A new run-based Connected Component Labeling for efficiently analyzing and processing holes

Florian Lemaitre, Lionel Lacassagne

A NEW RUN-BASED CONNECTED COMPONENT LABELING FOR EFFICIENTLY ANALYZING AND PROCESSING HOLES

Florian Lemaitre and Lionel Lacassagne

Sorbonne University, CNRS, LIP6, France
fisrtname.name@lip6.fr

ABSTRACT

This article introduces a new connected component labeling and analysis algorithm for foreground and background labeling that computes the adjacency tree. The computation of features (bounding boxes, first statistical moments, Euler number) is done on-the-fly. The transitive closure enables an efficient hole processing that can be filled while their features are merged with the surrounding connected component without the need to rescan the image. A comparison with existing algorithms shows that this new algorithm can do all these computations faster than algorithms processing black and white components.

***Index Terms*—** Connected component labeling and analysis, Euler number, adjacency tree, hole processing, hole filling.

1. INTRODUCTION & STATE OF THE ART

Connected Component Labeling (CCL) is a fundamental algorithm in computer vision. It consists in assigning a unique number to each connected component of a binary image. Since Rosenfeld [1], many algorithms have been developed to accelerate its execution time on CPU [2–4], SIMD CPU [5], GPU [6] or FPGA [7].

In the same time Connected Component Analysis (CCA) that consists in computing Connected Component (CC) features – like bounding-box to extract characters for OCR, or the first raw moments (S, S_x, S_y) for motion detection and tracking – has also risen [8–12]. Parallelized algorithms have been also designed [13–15]. The initial Union-Find algorithm [16] has been also analysed [17] and improved [18] with decision tree [19] and various path compression/modification algorithms [20, 21].

Some other features – useful for pattern classification/recognition – are computed by another set of algorithms: the Euler number with Bit-Quads [22], the adjacency (also known as homotopy or inclusion) tree [23] and more recently, foreground (FG) and background (BG) labeling (also known as B&W labeling) [24] and hole filling [24] with also improvements in the last decade: [25, 26].

Section 2 provides a short description of LSL algorithm that is used for our new algorithm, Section 3 introduces our new algorithm, Section 4 presents a benchmark of existing algorithms and their analysis and Section 5 is the conclusion.

2. CLASSICAL LSL

We chose to base our new algorithm on LSL [10] because it is run-based (segment processing) and thus is able to compute features very quickly compared to pixel-based algorithms.

The specificity of LSL is to combined segments with line-relative labeling to speed segment adjacency detection up. Like all direct CCA algorithms, it is split into two steps and performs a single image scan.

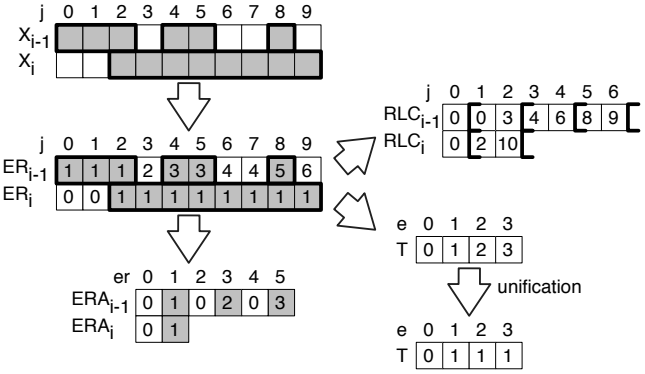


Fig. 1: LSL tables for classic foreground labeling: the *even* relative labels corresponding to background components are equivalent to zero in ERA tables (they will be equivalent to other labels for the proposed foreground and background labeling).

The first step assigns a temporary/provisional label to each connected component and computes its associated features. Equivalences between labels are built when needed. It is composed of a segment detection (Algorithm 1) and a segment Unification (Algorithm 2 modified for B&W labeling). From two consecutive input lines X_{i-1}, X_i, two relative labelings (ER_i, ER_{i-1}) are produced where FG runs (or segments) have odd numbers and BG runs have even numbers. The associated semi-open intervals are stored in tables RLC_{i-1}, and RLC_i. The table ERA_i holds the translation between *Relative* and *Absolute* labels: $ea = ERA_i[er]$.

