Turkish Journal of Electrical Engineering & Computer Sciences

http://journals.tubitak.gov.tr/elektrik/

Research Article

Turk J Elec Eng & Comp Sci
(2019) 27: 1243 – 1259

doi:10.3906/elk-1703-351

A fast and memory-efficient two-pass connected-component labeling algorithm for binary images

Bilal BATAINEH*
Department of Computer Science, Deanship of Common First Year, Umm Al-Qura University, Mecca, Saudi Arabia

Received: 29.03.2017 • **Accepted/Published Online:** 21.12.2018 • **Final Version:** 22.03.2019

Abstract: Connected-component labeling is an important process in image analysis and pattern recognition. It aims to deduct the connected components by giving a unique label value for each individual component. Many algorithms have been proposed, but they still face several problems such as slow execution time, falling in the pipeline, requiring a huge amount of memory with high resolution, being noisy, and giving irregular images. In this work, a fast and memory-efficient connected-component labeling algorithm for binary images is proposed. The proposed algorithm is based on a new run-base tracing method with a new resolving process to find the final equivalent label values. A set of experiments were conducted on different types of binary images. The proposed algorithm showed high performance compared to the other algorithms.

Key words: Binary images, CCL, connected-component labeling, image analysis, pattern recognition, segmentation

1. Introduction

Connected-component labeling (CCL) is an essential technique in computer visions, image analysis, and pattern recognition. It is used to give a unique value to all pixels that belong to individual components, usually in binary images [1–4]. CCL is required to segment the objects for many applications in robot vision, target tracking and recognition, traffic detection, vehicle license plate detection, video processing, automated surveillance and inspection, medical image analysis, computer-aided diagnosis, face and fingerprint identification, character recognition, and document analysis [4–7].

The most important features of any CCL algorithm are high speed and less memory requirement on random benchmark images [4,8–14]. In [8], Klaiber et al. claimed that the worst case scenario of CCL algorithms requires a huge amount of memory based on the image size. Based on [9], the main objective in CCL research is to propose a faster algorithm in several computer architectures, with a minimum amount of memory used, labels created, and complexity. He et al. mentioned in several works such as [5,10,11] that faster CCL algorithms are often required especially in real-time dynamic images, complicated geometric shapes, and complex connectivity applications.

Based on the state of the art, many CCL algorithms have been proposed; however, most of them have slow implementation, require huge memory, and their performance is not stable on different benchmark images, especially irregular and high-resolution images [8,12,15]. The two-pass algorithms are the most popular for CCL [5,9–12,16–18]. These algorithms are fast and affect all image types, but they usually fall in the pipeline

*Correspondence: bmbataineh@uqu.edu.sa

and have the problem of requiring large memory especially in large, high resolution irregular images. So far, no CCL algorithm has been able to contain all the advantages of fast execution time, small numbers of testing rules, and small amount of required buffering memory.

The main objective of the present work is to propose a simple, fast, and memory-efficient two-pass CCL algorithm for binary images. To achieve that, two principles are considered. Firstly, a new run-based technique for the first-pass stage is proposed. It is fast and extracts the minimum possible numbers of provisional values. Secondly, a new resolving process to find the final equivalent label values is proposed. It adopts a one-dimensional array rather than a two-dimensional array like most two-pass CCL algorithms, and resolves equivalent labels during the first-pass step to save time and memory. This paper is organized as follows: Section 2 presents the related works. Section 3 shows the proposed algorithm in detail. Section 4 presents the experiments and results, and the conclusions are presented the last section.

2. Related works

Many CCL algorithms have been proposed and they can be classified into three main approaches: one-pass, contour tracing, and raster-scan approaches.

2.1. One-pass CCL

One-pass or single-pass algorithms [19–21] scan the image once and give a new label value to the unlabelled pixel that is found. Then all the connected pixels are searched for and assigned the same value. This process is repeated until all the pixels get a label value. This type of algorithm in this approach is memory efficient [15]. However, they are rarely used because their implementation time is extremely slow, especially for irregular shapes.

2.2. Contour-tracing CCL

Contour-tracing algorithms [22–24] scan the image to find the contour pixels and give a new label value to the unlabeled contour pixel found. Then all the connected contour pixels are traced and given the same value. This process is repeated until all the contours pixels get a label value. After this, the pixels inside the contour get the same label value of the contour. These algorithms are average in memory requirement [15]. However, the main problem of these algorithms is the extremely slow implementation time for irregular images.

2.3. Raster-scan CCL

Raster-scan algorithms assign label values to pixels based on the neighboring label values or they assign a new label value if there are no neighboring label values. The algorithms of this approach are classified into multiscan and two-pass algorithms.

2.3.1. Multiscan algorithms

This class of CCL algorithms generate label values by scanning the image in an opposite direction recursively until no more label changes occur [25–27]. These algorithms do not require any auxiliary memory [15]. However, the processing time of multipass algorithms depends strongly on the shape structures and usually these algorithms are slow.

2.3.2. Two-pass algorithms

Two-pass algorithms consist of three main steps. The first step (first pass) scans the image to find out temporary/provisional label values of the pixels. The second step resolves the provisional values to find the final equivalent values. Lastly, the third step (second pass) gives each pixel its final label value [5,9–12,16–18].

Many two-pass algorithms have been proposed. The first algorithm was proposed by Rosenfeld and Pfaltz [16]. Most of the recent two-pass CCL algorithms are based on the principal of the Rosenfeld algorithm. The disadvantage of the Rosenfeld algorithm is the large memory requirement to store the provisional labels.

He et al. have presented several two-pass algorithms [5,10,11,28]. In [5], the provisional labels that are associated in the row are assigned to a set, and the smallest provisional value is obtained. The provisional labels in the set are resolved directly during the first scan. This algorithm is good in term of memory requirement. However, its implementation time depends on the image structure.

In both [10,28], a mask that consists of three neighboring pixels has been used to reduce the time to check the neighboring pixels in the first scan. These algorithms are fast. However, the memory required is still more compared to the other CCL algorithms. The algorithm [11] is based on a mask that consists of two rows neighboring pixels. This reduces the checking time in the first scan. However, the memory required is huge.

Grana et al. [12] used sixteen associated pixels and a decision tree to find the final equivalent label values. The memory requirement is large, and the execution time is increased with irregular and complex images. In [27], the decision tree and the union-find algorithms have been used to track equivalent labels. This algorithm is fast, but it requires large memory.

The algorithm [9] is one of the best algorithms in terms of execution time, but it has the classical disadvantage of a large memory requirement. The RCM algorithm [29] enhances the first-scan process using a 2×2 testing mask. This leads to generation of a large number of provisional label values, which leads to a huge memory requirement.

Based on the above, many CCL algorithms have been proposed. The available algorithms still face several disadvantages such as slow processing time, falling in the pipeline, and requiring a huge amount of memory [12–15]. The two-pass algorithms are the most popular. They are fast but need a large amount of memory with high resolution, noisy, or irregular images.

