transformation, a graph may hold the same value of F while the values of V and E change. In image processing, this kind of deformations is quite common.

As the vertices and edges may change under topological transformations (if we see vertices as common points in a line), we should select a more robust feature. If we focus on branches in the left graph in Figure 2 and view the vertices with degree 2 as normal points on edges, we can find they are invariant under topological transformation. Another merit must be mentioned here is that these branches are local and naturally suited to operation in image processing. To state branch, we define the following notions:

Definition 2.1. *For a connected planer graph, the branch degree of a point is the value that subtracts 2 from its degree.*

FIGURE 2. The branch-holding transformation of a graph

Definition 2.2. *Point with nonzero branch degree is called vertex, and the branch degree of a vertex V_i is noted as:*

$$BD(V_i) = n, \quad n = -1, 1, 2, \ldots, \quad i = 1, \ldots, V. \tag{3}$$

When we indicate a planer graph or make a comparison among a group of graphs, determining a single value for each graph is convenient. The sum of branch degrees for all vertices in a graph is an ideal index. We denote it as Y here. Actually, it has a direct relationship to the value of F for a planer graph. We prove it here.

Theorem 2.1. *For a connected planer graph,*

$$F = \frac{1}{2}\sum_{i=1}^{V} BD(V_i) + 1 = \frac{Y}{2} + 1. \tag{4}$$

Proof: For a connected planer graph, in graph theory, we know the sum of degrees for all vertices equals twice of the number of edges,

$$2E = \sum_{i=1}^{V} \text{degree}(V_i). \tag{5}$$

According to (2), we have

$$F = \frac{1}{2}\sum_{i=1}^{V} \text{degree}(V_i) - V + 1 = \frac{1}{2}\sum_{i=1}^{V} BD(V_i) + 1. \tag{6}$$

The above proof does not include vertices with branch degree of -1. Intuitively, we can image a line is attached to a planer graph, which causes a vertex with branch degree $+1$ and a branch degree -1 at the other end. Because adding the line does not change the number of faces, (4) still holds. Further observations show, if we choose Y as an indicator for a graph, we can decompose or merge vertices and hold Y unchanged as Figure 3 shows. Under this kind of branch-holding transformation, each connected graph can correspond to a unique standard form.

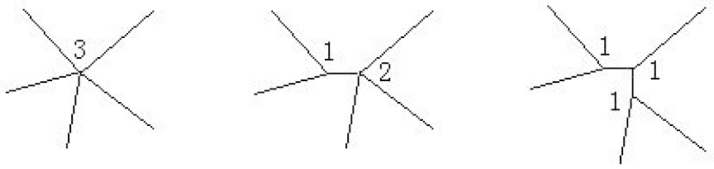

FIGURE 3. The decomposition of branch degree

If there is more than one graph, according to graph theory, (4) can be changed into (7) and it still holds. Here N_C represents the number of connected graphs.

$$F = \frac{Y}{2} + N_C. \tag{7}$$

3. **Algorithms.** Based on the analysis in Section 2, the problem of counting the number of components can be transferred to calculate the branch degrees Y in a binary image. We propose the following two methods to address the problem:

A. The algorithm based on tracing

A straightforward way to search branches is tracing along edges. Once we have an edge point in a binary image, we choose a neighborhood with size $d \times d$ and count the number N_{nb} of edge points along the circumference of its neighborhood. If the number N_{nb} is no less than 3, we compute the branch degree, set the inner points to background and push the edge points along the circumference onto the stack; otherwise we just set the

inner points to background and continue the tracing process until the stack is empty. We denote the algorithm as **A1** thereafter.

B. The algorithm based on blocking

Another way is to subdivide an image with a grid, and then count the number of edge points along the circumference of each patch (see gray points in Figure 4). We denote the algorithm as **A2** and illustrate the algorithm below:

