An efficient run-based Connected Component Labeling algorithm for processing holes

Florian Lemaitre, Nathan Maurice, and Lionel Lacassagne

LIP6, Sorbonne University, CNRS

Abstract. This article introduces a new connected component labeling and analysis algorithm framework that is able to compute in one pass the foreground and the background labels as well as 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 State-of-the-Art shows that this new algorithm can do all these computations faster than all existing algorithms processing foreground and background connected components or holes.

Keywords: Black & white processing, Connected Component Labeling and Analysis, Euler number, Adjacency tree, Hole processing, Hole filling.

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 [26], many algorithms have been developed to accelerate its execution time on CPU [3, 5, 12], SIMD CPU [14, 19], GPU [22] or FPGA [16].

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 [1, 13, 17, 18, 28]. Parallelized algorithms have also been designed [2, 6, 15]. The initial Union-Find algorithm [29] has been also analysed [30] and improved [7] with Decision Tree [31] and various path compression/modification algorithms [20, 21].

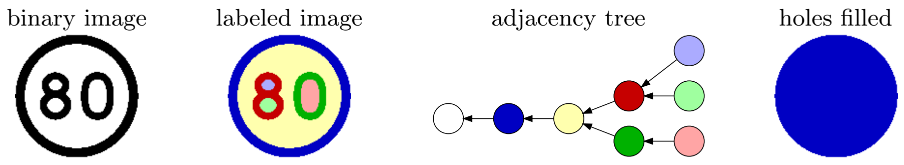

Fig. 1: Example of Black and White labeling with hole filling (FG in black)

Some other features – useful for pattern classification/recognition – are computed by another set of algorithms: the Euler number with Bit-Quads [8], the adjacency (also known as homotopy or inclusion) tree [25] and more recently, foreground (FG) and background (BG) labeling (also known as BW labeling) [10] and hole filling with also improvements in the last decade [23, 32]. Hole filling is an important part of medical image processing [27, 33]. An example of black and white labeling with adjacency tree and hole filling is shown in Figure 1.

Our contribution is a fast algorithm framework to process holes in black and white images. It can compute features of the CCs, the adjacency tree or the euler number of the image, and can fill holes.

This paper is split into the three following parts: Section 1 provides an overview of our new CCL algorithm. The specificities of black&white and hole processing are detailed in Section 2. Section 3 presents a benchmark of existing algorithms and their analysis.

1 General Overview of our new algorithm

We chose to base our new black and white algorithm on the existing LSL algorithm [18] and especially its lastest SIMD implementation, the FLSL algorithm [19]. The reason is two-fold: as the LSL is run-based (segment processing), it is able to compute features very quickly compared to pixel-based algorithms. The second reason is that FLSL is the fastest CCL algorithm currently available [19]. To be noted that FLSL does not explicitly support CCA, thus feature computation had to be back-ported from LSL to this new algorithm.

The LSL algorithm is a CCL/CCA algorithm based on Union-Find structure [29] to build the equivalence relationship between parts of the same connected component. The specificity of LSL is to be run-based: it first computes segments of same class pixels (either foreground of background), and then unifies intersecting segments from consecutive lines. This reduces both the number of temporary labels and the number of “Union” needed to process the image.

BW FLSL needs the following global tables:

- T: Equivalence table (Union-Find structure),
- F: Feature table, encodes the features of each label,
- I: Initial adjacency table, encodes the adjacency tree (explained later).

Figure 2 illustrates the LSL-related table usage on a simple example.

LSL is composed of four steps (Algorithm 1). During the first one, input pixels are read and grouped into segments of same class (foreground or background). This step computes the position of the segments (RLC_i) using semi-open intervals: $RLC_i[er]$ is the position of the first pixel of the er-th segment, whereas $RLC_i[er + 1]$ is the position of the first pixel *after* the er-th segment. This step is taken verbatim from [19].