3. The proposed algorithm

The proposed algorithm finds low numbers of provisional labels quickly and simultaneously with a direct resolving process on each individual pixel and uses fewer rules and less memory. This is ensured by giving the adjacent segmented line-relative pixels in the current and bottom rows the same label value (provisional or equivalence) in one process. This reduces the number of rules and testing, which leads to reduced execution time in generating the provisional labels and resolving processes. Moreover, the direct resolving process with each pixel makes it practical to store the labels in the one-dimensional array.

Based on the two-pass CCL approach, Figure 1 shows the flowchart of the proposed algorithm. The first-pass step consists of forward and backward scan techniques to test, resolve, and insert provisional label values of the pixels. After the first pass tests all the pixels in the image, the next step resolves the provisional values to find the final equivalent label values. Finally, the second-pass step replaces the provisional label values of the pixels with their final label values.

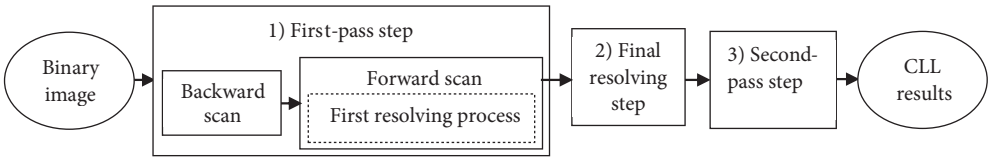


Figure 1. Flowchart of the proposed method.

In the proposed method, the input is a binary image that consists of the foreground and background pixels presented in the *Image* (X, Y), where X is the number of pixels in the row and Y is the number of pixels in the column. *Image* (x, y) denotes a pixel in the position x, y, where $1 \leq x \leq X$ and $1 \leq y \leq Y$. Due to the testing rules of the proposed algorithm, a value smaller than the possible label values is used initially for the foreground and background pixels of the input image. The smallest label value is 1; therefore, this work assumes that '0' denotes the background and '-1' denotes the foreground pixels. In addition, the boundary of one pixel of an input image is considered the background. To store the labels, $L(Z)$ is the representative label array, which is a one-dimensional array used for holding and resolving the label values. Initially, $L(Z) = Z$, where $Z = 1, 2, 3\ldots((X + Y)/4)$. Based on [5], the value $Z = ((X + Y)/4)$ is larger than any possible number of provisional labels.

3.1. First-pass step

In the first-pass phase, each pixel in the row is scanned twice in a backward and then a forward direction, and the associated pixels in both the current and bottom rows are given the same label values. The first scan, called the backward scan, moves backward from pixel X to 1 in the row to assign the label values to the current and its associated pixels. After the backward scan is completed, the second scan, called the forward scan, moves forward from pixel 1 to X in the same row to resolve and assign the label values. The same process is repeated on subsequent rows.

3.1.1. Backward scan

This process aims to assign label values to unlabeled pixels in two rows based on the neighboring label values or new label value if there are no neighboring label values. The backward scan tests each pixel in the row starting from the last pixel *Image* (X, y) to the first pixel *Image* (1, y) in the row. If the current scan pixel value is more than zero (*Image* $(x, y) > 0$), it means that it is a foreground pixel and already has a label value from the previous process. Thus, its label value is assigned to its associated foreground pixels in the left side *Image* $(x-1, y)$ and in the bottom row *Image* $(x+1, y+1)$, *Image* $(x, y+1)$, and *Image* $(x-1, y+1)$. The adopted mask is shown in Figure 2a, and the previous steps are shown in Algorithm (lines 1–7).

On the other hand, if the current pixel is foreground unlabeled pixel *Image* $(stx, y) = -1$, it means that the pixel has not been associated with a labeled pixel previously. Thus, if its left pixel is background *Image* $(x-1, y) = 0$ (Figure 2b), a new label value is assigned to the current pixel (*Image* $(x, y) =$ new label) and its associated foreground pixels in the bottom row *Image* $(x+1, y+1)$, *Image* $(x, y+1)$, and *Image* $(x-1, y+1)$. Otherwise, the pixel is ignored. This process is repeated until arriving at the first pixel in the row (*Image* $(1, y)$). Then the forward scan process is applied to the same row. The adopted mask of the backward scan is shown in Figure 2b, whereas the previous steps are shown in Algorithm (lines 8–13).

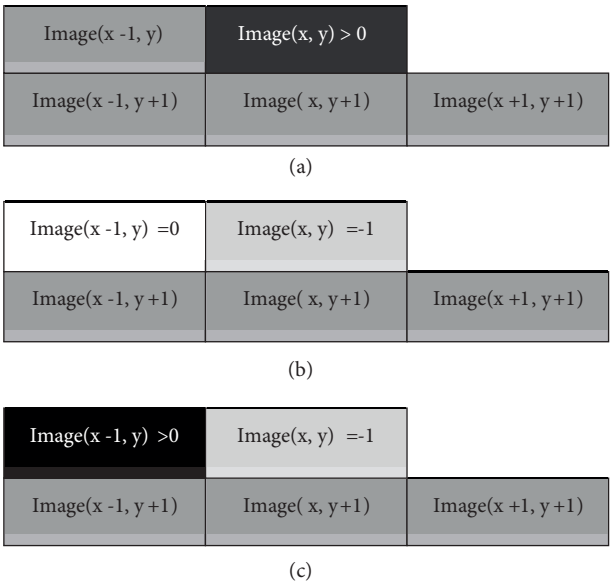


Figure 2. (a and b) depict the masks used in the backward scan, and (c) depicts the masks used in the forward scan. Black is the labeled pixel, white is the background pixel, light gray is the unlabeled foreground pixel, and dark gray could be foreground or background.

3.1.2. Forward scan

This process aims to resolve the provisional label value and assigns resolved values to ignored pixels of the previous backward scan process. In this work, the resolving process consists of two stages. The first stage is preparing a resolving process during the forward scan in the first-pass step with each pixel testing. It aims to find the equivalent or nearest value to the final equivalence label value of each pixel in the image. This reduces the amount of memory required and the resolving process time to find the final label values. The second stage is a one-time resolving process to find the final equivalence label values.

In the forward scan, unlabeled pixels must be associated to labeled pixels on the left side. To elaborate, the forward scan tests each pixel in the same row starting from the first pixel $Image\ (1,y)$ to the last pixel $Image\ (X,y)$. If the current pixel is an unlabeled foreground pixel ($Image\ (x,y) = -1$), it means that the associated left pixel must be a labeled foreground pixel ($Image\ (x-1,y)\ >0$). Thus, a resolved label value based on the label value of the left associated pixel ($L\ (Image\ (x-1,y))$ is assigned to the current pixel ($Image\ (x,y)$) and its associated foreground pixels in the bottom row ($Image\ (x+1,y+\ 1)$, $Image\ (x,y+\ 1)$, and $Image\ (x-1,y+1)$). The adopted mask is shown in Figure 2c and the previous steps are shown in Algorithm (lines 14–19). After each pixel testing process, rather than the pixel getting labeled or unlabeled, the first resolving process for the representative label array ($L\ (Label)$) is accrued concurrently.