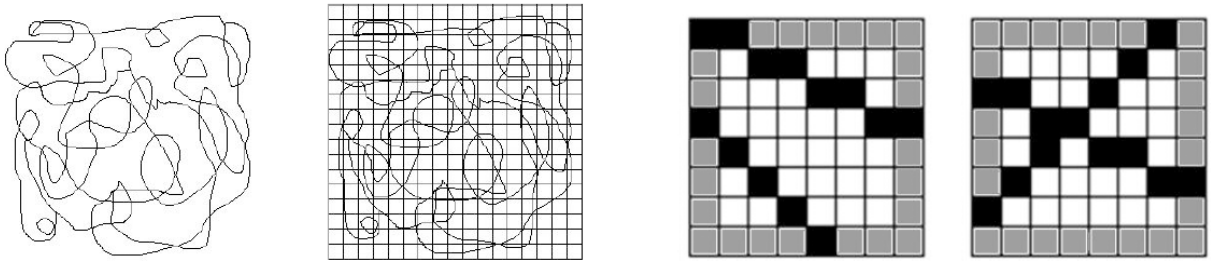

FIGURE 4. The grid searching method and a fake case. Left: a test graph; Center left: a test graph covered by grids; Center right: a fake case with branch degree 2; Right: a valid case with branch degree 2.

Algorithm A2: grid-searching algorithm

Step 1. Divide an image into patches with the same size.

Step 2. Search edge points along the circumference of each patch. If the number is -2, -1 or 0, we skip these patches; otherwise we compute its branch degree.

Step 3. Sum up branch degrees among selected patches and compute F by (6).

Several details on counting need to be mentioned here. First, successive edge points on the circumference of a patch will be counted as 1 (see the two top left pixels in Figure 4 the center right). Second, not all patches with the number of edge points no less than 3 include vertices. Figure 4 the center right shows a fake one. Therefore, we need to trace along edge points to check it for selected patches. Third, sometimes a grid line will cross a vertex and make the counted Y less than the correct value by 1. To remedy this error, for two adjacent patches with branch degree no less than 2, we should merge them together and check the circumference of the merged block.

For vertices with branch degree -1, if it is necessary to consider them in test images, we only need to count the patches with branch degree -1 while we can still discard patches with branch degree 0 or -2.

In some applications, we need to identify the contour of an object. For **A1**, if we store the traced edges, vertices and their connection relationship, the contour is easy to determine. For **A2**, since we only store the location of vertices, there is not a direct way to obtain contours. If we use the center of patches with branch degree 0 and connect them, we could get an approximate contour for an object.

4. **Experiments.** We choose the classic two-pass algorithm and the fast run-merging algorithm proposed in [8] to test the performance of the proposed algorithms. The algorithm given in [9] is representative too. However, its advantages are limited when objects appear with graphic structures. The run-merging algorithm is originally demonstrated for 8-neighborhood, but it can be applied to 4-neighborhood with small modification. Theoretically, it needs only one scanning and a merging operation to count the number of objects.

We conduct several experiments to test the performance of the proposed method in the following four aspects: (1) Do the two proposed algorithms consume less time than scanning-based algorithms? (2) Do the two proposed algorithms perform more robustly than scanning-based algorithms when there are disconnections in an image? (3) Are the

numbers counted by **A2** stable when the size of blocks changes within a proper range? (4) Does a larger block size consume less time for **A2**?

4.1. **The comparison of running time.** We use 26 typical test images with the size of 256×256, 512×512, 1024×1024, 2048×2048 pixels respectively to address the question (1). Figure 5 shows 6 of the test images with 256×256 pixels. Figure 6 is the comparison of the execution times among two-pass algorithm, run-merging algorithm, **A1** and **A2** with the block sizes of 5×5, 9×9, 13×13, 17×17, 21×21 pixels. We use the base 2 logarithm of the maximum, minimum and average execution time for comparison.