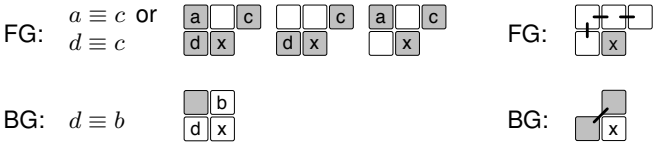
To find out which labels of the previous line are connected to the current segment (Unification), one has to read the value of relative labels from table ER_{i-1} at the positions given by RLC_i, and translates them into absolute labels to update the equivalence table T. Features are also computed during this step and stored in table F.

The second step solves the equivalence table by computing the transitive closure (TC) of the graph associated to the label equivalence and merge the features together. Unlike for CCL algorithms, the third step that performs a second scan to replace temporary labels of each connected component with their final labels (Algorithm 4) is usually avoided (but can be done for visual verification).

In order to differentiate *temporary* root and *final* root, we define a ($a \leftarrow \mathrm{Find}(e_a)$) as the *temporary* root of the absolute label e_a used in Algorithm 2, while r used in Algorithm 3 and Algorithm 4 is the *final* root of the connected component.

Algorithm 1: segment detection (LSL step 1a)

```
1  // prolog
2  RLC_i[0] ← 0
3  er ← 0
4  x_1 ← 0      // previous value of X (x_1 = X_i[j − 1])
5  for j = 0 to w − 1 do
6      x_0 ← X_i[j]      // current value
7      RLC_i[er + 1] ← j      // will later be overwritten if f = 0
8      f ← x_0 ⊕ x_1      // edge detection
9      er ← er + f
10     ER_i[j] ← er
11     x_1 ← x_0      // register rotation
12 // epilog
13 RLC_i[er + 1] ← w
14 ner_i ← er + 1
15 return ner_i
```



(a) Closing pixel patterns (b) New label patterns

Fig. 2: Patterns demonstrating how surroundings are detected when the current pixel x is processed. Foreground is 8-adjacent and background is 4-adjacent.

3. LSL AND HOLE PROCESSING

Our goal is to propose a new *all-in-one* algorithm that finds holes and processes them efficiently by computing the adjacency tree between foreground and background connected components. Then, filling holes only consists in modifying the adjacency tree and the equivalence table, without modifying pixels or labels in the image (the same approach can be used to apply arbitrary connected operators [27]). We also want our algorithm to be able to compute statistical features on-the-fly, to relabel the whole image and to compute adjacency features like the Euler number – or just a subset.

In the following, a "component" means a Connected Component, either foreground or background.

3.1. Holes and adjacency tree

A component C_1 is surrounded by another component C_2 – written $C_1 \sqsubset C_2$ – if and only if all paths from C_1 to the exterior of the image contain at least one pixel from C_2. A hole is a background component that is surrounded by a foreground component.

The adjacency tree is encoded in a new table I. For a label e_1 associated to a component C_1, $e_2 = I[e_1]$ is one of the temporary labels of the unique component C_2 that is both adjacent to C_1 and surrounding C_1 ($C_1 \sqsubset C_2$). $I[e_1] = -1$ if $e_1 = 0$, or e_1 is not a root label ($T[e_1] \neq e_1$). In short, the table I represents the adjacency tree whose edges are directed according to the surrounding relation.

We considered two methods to build the adjacency tree and the surrounding relation: detecting closing pixels [28], or looking at the adjacency at exterior pixels [26].

A closing pixel is a pixel neighboring both ends of a path (using 4-adjacency for BG and 8-adjacency for FG). Figure 2a shows the patterns of a closing pattern while it is being processed. The surrounding $C_1 \sqsubset C_2$ where C_1 and C_2 are adjacent can be detected during the Unification when C_2 is unified with itself. If C_2 is FG (8-adjacency), the pixel above the closing pixel is in C_1. If C_2 is BG (4-adjacency), the upper-left pixel is in C_1.

The other method relies on the adjacency at exterior pixels. In our case, we consider the top most pixels of a component because those require the creation of a new label which is easy to detect. Every time a new label is created, the label directly above the current pixel is recorded in I as its initial adjacency and speculative surrounding. It is actually simpler to look for the label on the left that is necessarily from the same component as above (Figure 2b). When two labels $a < b$ are unified, the initial adjacency $I[b]$ is discarded in favor of $I[a]$. The order on labels implies that top pixels of a are higher than top pixels of b – or at least at the same height. It means that the higher initial adjacency and speculative surrounding is kept while the other is discarded. Once a component has been fully scanned, the only initial adjacency kept for this component is the one from the root label which is, by construction, the label of top most pixels, and thus on the exterior of the component. The remaining initial adjacency and speculative surrounding is thus necessarily a true surrounding.