In details about the first resolving process of the forward scan as shown in Figure 3, the aim of the first resolving process is to replace all the associated provisional label values by the smallest equivalent value. Therefore, if the both the current and left associated pixels are the background pixels, it means both of them are labeled pixels. If the associated label values ($L\ (Image\ (x,y))$) and ($L\ (Image\ (x-1,y))$) are different, a resolving process is required. Therefore, the next level of depth will be tested ($L(L(Image\ (x,y)))$) and ($L(L(Image\ (x-1,y)))$). If both of them are equal, it shows that they share the same root value, and the large label value will be assigned the small value. If both of them are different, the third depth equivalent of

the largest value will be replaced by the second depth equivalent value of the small value as shown in Eq. (1):

$$L(L(L(Large))) = L(L(Small)) \tag{1}$$

In the case the left associated pixel is the background pixel ($Image(x-1,y)$ =0) and the left upper pixel is foreground pixel ($Image(x-1,y-1)$ >0), the ($Image(x-1,y-1)$ >0) is assumed as the associated pixel with the current pixel, and the same process is implemented based on that. Algorithm (lines 20–25) shows the steps of the first stage of the resolving process. Figure 4a shows a sample about the forward scan case.

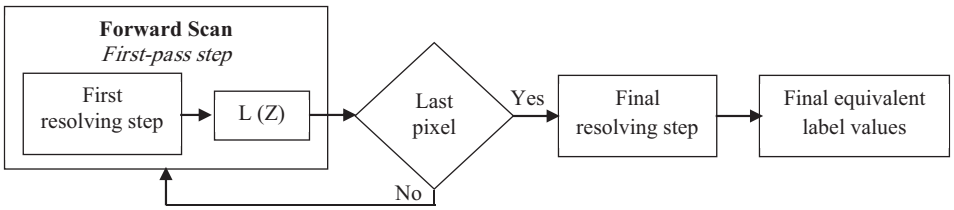


Figure 3. Flowchart of the resolving processes.

3.2. Final resolving step

After the first-pass step tests all pixels in the image, a one-dimensional array ($L\ (Z)$) is obtained. This array stores the provisional labels in ascending arrangement from small to large. The final resolving step occurs one time in the process on the representative label array $L\ (Z)$ to find the final equivalent label values. The main aim of this step is to replace all the associated provisional label values with one value only. By applying Eq. (2), the large provisional label is replaced by the smaller or equal associated value ($L\ (L\ (Z))$). The results are the final equivalence label values, which will be used in the next step. An example of the final resolving step is presented in Figure 4b.

$$L(Z) = L(L(Z)), \tag{2}$$

where $Z = 1$ is the largest label value. This step is shown in Algorithm (lines 8–13).

3.3. Second-pass step

Finally, each provisional label value in the image is replaced by its equivalent value in the representative label array *L (Label)* based on Eq. (3) (Algorithm (line 30)).

$$Image(x,y) = L(Image(x,y)), \tag{3}$$

All the steps of the proposed two-pass CCL algorithm are presented in Algorithm, while Figure 4 shows an example of the processes and the results of the proposed algorithm.

4. Experiments and results

The experiments and evaluations are presented in this section. The performance of the proposed algorithm is compared with the state of the art (HCS [5], LSL_{STD} [9], He et al. [10], HCS2 [11], Grana [12], Rosenfeld [16], and RCM [29] algorithms) based on the execution time of the first pass, the number of labels generated in the first pass, the execution time of the whole process, the stability of the performance with different benchmark images, and the memory required for buffering the provisional labels.

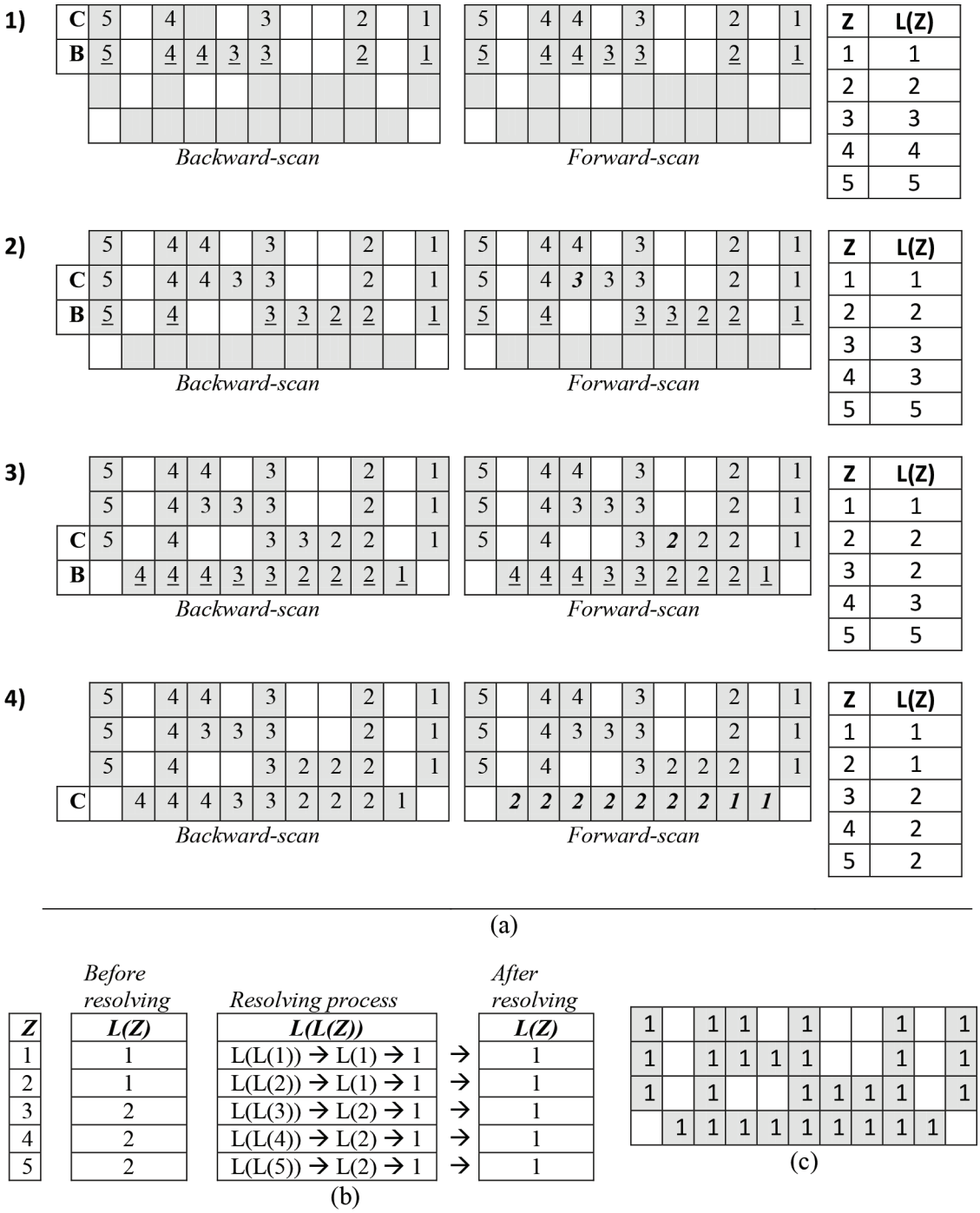


Figure 4. (a) The forward, backward, and label representative array of each scanned row, where 'C' is the current row, 'B' is the bottom row, underlined values are the associated pixels in the bottom row, and bold values are the values resolved during the forward scan; (b) depicts the final resolved values and (c) shows the final result.