During the second step, temporary labels are assigned to segments, and segments from current line are “unified” with segments of the previous line. This is done by computing the intersection of current segments with the ones above, and mark them equivalent. The equivalences between labels are recorded in the

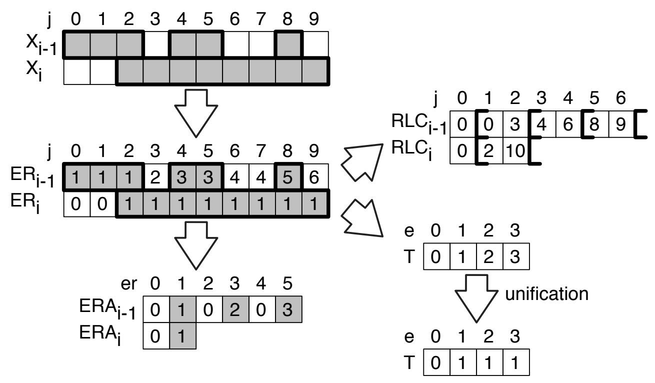

Fig. 2: Tables for the LSL (*ER* is actually not needed anymore).

Algorithm 1: New BW FLSL overview.
This paper contribution is highlighted with gray boxes.

```
1  $ne \leftarrow 1$    ▷ Reset number of labels
2  $I[0] \leftarrow -1$    ▷ Exterior component has no surrounding
3  $F[0] \leftarrow \varnothing$
4  for $i = 0$ to $h - 1$ do
5      RLE($i$)    ▷ Step 1a: Detect segments
6      Unify($i$)    ▷ Step 1b: Merge labels from adjacent line segments
7  Close()    ▷ Step 2: Transitively close the equivalence graph
8  if $relabel$ then
9      for $i = 0$ to $h - 1$ do
10         Relabel($i$)    ▷ Step 3: Write the label image
```

equivalence table T. In addition, when a label is assigned to a segment, the features of this segment is computed and merged with the features of this label.

As for the FLSL, this step actually uses a Finite State Machine that works similarly to a merged sort where the segments of both consecutive lines are iterated together. The detailed implementation of this FSM can be found in Algorithm 2.

The third step is the transitive closure of the equivalence graph: it makes each temporary label point directly to the root. The equivalence trees are flattened. During this step, features from temporary labels are also merged into their root. As will be explained later in this paper (Section 2.2), the hole filling and the Euler number computation are also done during the transitive closure.

The final step is a relabeling step: it produces a labeled image where each pixel is assigned the final root label of its connected component. It is actually a line by line RLE decoder. Like for FLSL, this algorithm is accelerated using an SIMD memset [19]. This step can be skipped if not required, for instance if one is interested only in the connected component features (CCA) without displaying the image of labels.