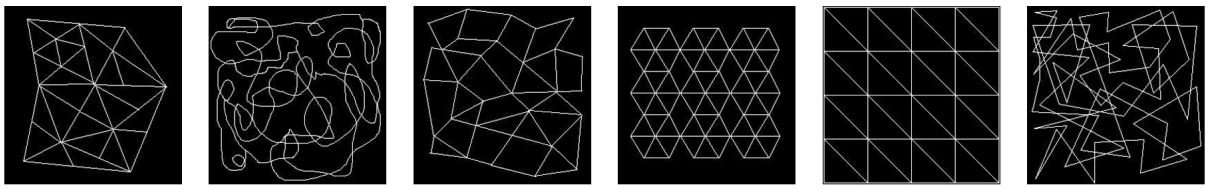

FIGURE 5. 6 test images with the size of 256×256 pixels

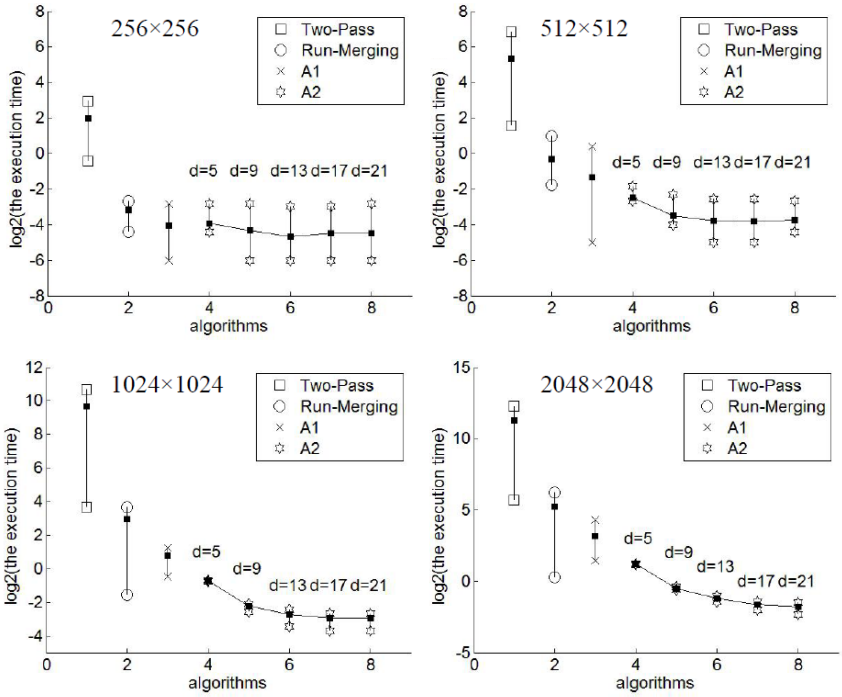

FIGURE 6. Execution time for test images with size of 256×256, 512×512, 1024×1024, 2048×2048 pixels. Algorithms include two-pass algorithm, run-merging algorithm, **A1** and **A2** with the block size as 5×5, 9×9, 13×13, 17×17, 21×21 pixels.

4.2. **The comparison of recall ratio.** To address the question (2), we randomly selected pixels in an image with probability p and set these pixels and their neighborhood (3×3 pixels) as background. Table 1 illustrates the counting number when p is 0.01 and 0.02 respectively. In Table 1, T is the correct number of components; S is the number given by scanning algorithms (both two-pass algorithm and run-merging algorithm get the same result); **A1** represents the number given by the algorithm **A1**. 5, 9, 13, 17 and 21 represent the number given by the algorithm **A2** when the block sizes are 5×5, 9×9, 13×13, 17×17 and 21×21. p is the probability to set a pixel and its 3×3 neighbor as background.

TABLE 1. The detected number of components in images with disconnected edges

T	$p = 0.01$							$p = 0.02$						
	S	A1	A2					S	A1	A2				
			5	9	13	17	21			5	9	13	17	21
28	14	28	22	24	23	21	22	11	24	21	22	22	21	21
50	29	46	42	47	43	45	44	21	39	43	48	39	42	40
27	17	25	26	27	25	25	24	6	21	23	22	20	20	21
59	42	57	59	59	57	56	54	26	49	52	51	46	47	43
33	22	33	27	26	32	25	31	4	5	26	24	28	20	23
77	54	75	70	73	70	67	70	42	6	67	70	64	63	60