The experiments were conducted on a PC-based workstation (Intel Course i5-6400 CPU @ 2.70 GHz, 8 GB memory, 64-bit OS). All the algorithms used were implemented with the VB.NET language.

Algorithm Forward-scan, backward-scan, and resolving processes of the proposed CCL algorithm.

```
// Initially: Label = 1, L (Z) = Z, // Image (x, y) = (white pixel = 0, black pixel = -1).
// The boundary of one pixel of an image is a background.
//-------------------------------------------------
// 1st pass process
1:  For y = 1 to Y  // for each row
2:      For x = X to 1 // for each pixel in the row (backward)
3:      IF (Image(x, y) > 0) Then // if scoped pixel is labeled
4:          IF Image (x, y + 1) = -1, Then Image (x, y + 1) ← Image(x, y)
5:          IF Image (x - 1, y + 1) = -1, Then Image (x - 1, y + 1) ← Image(x, y)
6:          IF Image (x + 1, y + 1) = -1, Then Image (x + 1, y + 1) ← Image(x, y)
7:          IF Image (x - 1, y) = -1, Then Image (x - 1, y) ← Image(x, y)
        // if scoped is pixel is unlabeled & next pixel is background
8:      ELSEIF (Image(x, y) = -1 AND Image(x - 1, y) = 0) Then
9:          Image(x, y) = Label
10:         Label = Label + 1
11:         IF Image (x, y + 1) = -1, Then Image (x, y + 1) ← Image(x, y)
12:         IF Image (x - 1, y + 1) = -1, Then Image (x - 1, y + 1) ← Image(x, y)
13:         IF Image (x + 1, y + 1) = -1, Then Image (x + 1, y + 1) ← Image(x, y)
14:     For x= 1 to X // for each pixel in the row (Forward)
15:     IF (Image(x, y) = -1) Then
16:         Image(x, y) ← L(Image(x - 1, y))
17:         IF Image (x, y + 1) = -1, Then Image (x, y + 1) ← Image(x, y)
18:         IF Image (x - 1, y + 1) = -1, Then Image (x - 1, y + 1) ← Image(x, y)
19:         IF Image (x + 1, y + 1) = -1, Then Image (x + 1, y + 1) ← Image(x, y)
        // Resolving (1st Stage). V1 = Image(x, y), V2 = Image(x - 1, y),
        // NOTE, If Image(x - 1, y) = 0"background", then V2 = Image(x - 1, y - 1).
20:     IF (L(V1) != L(V2)) AND (V2 != 0) Then
21:         IF L(L(V1)) > L(L(V2))) Then  L(L(L(V1))) ← L(L(V2)))
22:         ELSEIF L(L(V2)) > L(L(V1))) Then  L(L(L(V2))) ← L(L(V1)))
23:         ELSEIF L(L(V2)) = L(L(V1))) Then
24:         IF L(V1) > L(V2) Then L(L(L(V1))) ← L(L(V2)))
25:             ELSE  L(L(L(V2))) ← L(L(V1))
26: Return Image(x, y) // next row
// Resolving
27: For Z= 1 to Label
28:   L(Z) ← L(L(Z))
29: // 2nd pass process
30: For each pixel ,  Image(x, y) ← L(Image(x, y))
```

4.1. Databases

Several types of images are used in experiments, such as random noise, fingerprint, textual, skeletons, miscellaneous, aerials, shapes, and texture images. The random noise images dataset consists of 91 images in 512 ×

512, 256 × 256, 128 × 128, 64 × 64, and 32 × 32 sizes. These images present a randomly generated 1 × 1 noise size. In addition, the 512 × 512 size image presents randomly generated 1 × 1, 2 × 2, 4 × 4, 8 × 8, and 16 × 16 noise sizes. Each noise size is presented in nine images of densities from 10% to 90%, and the full white and black images are included. The fingerprint images dataset consists of 16 randomly selected images from the UPEK Fingerprint Database of size 248 × 338. The textual dataset presents the binary dataset of DIBCO2010 (H_DIBCO2010_GT), and it consists of 10 handwritten document binary images in several sizes. Ten images from miscellaneous, 10 images from aerials, and 16 texture images in 512 × 512 selected from the database of the University of Southern California were used. These images were converted to the binary format by the Niblack binarization algorithm [30]. Finally, 69 binary-shaped images selected from the MPEG7_CE-Shape-1 dataset were used in several sizes. In total, 232 images were used in these experiments. Examples of the dataset images used are shown in Figure 5.

4.2. Experiments of the first-pass step

Firstly, the random noise images were used to evaluate the performance of the first-pass step based on the number of pixels in the image. The dataset of images used is of sizes 512 × 512, 256 × 256, 128 × 128, 64 × 64, and 32 × 32 that are created from random one-pixel noise (black pixels on white background) of 50% density.

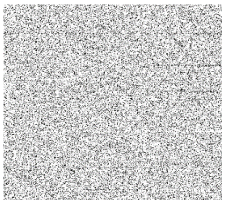
Based on the results of the number of labels generated in Figure 6a, the proposed algorithm showed the best and the most stable performance in terms of the number of generated provisional labels with less execution time. The performance of the proposed algorithm was similar to that of LSL_{STD} and HCS, while it was slightly better than the Grana algorithm. However, according to the performance based on the execution time in Figure 6b, the execution time of HCS, LSL_{STD}, and Grana was greater than that of the proposed algorithm. The proposed algorithm had the second best execution time. In terms of both the number of generated provisional labels and execution time of the first pass, the proposed algorithm is the most stable algorithm with increasing pixel number. This algorithm has the advantage of generating fewer provisional labels in less time.

The previous experiment was repeated on all the benchmark datasets. Based on the results shown in Figure 7a, the proposed algorithm, HCS and then the LSL_{STD} and Grana algorithms had the best performance in terms of number of provisional labels generated. Fewer provisional labels were generated in all the cases of the images, which led to a reduction in buffering memory requirement. However, in terms of execution time (Figure 7b), the proposed algorithm showed better performance than HCS, LSL_{STD}, and Grana, whereas the He et al. [10], RCM, and Rosenfeld algorithms showed faster time than the proposed algorithm. Thus, the proposed algorithm gave better performance for different types of images in both previous cases together than all the other algorithms as shown in Figure 7.