Algorithm 2: New BW unification (BW FLSL step 1b).
Black and White related processing is highlighted with gray boxes.

```
init:
    $er \leftarrow 0$    ▷ Index of current line segment (relative label)
    $er' \leftarrow 0$    ▷ Index of previous line segment
    $a \leftarrow 0$    ▷ Label of current segment, the first is necessarily the exterior
    $j_0 \leftarrow RLC_i[er]$    ▷ Starting position of current segment
    $j_1 \leftarrow RLC_i[er+1]$    ▷ End position of current segment
    $c_8 \leftarrow 0$    ▷ $c_8 = 1$ if current segment is 8-adjacent, here, BG is 4-adjacent
    ▷ virtual segments allowing to avoid testing for the end of previous line
    $S_0 \leftarrow RLC_{i-1}[ner_{i-1}]$    $S_1 \leftarrow RLC_{i-1}[ner_{i-1}+1]$    ▷ Save past-the-end
    if $ner_{i-1}$ is odd then $RLC_{i-1}[ner_{i-1}] \leftarrow w+1$
    $RLC_{i-1}[ner_{i-1}+1] \leftarrow w+2$
    goto increment previous
new label:
    $T[ne] \leftarrow ne$    ▷ On-the-fly initialization of the equivalence table
    $F[ne] \leftarrow \varnothing$    ▷ On-the-fly initialization of the feature table
    $I[ne] \leftarrow a$    ▷ Initial adjacency: $a$ is the label of previous segment
    $a \leftarrow ne$
    $ne \leftarrow ne+1$
write label:
    $F[a] \leftarrow F[a] \cup \texttt{computeFeatures}(i, j_0, j_1)$
    $ERA_i[er] \leftarrow a$
increment current:
    $er \leftarrow er+1$    ▷ Next segment of current line
    $er' \leftarrow er'-1$    ▷ Previous segment of previous line intersects current segment
    $c_8 \leftarrow c_8 \oplus 1$    ▷ Adjust adjacency for current component
    $j_0 \leftarrow j_1$
    $j_1 \leftarrow RLC_i[er+1]$
    if $er = ner_i$ then goto end
    if $RLC_{i-1}[er'] \geqslant j_1 + c_8$ then goto new label
prolog:
    $a \leftarrow \mathrm{Find}(T, ERA_{i-1}[er'])$
increment previous:
    $er' \leftarrow er'+2$
    if $RLC_{i-1}[er'] \geqslant j_1 + c_8$ then goto write label
unify:
    $e \leftarrow \mathrm{Find}(T, ERA_{i-1}[er'])$
    ▷ Union of the two root labels $e$ and $a$
    if $e \neq a$ then
        if $e < a$ then swap $e, a$
        $T[e] \leftarrow a$
    goto increment previous
end:
    if $ner_i$ is odd and $a \neq 0$ then $T[a] \leftarrow 0$
    $RLC_{i-1}[ner_{i-1}] \leftarrow S_0$    $RLC_{i-1}[ner_{i-1}+1] \leftarrow S_1$    ▷ Restore past-the-end
```

Algorithm 3: New BW Transitive closure (BW FLSL step 2).
Black and White related processing is highlighted with gray boxes.

```
for $e = 0$ to $ne - 1$ do
    $a \leftarrow T[e]$    ▷ ancestor
    if *Hole filling* and $e = a$ then    ▷ If label is root
        $i \leftarrow I[e]$    ▷ label of the surrounding component
        if $T[i] > 0$ then $a \leftarrow i$    ▷ $e$ has a surrounding (is not the exterior)
    if $a < e$ then
        $r \leftarrow T[a]$
        $T[e] \leftarrow r$    ▷ Transitive Closure: $r = T[T[e]]$
        $F[r] \leftarrow F[r] \cup F[e]$    ▷ Feature merge
    else    ▷ $e$ is a root
        $I[e] \leftarrow T[I[e]]$    ▷ point adjacency to root
        if *Euler number computation* then $E[e] \leftarrow 0$
if *Euler number computation* then
    for $e = ne - 1$ to $0$ step $-1$ do
        if $T[e] = e$ then
            $i \leftarrow I[e]$
            $E[i] \leftarrow E[i] + 1 - E[e]$
```

Algorithm 4: New BW Relabeling (BW FLSL step 3).

```
$j_0 \leftarrow RLC_i[0]$    ▷ $j_0$ is 0
for $er = 0$ to $ner_i - 1$ step 1 do
    $e \leftarrow ERA_i[er]$    ▷ provisional label
    $r \leftarrow T[e]$    ▷ final label
    $j_1 \leftarrow RLC_i[er + 1]$
    $Y_i[j_0, j_1[ \leftarrow r$    ▷ Memset
    $j_0 \leftarrow j_1$    ▷ Register rotation
```

The two first steps (RLE encoder and segment unification) are done together and thus require only a single scan of the image. The transitive closure step does not scan the image, but requires to scan the equivalence table holding the relation between temporary labels. The relabeling step, when done, needs a second pass over the image to produce the image of labels.

2 Specificities of black and white labeling and hole processing

In the following, both BG and FG connected components are considered. For the sake of simplicity, a “component” designates a connected component.