We chose to use the initial adjacency method as it saves one extra branch and one extra Find within the Unification compared to the closing pixel method. Moreover, the update of I when an adjacency is discarded is actually not necessary as I is accessed only for root labels whose initial adjacencies are kept by construction. While the adjacency is a local property, the surrounding is not and thus is defined and correct only when the component has been fully scanned. Consequently, initial adjacency builds a speculative I that is correct only at the end of the image scan and that cannot be worked on beforehand.

3.2. New Black and White LSL

Our new algorithm – LSL BW – extends the unification and the transitive closure steps to support B&W labels and the adjacency tree (Algorithm 2 and Algorithm 3). B&W labels require to process both odd and even segments instead of just odd ones (in Algorithm 2 and 4). Consequently, ERA_i table does not necessarily have zeros at even indices anymore. The first encoded segment is always a BG one, but might have 0 length if the first pixel is FG. Thus, the unification needs a prolog (Algorithm 2, lines 1 to 5) to skip the first segment if empty, and an epilog (Algorithm 2, lines 39 to 44) to attach the first segment and the last segment if it is BG to the exterior. The transitive closure need no modification to handle Black&White labels.

In addition, the correction of coordinate and indices are now dependant on the parity of the segment being processed (Algorithm 2, lines 13 and 18). Note that Algorithm 2 implements the algorithm using 8-adjacency for FG and 4-adjacency for BG. The line 11 should be modified to $c_8 \leftarrow p \oplus 1$ if one wants the complementary adjacency (4-FG, 8-BG).

The construction of the adjacency table requires only a few modifications to the algorithms to set the initial adjacency (Algorithm 2 line 37) and to convert temporary labels into final labels within the I table (Algorithm 3 line 11). Hole filling is done during the transitive closure (Algorithm 3, lines 3 to 5).

Like for classical LSL, the computed features for each FG and BG components are the bounding-box and the first statistical moments S, S_x, S_y.

Algorithm 2: B&W Unification (step 1b)

```
er_s ← 0    // starting segment
if RLC_i[1] = 0 then
    ERA_i[0] ← 0
    er_s ← 1
// White-only would have er_s = 1 and step = 2
for er = er_s to ner_i − 1 step 1 do
    // semi open interval segment extraction [j_0, j_1[
    j_0 ← RLC_i[er]
    j_1 ← RLC_i[er + 1]
    p ← er mod 2    // parity of current segment
    c_8 ← p    // if current segment is 8-C, c_8 = 1
    f ← ComputeFeatures(i, j_0, j_1)
    // fix extension in case of 8-connected component
    j_0 ← max(j_0 − c_8, 0)
    j_1 ← min(j_1 + c_8, w)
    er_0 ← ER_{i−1}[j_0]
    er_1 ← ER_{i−1}[j_1 − 1]    // right compensation
    // fix label parity: BG segments are even, FG segments are odd
    er_0 ← er_0 + ((er_0 mod 2) ⊕ p)
    er_1 ← er_1 − ((er_1 mod 2) ⊕ p)
    if er_1 ≥ er_0 then
        e_a ← ERA_{i−1}[er_0]
        a ← Find(e_a)
        for er_k = er_0 + 2 to er_1 step 2 do
            ea_k ← ERA_{i−1}[er_k]
            a_k ← Find(ea_k)
            if a < a_k then    // Union: min propagation
                T[a_k] ← a
            if a > a_k then    // Union: min extraction
                T[a] ← a_k
                a ← a_k
        ERA_i[er] ← a    // the global min
        F[a] ← F[a] ∪ f    // update features
    else    // new label
        ERA_i[er] ← ne
        F[ne] ← f
        I[ne] ← ERA_i[er − 1]    // Initial adjacency
        ne ← ne + 1
// first and last BG segments shall be connected to 0
a ← Find(ERA_i[0])
T[a] ← 0
if ner_i is odd then    // last segment is BG
    a ← Find(ERA_i[ner_i − 1])
    T[a] ← 0
```