4.3. Experiments of complete performance

The next experiment evaluates the performance of the complete process of the proposed algorithm. A set of experiments were conducted to show the final performance of the proposed algorithm based on the execution time of the full process and the required buffering memory. Based on the results shown in Table 1 and Figure 8, the proposed algorithm is faster than the other algorithms with irregular and complex images.

Table 2 presents the buffering memory required of the provisional label for the resolving process in the worst case based on each algorithm structure. This experiment focuses on the main array structure that was used for the resolving process without being concerned about the number or structure of the secondary arrays used. The average number of provisional labels generated for each random noise image was extracted and then



(a)

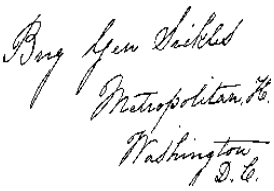
LSL STD	RCM
P: 5196	P: 6751
T: 623	T: 1354
HCS	**HCS2**
P: 5195	P: 6751
T: 1149	T: 773
Rosen.	**Grana**
P: 6751	P: 5429
T: 1369	T: 949
He. [10]	**Propos**
P: 8319	P: 5196
T: 2754	T: 9

(b)

LSL STD	RCM
P: 64	P: 103
T: 26	T: 25
HCS	**HCS2**
P: 64	P: 103
T: 32	T: 26
Rosen.	**Grana**
P: 103	P: 103
T: 25	T: 29
He. [10]	**Propos**
P: 152	P: 64
T: 25	T: 10

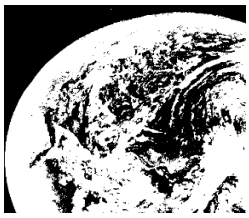
(c)

LSL STD	RCM
P: 59	P: 328
T: 9	T: 9
HCS	**HCS2**
P: 60	P: 139
T: 7	T: 8
Rosen.	**Grana**
P: 139	P: 58
T: 6	T: 7
He. [10]	**Propos**
P:140	P: 59
T: 25	T: 5



(d)

LSL STD	RCM
P: 79	P: 3015
T: 16	T: 385
HCS	**HCS2**
P: 97	P: 1109
T: 15	T: 34
Rosen.	**Grana**
P: 1109	P: 354
T: 42	T: 14
He. [10]	**Propos**
P:1140	P: 97
T: 36	T: 8



(e)

LSL STD	RCM
P: 1127	P: 3025
T: 57	T: 317
HCS	**HCS2**
P: 1127	P: 1596
T: 76	T: 115
Rosen.	**Grana**
P: 1596	P: 1158
T: 105	T: 64
He. [10]	Propos
P: 3988	P: 1127
T: 117	T: 9

(f)

LSL STD	RCM
P: 2691	P: 6149
T: 224	T: 1388
HCS	**HCS2**
P: 2691	P: 3526
T: 270	T: 204
Rosen.	**Grana**
P: 3526	P: 2786
T: 438	T: 264
He. [10]	**Propos**
P: 3805	P: 2691
T: 403	T: 10

(g)

LSL STD	RCM
P: 221	P: 1617
T: 10	T: 89
HCS	**HCS2**
P: 222	P: 462
T: 33	T: 13
Rosen.	**Grana**
P: 462	P: 290
T: 15	T: 13
He. [10]	**Propos**
P:483	P: 222
T: 16	T: 3

(h)

LSL STD	RCM
P: 2278	P: 6192
T: 239	T: 1633
HCS	**HCS2**
P: 2278	P: 3279
T: 343	T: 271
Rosen.	**Grana**
P: 3279	P: 3279
T: 474	T: 310
He. [10]	**Propos**
P:3852	P: 2278
T: 3523	T: 34

Figure 5. Number of provisional labels (P) and execution time [ms] (T) of CCL algorithms for examples images of (a and b) random noise image in different noise size and density; (c) is a shape image, (d) is a textual image, (e) is an aerial image, (f) is a texture image, (j) is a fingerprint image, and (h) is a miscellaneous image.