2.1 Black and White labeling

Classical LSL does not process background components, but thanks to its segment design and its semi-open interval encoding, it is easily adaptable. Indeed, the end of a foreground segment is the beginning of the following background one, and vice-versa. Thus, no modification to the RLC tables is required. The RLE encoder remains identical.

The unification step needs a few adjustments. First, it iterates over both odd (FG) and even (BG) segments instead of just the odd ones. This requires to adapt the FSM itself. Indeed, the classical FSM needs to handle cases where multiple segments should be skipped because of the lack of intersection. When processing both FG and BG segments, this cannot happen anymore as all segments (FG and BG) are iterated over sequentially. Consequently, we just need to check if the start of the segment on the previous line ($RLC_{i-1}[er']$) is after the end of the segment on current line ($RLC_i[er+1]$). This makes the new FSM actually *simpler* than the one used by the FLSL algorithm.

To process both FG and BG components, we need to consider complementary connectivity: either 8-adjacency for FG and 4-adjacency for BG, or the 4-adjacency for FG and 8-adjacency for the BG. This is done with the c_8 variable (Algorithm 2, line 7) that defines the current connectivity. It is equals to 1 when processing 8-adjacent component and 0 otherwise. Therefore, changing the background connectivity from 4 to 8 can be easily done by setting c_8 to 1 instead of 0 (Algorithm 2, line 7).

Labels are also assigned to background labels, thus, ERA_i does not necessarily have 0 at even indices. The first encoded segment of a line i is always a BG segment, but has zero length if the first pixel of the line is FG.

Like the unification, the relabeling also needs to iterate over both FG and BG segments.

2.2 Holes and adjacency tree computation

Let us introduce some notations about holes. A component C_1 is surrounded by another component C_2 – written $C_1 \sqsubset C_2$ – *iif* all paths from C_1 to the exterior of the image contain at least one pixel from C_2. A hole in a foreground component W is a background component B that is surrounded by W.

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 ($C_1 \sqsubset C_2$). The label e_2 is not necessarily a root label during the execution of the algorithm. $I[e_1]$ equals -1 if $e_1 = 0$, or if e_1 is not a root label ($T[e_1] \neq e_1$). In other words, the table I represents the adjacency tree whose edges are directed according to the surrounding relation. In the following, we consider for the sake of simplicity 8-adjacency for the FG and 4-adjacency for the BG.

We considered two methods to build the adjacency tree and the surrounding relation: detecting *closing pixels* [9], or looking at the adjacency at exterior pixels [23], and more precisely looking at the *initial adjacency*.

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.

The *initial adjacency* method works as follow. 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.When two root labels a and b (with $a < b$) are unified, the initial adjacency $I[b]$ is discarded in favor of $I[a]$ (and $T[b] \leftarrow a$). The order relation 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, only the initial adjacency of the root label remains. The root label being by construction the label of top most pixels, its initial adjacency is necessarily on the exterior of the component. The remaining initial adjacency and speculative surrounding is thus necessarily a true surrounding.

Hole filling is done during the transitive closure (Algorithm 3, lines 3-5). This is done by merging any component that is neither ⓪ nor directly surrounded by ⓪ with their surrounding component. The initial surrounding relation of such a component is transformed into an equivalence relation ($T[e] \leftarrow I[e]$). In fact, arbitrary connected operators can be implemented using the same principle. One would only need to change the criteria to merge a component into its surrounding in order to implement any connected operator.

Euler number is the difference between the number of connected components and the number of holes [8]. Because we have labeled both BG and FG component, it is trivial to compute. In fact, thanks to the adjacency tree, we can even compute the euler number of a component without much effort (Algorithm 3, line 13-17).

2.3 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 ❶, ②, ③, ❹ and ❺. Their initial adjacency is set as their speculative surrounding: ❶ $\sqsubset$ ⓪, ② $\sqsubset$ ❶, ③ $\sqsubset$ ❶, ❹ $\sqsubset$ ② and ❺ $\sqsubset$ ③.