Algorithm 3: B&W Transitive closure (step 2)

```
for e = 0 to ne − 1 do
    a ← T[e]    // ancestor
    if Hole filling and e = a then    // If label is root
        i ← I[e]    // label of the surrounding component
        if T[i] > 0 then a ← T[e] ← i
    if a < e then
        r ← T[a]
        T[e] ← r    // Transitive Closure
        F[r] ← F[r] ∪ F[e]    // Feature merge
    else    // e is a root
        I[e] ← T[I[e]]    // point adjacency to root
```

Algorithm 4: B&W Relabeling (step 3)

```
for i = 0 to h − 1 do
    j_0 ← RLC_i[0]    // j_0 is 0
    // White-only would have have step = 2 and unconditionally store
      the BG label for even er
    for er = 0 to ner_i − 1 step 1 do
        e ← ERA_i[er]    // provisional label
        r ← T[e]    // final label
        j_1 ← RLC_i[er + 1]
        E_i[j_0, j_1[ ← r
        j_0 ← j_1
```

3.3. Example

Figure 3 shows how our algorithm builds the equivalence table T and the adjacency tree I on a simple, yet complete, example. It shows the input image with initial labels and their speculative surrounding (FG in gray and BG in white), as well as a graph representing both the equivalence table T and the adjacency tree I.

On the first three lines ($i = 0$, $i = 1$ and $i = 2$), five new labels are created ①, ②, ③, ④ and ⑤. Their initial adjacency is set as their speculative surrounding: ① ⊏ ⓪, ② ⊏ ①, ③ ⊏ ①, ④ ⊏ ② and ⑤ ⊏ ③.

At $i = 3$, two new labels are created with the following speculative surroundings: ⑥ ⊏ ④ and ⑦ ⊏ ⑤. In addition, ③ ≡ ② is detected. Consequently, the speculative surrounding of ③ is discarded in favor of ② ⊏ ①.

At $i = 4$, as ⑤ ≡ ④, the speculative surrounding ⑤ ⊏ ③ is discarded.

At $i = 5$, two new equivalences are detected: ② ≡ ⓪ and ⑦ ≡ ⑥. Consequently, the speculative surroundings of ② and ⑦ are dropped. The component ⓪②③ has no more surrounding as ⓪ is the exterior of the image. While the algorithm is not capable to detect it, we can see that the surrounding ⑥ ⊏ ④ is no more speculative and is actually final.

At $i = 6$, the last equivalence ④ ≡ ① is detected and the speculative surrounding ④ ⊏ ② is discarded, and the surrounding ① ⊏ ⓪ is kept.

This leads to the final state before transitive closure where all remaining surroundings (⑥ ⊏ ④ and ① ⊏ ⓪) are no more speculative and are actually true surroundings. When holes are filled, the adjacency edge ⑥ ⊏ ④ is replaced by an equivalence edge ⑥ ≡ ④. Note that our algorithm actually fills hole during transitive closure and not beforehand.

4. BENCHMARK & PERFORMANCE ANALYSIS

We measured the performance of our algorithms using a protocol similar to [29]. We tested randomly generated 2048×2048 images with varying density and granularity on a Skylake Gold 6126 Xeon @2.60GHz. Grana's [4] and Diaz' [30] works have been ran and measured on our machine. We also ran the CCA algorithms from [2] on our machine and kept three of them: Rosenfeld+F as the reference, He HCS2+F and LSL_{STD}+F as the fastest pixel-based and run-based algorithms at this time. The other ones have been estimated from their paper. To have comparable results across machines, we give all the results in cycles per pixel (cpp) that is the execution time multiplied by the clock frequency and divided by the number of pixels.