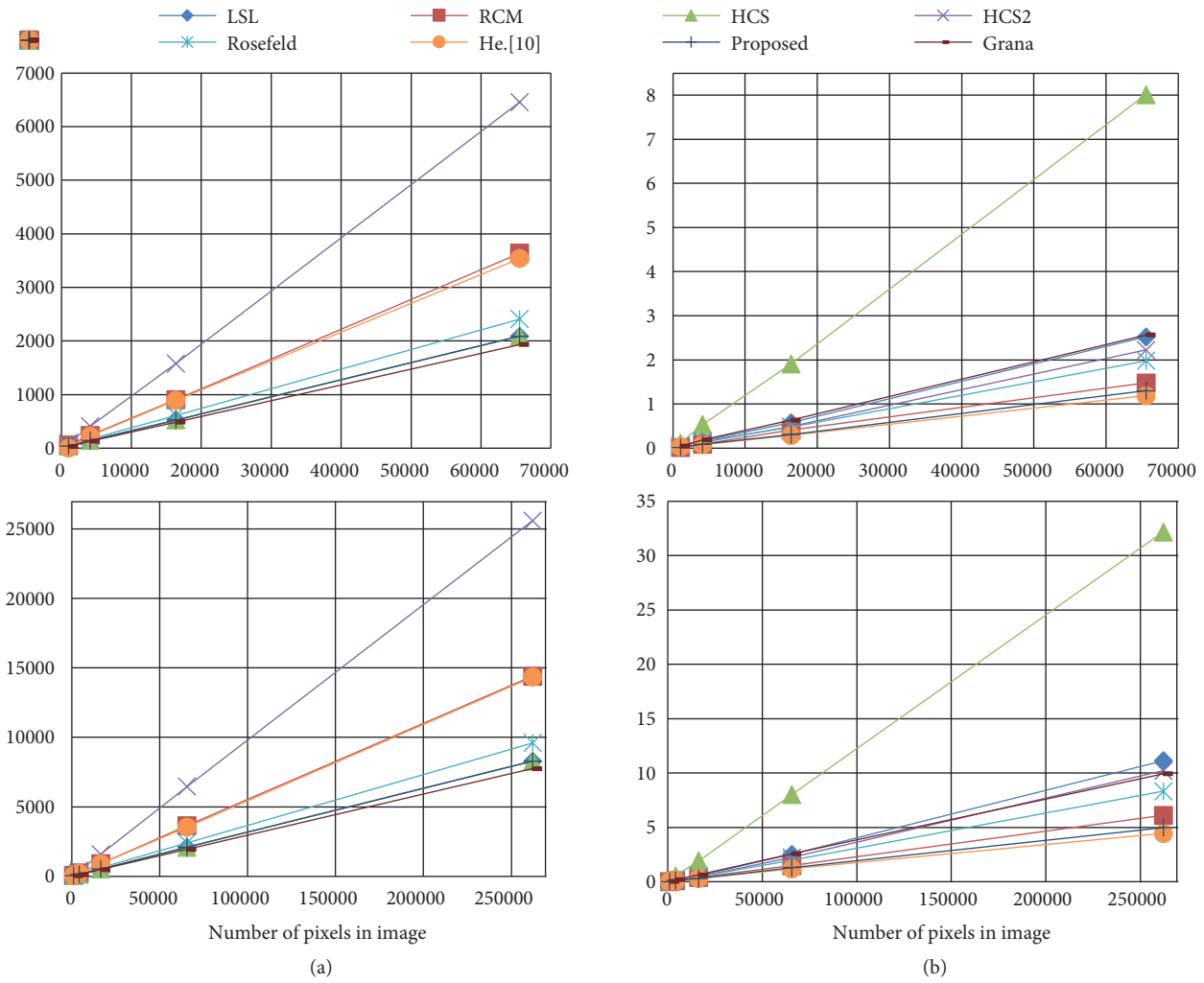


Figure 6. The efficiency of the first pass on the noise images, (a) number of provisional labels, and (b) execution time (ms) (top: smaller number of pixels, bottom: larger number of pixels).

the required memory was calculated. LSL_{STD}, RCM, Rosenfeld, and Grana used a two-dimensional array in the size of the number of provisional labels to store the provisional label values for the resolving process. This leads to a huge memory requirement with large, noisy, irregular, and high resolution images.

On the other hand, the proposed algorithm and HCS used a one-dimensional array in the size of the number of provisional labels. The proposed algorithm used only one array in the size of the number of provisional labels. This leads to using memory of small size under any image type. Therefore, the small number of provisional labels generated and use of a one-dimensional array makes the proposed algorithm extremely efficient compared to the other two-pass CCL algorithms. Figure 9 shows the increment in the required memory with the increase in the number of objects in the image. It is clear that the memory required in the proposed algorithm and the HCS algorithm is kept small and stable with any number of objects compared with the other algorithms.

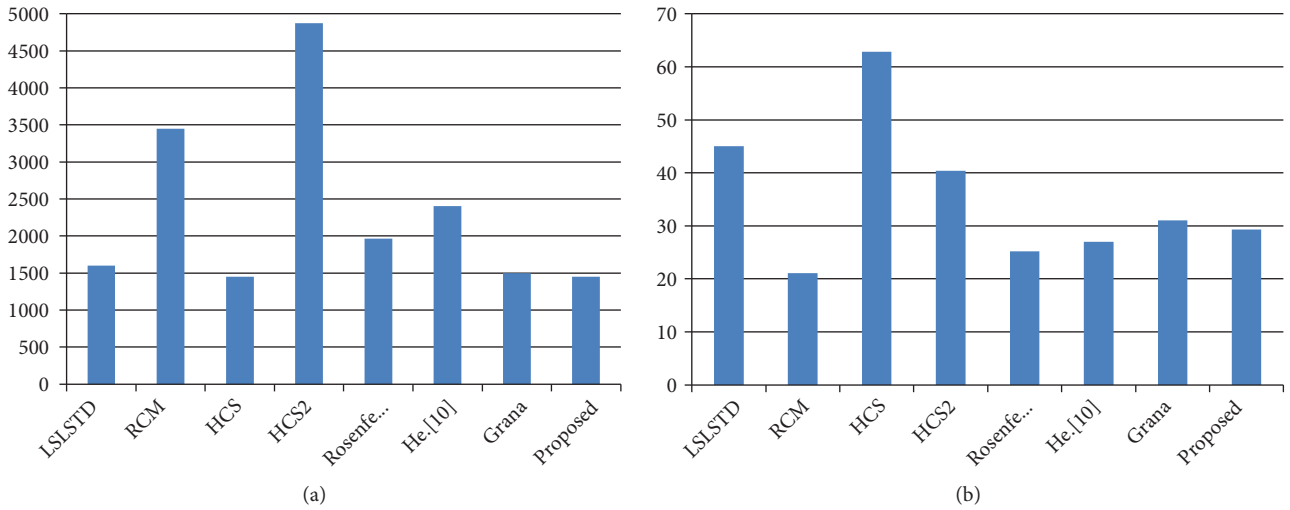


Figure 7. (a) The average of provisional label numbers and (b) average of execution time (ms) of all datasets.

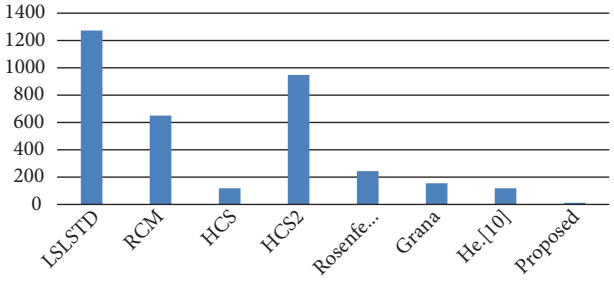


Figure 8. The average execution time (ms) of the full performance of each algorithm on all types of datasets.

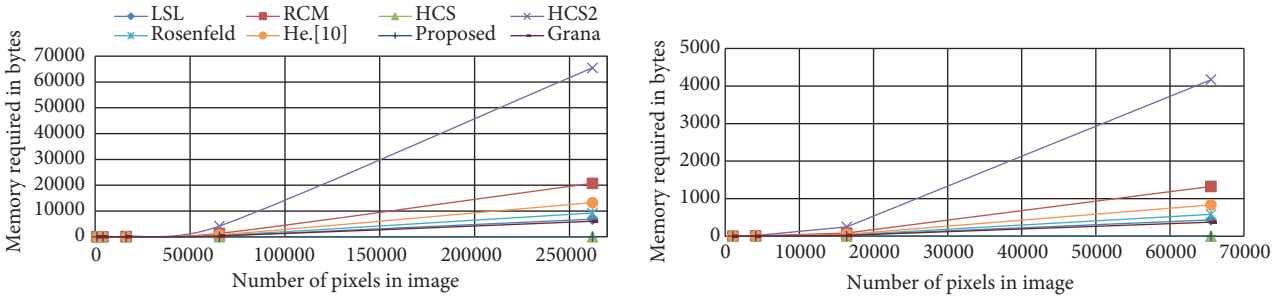


Figure 9. The average of the memory required in bytes (32 bits) for buffering the provisional label of the random noise image database (upper figure: larger number of pixels, lower figure: smaller number of pixels).

4.4. Discussion

In this work, the results were compared under two sets of conditions: based on the number of extracted provisional labels and its required time during the first-pass step, and based on the required buffering memory and time of the final resolving step. The number of provisional labels is an influential factor in the two-pass CCL approach. The large number of provisional values requires a large memory and more time during the resolving process, and vice versa.