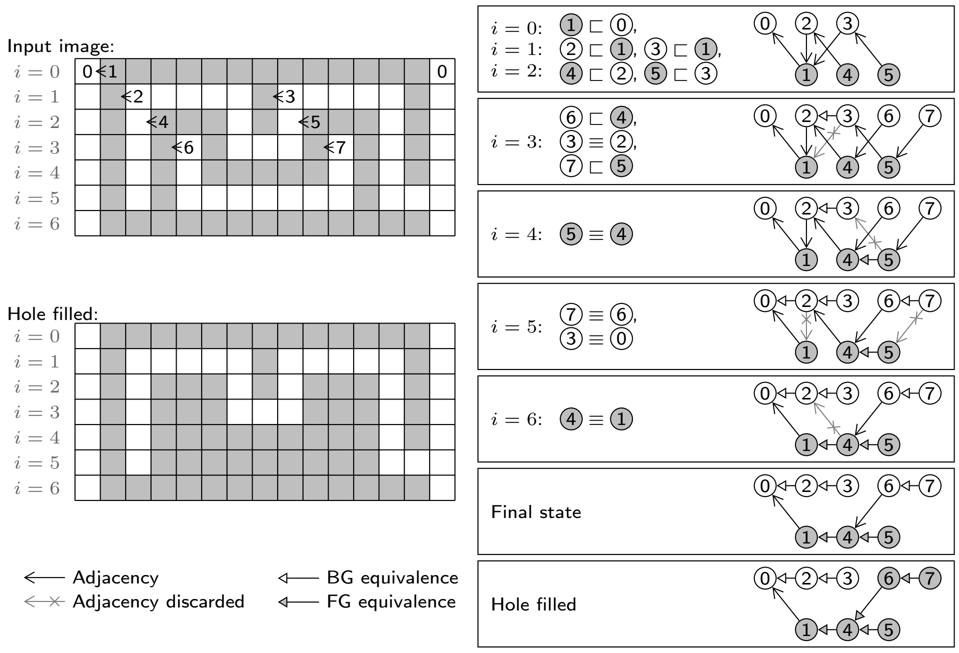

Fig. 3: Step by step example of our new BW labeling focusing on equivalence and adjacency computation.

At $i = 3$, two new labels are created with the following speculative surroundings: ⑥ $\sqsubset$ ❹ and ⑦ $\sqsubset$ ❺. In addition, ③ $\equiv$ ② is detected. Consequently, the speculative surrounding of ③ is discarded in favor of ② $\sqsubset$ ❶.

At $i = 4$, as ❺ $\equiv$ ❹, the speculative surrounding ❺ $\sqsubset$ ③ is discarded.

At $i = 5$, two new equivalences are detected: ② $\equiv$ ⓪ and ⑦ $\equiv$ ⑥. Therefore, 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 $\sqsubset$ ❹ is no more speculative and is actually definitive.

At $i = 6$, the last equivalence ❹ $\equiv$ ❶ is detected and the speculative surrounding ❹ $\sqsubset$ ② is discarded, and the surrounding ❶ $\sqsubset$ ⓪ is kept.

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

3 Benchmark & Performance Analysis

We measured the performance of our algorithms using a protocol similar to [4]. All the algorithms are sequential and no multithreading is used. We tested randomly generated 2048×2048 images with varying density and granularity on a Skylake

Gold 6126 Xeon @2.60GHz. We focus our analysis on $g = 1$ as it is the worst case for run-based algorithms like FLSL. Grana's [3], Diaz' [24] and Lemaitre's [19] CCL algorithms have been ran and measured on this machine. The feature computation with FLSL has been back-ported from classical LSL and was not part of its paper. 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. In addition, we tested multiple variants of our algorithm that computes a subset of *Euler number*, *hole filling*, *feature computation* and *relabel* in order to compare to existing algorithms that do not compute all of them. Especially, the Euler number computation has been implemented for the sole purpose of comparing our new algorithm with the State-of-the-Art. For CCA algorithms, the seven standard features are computed: the surface, the bounding box (x_{min}, x_{max}, y_{min}, y_{max}) and the first statistical raw moments (S_x, S_y).