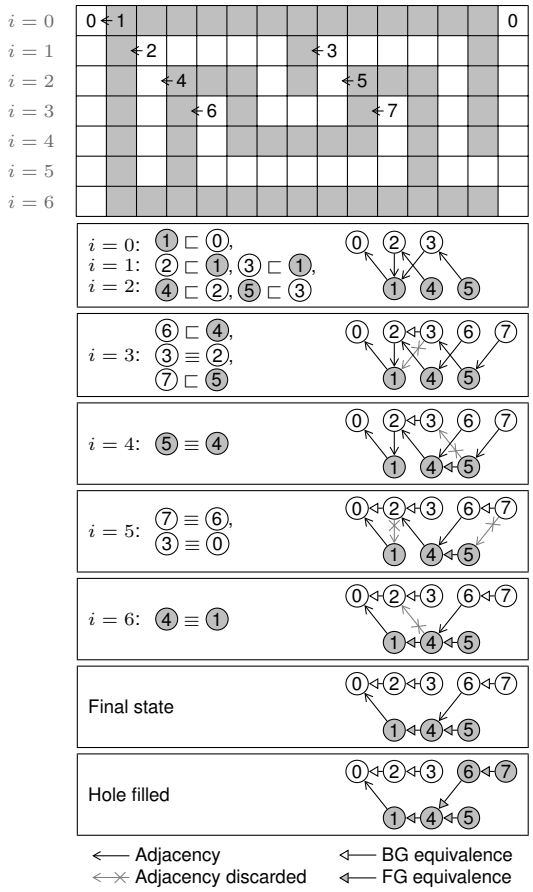


Fig. 3: Step by step example of our new B&W labeling focusing on equivalences building and adjacency setting.

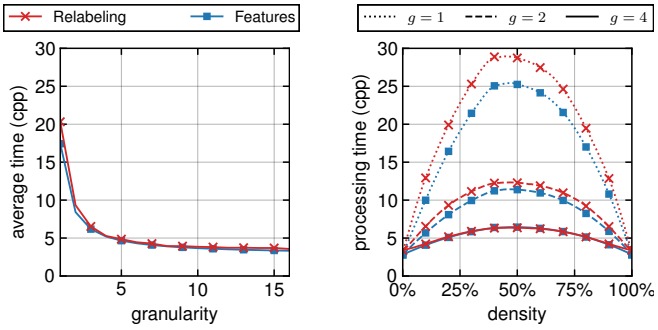
Table 1 shows the minimal and maximal processing time of our new labeling algorithm. The first line corresponds to a *base* processing: foreground and background CC labeling and computing their adjacency tree. The next lines provide the extra times to do extra computations like Euler or Hole Filling and B&W Feature Computation. The extra times are the worst case we measured for doing this extra computation. One can then estimate the total processing time for all the computations they are interested in just by adding all the extra times.

On this table, we can observe that the minimal extra time for all computations but relabeling is 0. This is a property of run-based algorithms: those computation times depend on the number of segments – which is 1 per line for empty images. Euler number computation and Hole filling are really inexpensive using our approach. On the opposite, relabeling is very expensive, almost doubling the total time, and should be avoided if not required.

Figure 4 shows the processing time of the labeling with hole filling and either relabeling or feature computation depending on both granularity and density. We can see that the processing time quickly decreases with higher granularity. Random images with $g = 1$ are highly artificial but are useful to stress the algorithms where they appear to be slowest. Our labeling is much faster at $g = 4$ with an average time of 5.3 cpp. This is interesting because natural images (like SIDBA database) usually have an average processing time close to the processing time of random images with granularity greater than 4, according to [2].

	min	max
B&W + Adjacency	2.80	15.3
+Euler number	+ 0	+ 0.66
+Hole Filling	+ 0	+ 0.74
+Feature Computation	+ 0	+ 10.3
+Relabeling	+ 0.59	+ 13.2

Table 1: Minimal and maximal processing time in cpp of our base B&W algorithm and extra computation for 2048×2048 random images.



(a) Average processing time over foreground density for $g \in [1, 16]$

(b) Processing time depending on foreground density for $g \in \{1, 2, 4\}$

Fig. 4: Processing time in cpp of hole filling with either feature computation or relabeling.

In Table 2, each State-of-the-Art algorithm are compared to one configuration of our new algorithm that computes at least as much. The execution time of our algorithm is close to the best classical foreground CCL algorithm [4] despite computing background labels and adjacency tree in addition. This is possible thanks to our lightweight adjacency-tree computation and the use of segments that allows a *symmetric* computation of BG and FG components.

Our work outperforms both existing B&W algorithms [30, 31] or CCL algorithm with Euler number computation [28]. While we compute much more, Euler computation is faster than dedicated algorithms [25,32]. Our run-based algorithm is faster than pixel-based CCA algorithms, like already observed in [2] with classical LSL. Classical LSL$_{STD}$+F – the base of our algorithm – remains obviously faster. The most noticeable difference is for the maximum processing time: our BW algorithm needs to process twice as much segments than classical LSL.