Table 1. The average execution time (ms) of the full performance of each algorithm on each type of dataset.

		Noise	Fingerprint	Text	Misc.	Aerials	Texture	Shape
LSL_{STD}	Min.	63.17	11.75	41.2	704	2719	5031	27.36
	Mean	65.65	12.1	42.36	784.8	2777.8	5205.6	27.79
	Max.	70.45	12.5	44.8	812	2874	5284	28.49
RCM	Min.	157.06	82	665	495.3	1359.27	1711.25	29.42
	Mean	160.12	84.03	672.86	511.18	1363.51	1736.9	30.85
	Max.	161.79	88.88	688.22	528.1	1369.27	1748.75	31.89
HCS	Min.	55.02	15.75	59.1	82.8	230.18	312.6	25.92
	Mean	57.94	16.45	61.44	85.58	235.47	340.17	26.24
	Max.	67.62	18.81	64.7	90.6	240	368.8	27.11
HCS2	Min.	148.3	103.88	808.9	534.66	1994.55	2730.5	25.21
	Mean	149.79	104.74	824.7	594.18	2094.77	2841.42	25.83
	Max.	152.58	105.94	840.1	815.9	2178.4	2891	26.45
Rosenfeld	Min.	176.28	17.63	78.7	151.4	463.09	767.73	31.07
	Mean	178.22	18.16	80.76	154.1	474.16	772.14	31.44
	Max.	181.1	18.88	85.4	162.4	487.18	785.53	32.21
Grana	Min.	138.73	13.69	31.2	96.8	305.9	451.2	35.78
	Mean	140.97	14.15	32.3	98.78	308.27	457.76	36.28
	Max.	143.72	15.75	32.8	101.6	309.72	462.4	37.07
He [10]	Min.	53.64	13.5	57.8	79.6	227.27	309	24.01
	Mean	73.94	15.23	60.66	86.28	232.42	328.38	25.37
	Max.	109.6	16.63	62.7	93.7	238.64	341.5	27.32
Proposed	Min.	7.06	3.69	28.9	10.7	7	9.2	7.89
	Mean	7.18	3.98	29.86	12.12	11.92	12.9	8.07
	Max.	7.38	4.31	31.2	14.1	14.27	15.9	8.24

Table 2. The memory required in bytes (32 bits) based on the random noise image database. The Label column denotes the average number of provisional labels; Memory denotes the required memory, G1: 1×1 pixel noise, G2: 2×2 pixel noise, and G4: 4×4 pixel noise.

	G1		G2		G4	
	Label	Memory	Label	Memory	Label	Memory
LSL_{STD}	11782	138,815,524	3055	9,333,025	749	561,001
RCM	7852	61,653,904	3055	9,333,025	749	561,001
HCS	6344	6856	2020	2532	749	1261
HCS2	7852	61,653,904	3055	9,333,025	749	561,001
Rosenfeld	6775	45,900,625	1734	3,006,756	440	193,600
He. [10]	6775	45,900,625	1734	3,006,756	440	193,600
Proposed	6775	6775	1734	1734	440	440
Grana	6775	45,900,625	1734	3,006,756	440	193,600

Another influential factor is the adopted technique for the resolving process to find the final equivalent label values. Most of the two-pass CCL algorithms used a two-dimensional array in the size of the number of provisional labels to store the provisional label values for the resolving process. That requires a huge memory and more execution time. This case appeared in the He et al. [10], HCS2 [11], Grana [12], Rosenfeld [16], and RCM [29] algorithms. On the other hand, the proposed and HCS algorithms use a one-dimensional array in the size of the number of provisional labels. That uses a small size of memory and fast resolving process under any image type.

All the above factors interact with each other to produce the full performance of each algorithm. The ability of the CCL algorithms to overcome the challenges of buffering memory and execution time of the whole CCL process was the focus of all previous research. Based on the experiments, the results can be summarized as follows:

- The HCS2 algorithm extracted a large number of provisional label values with a slow first-pass step. That led to the use of a large amount of memory with a slow resolving process. Therefore, most of CCL algorithms used were better than HCS2 in all cases.
- The LSLSTD algorithm extracted a small number of provisional label values, but the first-pass step was slow. The resolving process required less memory compared with the HCS2, RCM, He et al. [10], and Rosenfeld algorithms. However, it was notable that the execution time of the LSLSTD algorithm is uneven depending on the type of images. As shown in Table 2, the LSLSTD algorithm was fast with noise, fingerprint, text, miscellaneous, and shape images. However, it was extremely slow with aerials and texture images. That means the LSLSTD algorithm is not effective with irregular images.
- The RCM algorithm was fast to extract provisional label values. However, the number of extracted provisional labels was large. That led to the use of a large amount of memory with a slow resolving process. The Rosenfeld, Grana, He et al. [10], HCS, and the proposed algorithms showed better performance with the whole CCL process.
- The Rosenfeld, Grana, and He et al. [10] algorithms extracted a reasonable number of provisional labels with a reasonable execution time for the first-pass step. This led to the use of reasonable buffering memory during the fast resolving step. The results showed that the proposed and HCS algorithms were faster and required smaller memory.
- The HCS algorithm extracted a small number of provisional labels, but the first-pass step was slow. Because the HCS algorithm adopted a run-base resolving process technique, it used little buffering memory like the proposed method. Moreover, the execution time of the resolving process was extremely fast. Accordingly, the HCS algorithm produced better performance than previous algorithms with the whole CCL process.
- The proposed algorithm extracted a small number of provisional label values during a fast first-pass step. Moreover, it adopted a run-base technique for the resolving process, which means it used little buffering memory. In addition, its execution time of the whole CCL process was faster than that of the other algorithms.

The above analysis shows that the proposed method is more effective than previous methods. The proposed algorithm gathered all the required advantages of the CCL process. It generated fewer provisional

label values similar to the LSLSTD, HCS, and Grana algorithms. In terms of execution time of provisional label values, the proposed algorithm's performance was similar to that of the set of faster algorithms such as LSLSTD and Grana. In addition, in terms of execution time of the whole CCL process, the proposed algorithm's performance was faster than that of the other algorithms such as HCS and He et al. [10]. Moreover, the performance of the proposed algorithm in terms of the memory required was much better than that of the other algorithms. The only two-pass CCL algorithms that require less memory are HCS and the proposed algorithm. However, the execution time of the HCS algorithm is slow compared with the proposed algorithm.

5. Conclusion

In this paper, a fast, memory-efficient, and simple two-pass CCL algorithm for binary images was presented. The proposed algorithm is based on a new first-pass phase, which adopted a new run-based tracing technique in the backward scan process generating a small number of provisional labels. Next, a new resolving algorithm occurred directly in the process of the forward scan. It requires an extremely small amount of buffering memory, because it uses a one-dimensional array to store the labels. A set of experiments were conducted on different benchmark datasets of binary images. The results were compared with those of a set of the best-known CCL algorithms, such as LSL_{STD}, RCM, HCS, HCS2, Rosenfeld, and Grana. The experiments showed that the proposed algorithm gathered the best performances in all the required features of CCL algorithms. It has a short execution time and requires an extremely small amount of memory compared to state of the art two-pass CCL algorithms.