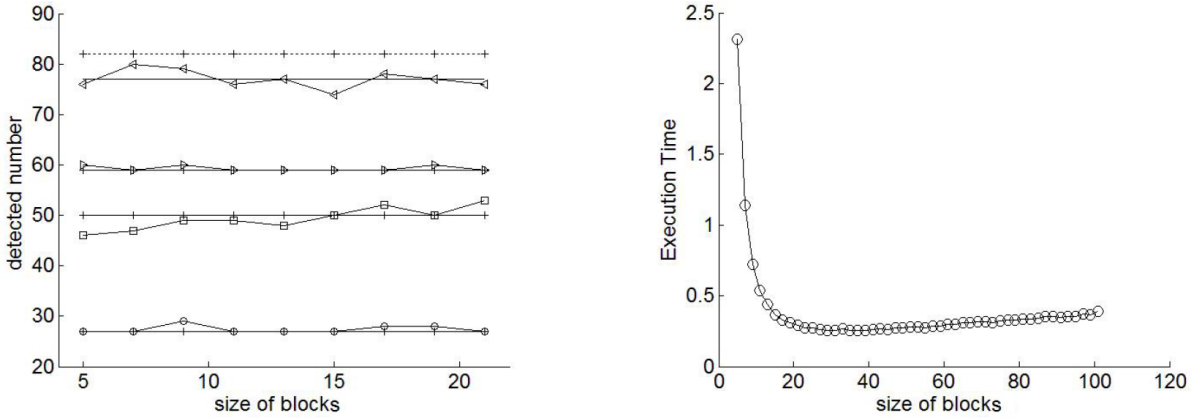

FIGURE 7. Left: the possible miscounts of **A1** and **A2**. Accurate values are shown by full lines; the dashed lines with "+" are given by **A1**; fluctuant lines are the results of **A2** with different block sizes. These are results from 4 images (shown in different symbols). Right: the corresponding changes of the execution times of **A2** as the block size increases for a 2048×2048 image.

4.3. **The robustness of A1 and A2.** For the problem (3), although we have identified several special cases causing miscounts, **A1** and **A2** sometimes can only obtain approximate results. However, we can see from the left graph in Figure 7, the results are fairly stable. For the question (4), it seems that a larger block size can take less execution time for **A2**. However, experimental results in the right graph of Figure 7 suggest the opposite evidence. The possible reason is that as the block size gets bigger the tracing and checking time for vertices in patches may get larger too.

5. **Conclusions.** In this paper, we illustrate a novel component counting approximate method applying to grain images based on Euler polyhedron formula. The proposed algorithms have higher efficiency and robustness than scanning-based methods. Although our method is approximate at this stage, the idea is novel and the feature we used is rarely discussed. The algorithm runs very fast and can be used as a pre-processed step. When applying to gray images, we only need to modify this step – counting the number of edge points along the circumference of each patch.

Besides the number of components, connected components labeling algorithms can also determine the shape and size of the components. How to develop the two proposed algorithms to achieve the same goal will be the key emphasis of study in the future.

Acknowledgment. This work was supported by the Grant-in Aid for Scientific Research from the Japanese Ministry for Education, Science, Culture and Sports under the Grant

No. 26330336, No. 15H01130, and No. 15K00253, in part by the Ministry of Education, Culture, Sports, Science and Technology of Japan (MEXT) Support Program for the Strategic Research Foundation at Private Universities (2013-2017) and in part by the National Bioscience Database Center (NBDC) of the Japan Science and Technology Agency (JST).

REFERENCES

[1] L. Shapiro and G. Stockman, *Computer Vision*, Prentice Hall, New Jersey, 2001.
[2] R. E. Tarjan, Efficiency of a good but not linear set union algorithm, *Journal of Association for Computing Machinery*, vol.22, no.2, pp.215-225, 1975.
[3] D. Santiago, T. Ren, G. Cavalcanti and T. Jyh, Efficient 2×2 block-based connected components labeling algorithms, *IEEE International Conference on Image Processing (ICIP)*, Quebec, Canada, pp.4818-4822, 2015.
[4] L. Cabaret, L. Lacassagne and L. Oudni, A review of world's fastest connected component labeling algorithms: Speed and energy estimation, *IEEE International Conference on Design and Architectures for Signal and Image Processing*, Madrid, Spain, 2014.
[5] R. H. Haralick, Some neighborhood operations, *Real Time/Parallel Computing Image Analysis*, Plenum Press, New York, 1981.
[6] K. Suzuki, I. Horiba and N. Sugie, Linear-time connected-component labeling based on sequential local operations, *Computer Vision and Image Understanding*, vol.89, no.1, pp.1-23, 2003.
[7] H. Samet, Connected component labeling using quadtrees, *Journal of the Association for Computing Machinery*, vol.28, no.3, pp.487-501, 1981.
[8] L. He, Y. Chao and K. Suzuki, A run-based two-scan labeling algorithm, *IEEE Trans. Image Processing*, vol.17, no.5, pp.749-756, 2008.
[9] L. He and Y. Chao, A very fast algorithm for simultaneously performing connected-component labeling and euler number computing, *IEEE Trans. Image Processing*, vol.24, no.9, pp.2725-2735, 2015.
[10] 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, no.2, pp.206-220, 2004.
[11] F. Harary, *Graph Theory*, Addison-Wesley, Menlo Park, 1969.