5. CONCLUSION

In this article, we have introduced a new connected component labeling and analysis algorithm that is able to do in one single pass of the image, both the Euler number computation but also a double foreground and background labeling with the adjacency tree computation. The modified transitive closure algorithm enables an efficient hole processing: holes can be filled and the surrounding connected components are updated on-the-fly. Our approach can easily be adapted to other connected operators like filtering out components based on their statistical features.

Algorithm	Their			Our		
	compute	min	max	compute	min	max
Grana [4]	WR	3.40	25.7	BWAR	3.41	28.9
Diaz [30]	BWAR	18.4	59.0			
He BW [31]	BWER	9.00	79.7	BWAER	3.41	29.0
He combined [28]	WER	16.6	48.0			
He run-based [32]	E	5.54	36.5	BWAE	2.80	15.9
He bit-quad [25]	E	2.87	23.7			
Rosenfeld+F* [2]	WF	4.68	48.0	BWAF	2.80	24.8
He HCS2+F* [2]	WF	4.33	38.6			
LSL$_{STD}$+F [2]	WF	2.69	14.4			

B : Black labeling (BG) A: Adjacency tree F: Feature Computation
W: White labeling (FG) E: Euler number R: Relabel
$*$: standard CCL algorithms transformed into 1-pass CCA algorithms (features only)

Table 2: Performance comparison between State-of-the-Art algorithms ("Their") and this work ("Our"). The "compute" columns show what is computed by the algorithms. "min" and "max" columns show the minimum and maximum processing time in cpp measured for each algorithm.

As far as we know our algorithm is faster than B&W labeling algorithms and algorithms computing features related to hole processing.

6. REFERENCES

[1] A. Rosenfeld and J.L. Platz, "Sequential operator in digital pictures processing," *Journal of ACM*, vol. 13,4, pp. 471–494, 1966.

[2] L. Cabaret and L. Lacassagne, "What is the world's fastest connected component labeling algorithm ?," in *IEEE International Workshop on Signal Processing Systems (SiPS)*, 2014, pp. 97–102.

[3] L. He, X. Ren, Q. Gao, X. Zhao, B. Yao, and Y. Chao, "The connected-component labeling problem: a review of state-of-the-art algorithms," *Pattern Recognition*, vol. 70, pp. 25–43, 2017.

[4] F. Bolelli, S. Allegretti, L. Baraldi, and C. Grana, "Spaghetti labeling: Directed acyclic graphs for block-based connected components labeling," *Transactions on Image Processing*, vol. PP, pp. 1–14, 2019.

[5] A. Hennequin, I. Masliah, and L. Lacassagne, "Designing efficient SIMD algorithms for direct connected component labeling," in *ACM Workshop on Programming Models for SIMD/Vector Processing (PPoPP)*, 2019, pp. 1–8.

[6] D. P. Playne and K. Hawick, "A new algorithm for parallel connected-component labelling on GPUs," *IEEE Transactions on Parallel and Distributed Systems*, 2018.

[7] M. Klaiber, D. Bailey, and S. Simon, "A single cycle parallel multi-slice connected components analysis hardware architecture," *Journal of Real-Time Image Processing*, 2016.

[8] D. Bailey and C. Johnston, "Single pass connected component analysis," in *Image and Vision New Zeland (IVNZ)*, 2007, pp. 282–287.

[9] L. Lacassagne and A. B. Zavidovique, "Light Speed Labeling for RISC architectures," in *IEEE International Conference on Image Analysis and Processing (ICIP)*, 2009.

[10] L. Lacassagne and B. Zavidovique, "Light Speed Labeling: Efficient connected component labeling on RISC architectures," *Journal of Real-Time Image Processing (JRTIP)*, vol. 6, no. 2, pp. 117–135, 2011.

[11] J. W. Tang, N. Shaikh-Husin, U. U. Sheikh, and M. N. Marsono, "A linked list run-length-based single-pass connected component analysis for real-time embedded hardware," *Journal of Real-Time Image Processing*, 2016.

[12] L. He, X. Ren, X. Zhao, B. Yao, H. Kasuya, and Y. Chao, "An efficient two-scan algorithm for computing basic shape features of objects in a binary image," *Journal of Real-Time Image Processing*, vol. 16, pp. 1277–1287, 2019.