Acknowledgment

Special thanks are due to the Deanship of Scientific Research, Umm Al-Qura University, Saudi Arabia.

References

[1] Soh Y, Ashraf H, Hae Y, Kim I. A hybrid approach to parallel connected component labeling using cuda. International Journal of Signal Processing Systems 2013; 1 (2): 130-135.

[2] Di Stefano L, Bulgarelli A. A simple and efficient connected components labeling algorithm. In: IEEE International Conference Proceedings in Image Analysis and Processing; Venice, Italy; 1999. pp. 322-327.

[3] Rakhmadi A, Othman Z. Bade A, Rahim M, Amin I. Connected component labeling using components neighbors-scan labeling approach. Journal of Computer Science 2010; 6 (10): 1070-1078.

[4] He L, Chao Y. A very fast algorithm for simultaneously performing connected-component labeling and Euler number computing. IEEE Transactions on Image Processing 2015; 24 (9): 2725-2735. doi: 10.1109/TIP.2015.2425540

[5] He L, Chao Y, Suzuki K. A run-based two-scan labeling algorithm. IEEE Transactions on Image Processing 2008; 17 (5): 749-756. doi: 10.1109/TIP.2008.919369

[6] Eusse JF, Leupers R, Ascheid G, Sudowe P, Leibe B, Sadasue T. A flexible ASIP architecture for connected components labeling in embedded vision applications. In: IEEE Design, Automation and Test in Europe Conference and Exhibition; Dresden, Germany; 2014. pp. 1-6.

[7] He L, Chao Y, Suzuki K. An algorithm for connected-component labeling, hole labeling and Euler number computing. Journal of Computer Science and Technology 2013; 28 (3): 468-478. doi.org/10.1007/s11390-013-1348-y

[8] Klaiber M, Rockstroh L, Wang Z, Baroud Y, Simon S. A memory-efficient parallel single pass architecture for connected component labeling of streamed images. In: International Conference on In Field-Programmable Technology; Seoul, South Korea; 2012. pp. 159-165.

[9] Lacassagne L, Zavidovique B. Light speed labeling: efficient connected component labeling on rise architectures. Journal of Real-Time Image Processing 2011; 6 (2): 117-135. doi.org/10.1007/s11554-009-0134-0

[10] He L, Chao Y, Suzuki K. An efficient first-scan method for label-equivalence-based labeling algorithms. Pattern Recognition Letters 2010; 31 (1): 28-35. doi.org/10.1016/j.patrec.2009.08.012

[11] He L, Chao Y, Suzuki K. A new two-scan algorithm for labeling connected components in binary images. In: IEEE Proceedings of the World Congress on Engineering; London, UK; 2012. pp. 1141-1146.

[12] Grana C, Borghesani D, Cucchiara R. Connected component labeling techniques on modern architectures. In: International Conference on Image Analysis and Processing Image Analysis and Processing; Italy; 2009. pp. 816-824.

[13] Cabaret L, Lacassagne L, Oudni L. A review of world's fastest connected component labeling algorithms: Speed and energy estimation. In: IEEE Conference in Design and Architectures for Signal and Image Processing; Madrid, Spain; 2014. pp. 1-6.

[14] Hashmi MF, Pal R, Saxena R, Keskar AG. A new approach for real time object detection and tracking on high resolution and multi-camera surveillance videos using GPU. Journal of Central South University 2016; 23 (1): 130-144. doi.org/10.1007/s11771-016-3056-6

[15] Walczyk R, Armitage A, Binnie TD. Comparative study on connected component labeling algorithms for embedded video processing systems. In: International Conference in Image Processing, Computer Vision, and Pattern Recognition; Las Vegas, NV, USA; 2010. pp. 176-183.

[16] Rosenfeld A, Pfaltz JL. Sequential operations in digital picture processing. Journal of the ACM 1966; 13 (4): 471-494.

[17] Gotoh T, Ohta Y, Yoshida M, Shirai Y. Component labeling algorithm for video rate processing. In: Hague International Symposium, International Society for Optics and Photonics; Hague, the Netherlands; 1987. pp. 217-224.

[18] Wu K, Otoo E, Suzuki K. Optimizing two-pass connected-component labeling algorithms. Pattern Analysis and Applications 2009; 12 (2): 117-135. doi.org/10.1007/s10044-008-0109-y

[19] Shima Y, Murakami T, Koga M, Yashiro H, Fujisawa H. A high-speed algorithm for propagation-type labeling based on block sorting of runs in binary images. In: 10th IEEE International Conference in Pattern Recognition; Atlantic City, NJ, USA; 1990. pp. 655-658.

[20] Devi GG, Sumathi CP. Positional connected component-labeling algorithm. Indian Journal of Science and Technology 2014; 7 (3): 306-311.

[21] Bailey DG, Johnston CT. Single pass connected components analysis. In: Proceedings of Image and Vision Computing New Zealand; Hamilton, New Zealand; 2007. pp. 282-287.

[22] Chang F, Chen CJ, Lu CJ. A linear-time component-labeling algorithm using contour tracing technique. Computer Vision and Image Understanding 2004; 93 (2): 206-220.

[23] Chang F, Chen CJ. A component-labeling algorithm using contour tracing technique. In: 7th IEEE International Conference on Document Analysis and Recognition; Edinburgh, UK; 2003. pp. 741-745.

[24] Pavlidis T. Algorithms for Graphics and Image Processing. Heidelberg, Germany: Springer Science & Business Media, 2012.

[25] Haralick RM. Some neighborhood operators. In: Real-Time Parallel Computing. Boston, MA, USA: Springer, 1981.

[26] Suzuki K, Horiba I, Sugie N. Linear-time connected-component labeling based on sequential local operations. Computer Vision and Image Understanding 2003; 89 (1): 1-23. doi.org/10.1016/S1077-3142(02)00030-9

[27] Wu K, Otoo E, Shoshani A. Optimizing connected component-labeling algorithms. In: Medical Imaging 2005: Image Processing. International Society for Optics and Photonics; San Diego, CA, USA; 2005. pp. 1965-1976.

[28] He L, Chao Y, Suzuki K, Wu K. Fast connected-component labeling. Pattern Recognition 2009; 42 (9): 1977-1987. doi.org/10.1016/j.patcog.2008.10.013

[29] Hernandez-Belmonte U, Ayala-Ramirez V, Sanchez-Yanez R. Enhancing CCL algorithms by using a reduced connectivity mask. In: Mexican Conference on Pattern Recognition; Querétaro, Mexico; 2013. pp. 195-203.

[30] Niblack W. An Introduction to Digital Image Processing. Englewood Cliffs, NJ, USA: Prentice-Hall, 1986.