[13] L. Cabaret, L. Lacassagne, and D. Etiemble, "Parallel Light Speed Labeling for connected component analysis on multi-core processors," *Journal of Real-Time Image Processing (JRTIP)*, vol. 15, no. 1, pp. 173–196, 2018.

[14] A. Hennequin, Q. L. Meunier, L. Lacassagne, and L. Cabaret, "A new direct connected component labeling and analysis algorithm for GPUs," in *IEEE International Conference on Design and Architectures for Signal and Image Processing (DASIP)*, 2018, pp. 1–6.

[15] D. G. Bailey and M. J. Klaiber, "Zig-zag based single-pass connected components analysis," *Journal of Imaging*, vol. 5,45, pp. 1–26, 2019.

[16] R.E. Tarjan, "Efficiency of good but not linear set union algorithm," *Journal of ACM*, vol. 22,2, pp. 215–225, 1975.

[17] R.E. Tarjan and J. Leeuwen, "Worst-case analysis of set union algorithms," *Journal of ACM*, vol. 31, pp. 245–281, 1984.

[18] Z. Galil and G.F. Italiano, "Data structures and algorithms for disjoint set union problems," *ACM Computing Survey*, vol. 23,3, pp. 319–344, 1991.

[19] K. Wu, E. Otoo, and K. Suzuki, "Optimizing two-pass connected-component labeling algorithms," *Pattern Analysis and Applications*, vol. 12, pp. 117–135, 2009.

[20] F. Manne and M.A. Patwary, "A scalable parallel union-find algorithm for distributed memory computers," in *Parallel Processing and Applied Mathematics*, LNCS 6067 Springer, Ed., 2009, pp. 186–195.

[21] M.A. Patwary, J.R. Blair, and F. Manne, "Experiments on union-find algorithms for the disjoint-set data structure," in *International symposium on experimental algorithms (SEA)*, LNCS 6049 Springer, Ed., 2010, pp. 411–423.

[22] S. B. Gray, "Local properties of binary images in two dimensions," *Transactions on Computers*, vol. 20, 5, pp. 551–561, 1971.

[23] A. Rosenfeld, "Digital topology," *The American Mathematical Monthly*, vol. 28, 8, pp. 621–360, 1979.

[24] L. He, Y. Chao, and K. Suzuki, "A new algorithm for labeling connected-components and calculating the euler number, connected-component number, and hole number," in *International Conference on Pattern Recognition (ICPR)*, 2012, pp. 3099–3102.

[25] B. Yao, L. He, S. Kang, Y. Chao, and X. Zhao, "Bit-quad-based euler number computing," *Transaction on Information and Systems*, vol. E100-D,9, pp. 2197–2204, 2017.

[26] F. Diaz del Rio, H. Molina-Abril, and P. Real, "Computing the component-labeling and the adjacency tree of a binary digital image in near logarithmic-time," in *Workshop on Computation Topology in Image Context(CITIC)*. Springer, 2019, pp. 82–95.

[27] Philippe Salembier and Michael Wilkinson, "Connected operators," *Signal Processing Magazine, IEEE*, vol. 26, pp. 136 – 157, 12 2009.

[28] L. He and Y. Chao, "A very fast algorithm for simultaneously performing connected-component labeling and euler number computing," *Transaction on Image Processing*, vol. 24,9, pp. 2725–2735, 2017.

[29] F Bolelli, M. Cancilla, L. Baraldi, and C. Grana, "Toward reliable experiments on the performance of connected components labeling algorithms," *Journal of Real-Time Image Processing (JRTIP)*, pp. 1–16, 2018.

[30] F. Diaz del Rio, P. Sanchez-Cuevas, H. Molina-Abril, and P. Real, "Parallel connected-component-labeling based on homotopy trees," *Pattern Recognition Letters*, vol. 131, pp. 71–78, 2020.

[31] L. He, Y. Chao, and K. Suzuki, "An algorithm for connected-component labeling, hole labeling and euler number computing," *Journal of Computer Science and Technology*, vol. 28,3, pp. 468–478, 2013.

[32] B. Yao, L. He, S. Kang, X. Zhao, and Y. Chao, "new run-based algorithm for euler number computing," *Pattern Analysis and Applications*, vol. 2, pp. 49–58, 2015.