		min	max
BW + Adjacency	(BWA)	0.36	12.7
+Euler number	(E)	+ 0	+ 0.29
+Hole Filling	(H)	+ 0	+ 0.50
+Feature Computation	(F)	+ 0	+ 8.59
+Relabeling	(R)	+ 0.59	+ 3.66

Table 1: Processing time in cpp of the core part of our new BW FLSL as well as the extra processing time for extra computation. Minimal and maximal times are given for 2048×2048 random images. Min time reached for $d = 0\%$ and max time reached for $g = 1$ and $d \simeq 40\%$.

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 (BWA). The next lines provide the extra times for extra computations like *Euler number* (E) or *hole filling* (H), *feature computation* (F) and *relabeling* (R). The extra times are the best (min) and worst (max) case we measured for doing these extra computations. One can then estimate the total processing time for a given combination of {E, H, F, R} just by adding the associated 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.

In the worst case, Feature computation adds a large extra time because the seven features need to be written for each and every labels which highly increases the number of memory accesses. The minimal extra processing time for *relabeling* is non-zero as a second scan of the image is required to produce the output image of labels. Therefore, its computation should be avoided if not required. But thanks to the SIMD RLE decoder, this processing remains fast in the worst case. One can also notice that *Euler number* computation and *hole filling* are inexpensive using our approach.

algorithm		compute	min	avg	max
He bit-quad		E	2.87	14.0	23.7
He BW	(with R)	BWER	16.5	51.0	79.6
Diaz	(with R)	BWAR	18.4	36.8	59.0
Spaghetti(FG) + Spaghetti(BG)	(with R)	BWR	5.76	32.7	51.2
FLSL(FG+R) + FLSL(BG+R)	(with R)	BWR	2.34	17.0	24.4
FLSL(FG+F) + FLSL(BG+F)	(with F)	BWF	1.68	20.5	30.3
FLSL(BG) + FLSL(FG+F)	(with F)	WF**H**	1.89	19.8	33.7
BW FLSL+ER		BWAER	0.98	10.0	14.6
BW FLSL+F		BWAF	0.38	13.0	20.0
BW FLSL+FH		BWAF**H**	0.38	14.0	20.7

B: Black labeling (BG)	A: Adjacency tree	F: Feature Computation	R: Relabel
W: White labeling (FG)	E: Euler number	**H**: Hole filling	

Table 2: Performance comparison between State-of-the-Art algorithms and this work (BW FLSL). The "compute" column shows what is computed. Processing time in cycle/pixel for 2048×2048 random images at $g = 1$.

In Table 2, each State-of-the-Art algorithm are compared to one configuration of our new algorithm that computes at least as much. Our base algorithm BW FLSL+ER that computes the adjacency tree, the Euler number relabels the output image is faster than any black and white CCL algorithm. In average it is 5.1× faster than He BW [11] and 3.6× faster than Diaz [24]. It is even faster than He bit-quad [32] whose sole purpose is to compute the Euler number of the image. This speed difference comes mainly from the efficient use of runs, the use of SIMD, and the low overhead computation of the adjacency tree. Even though a single execution of FLSL is faster than BW FLSL, FLSL process only a FG components. Thus, two executions of FLSL (and Spaghetti) are needed to compute any hole related property.

Therefore, BW FLSL is from 3.3× up to 5.9× faster than Spaghetti and from 1.4× up to 1.7× faster than FLSL to have both black and white labels or holes filled. In addition, BW FLSL computes the adjacency tree with no extra cost.

4 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 whereas features are computed to take this change into account.
As far as we know our new algorithm outperforms all published algorithms for BW labeling and hole processing. In addition, it is easily tunable: its structure can be adapted to other connected operators like filtering out components based on their statistical features.

References

1. Bailey, D., Johnston, C.: Single pass connected component analysis. In: Image and Vision New Zeland (IVNZ). pp. 282–287 (2007)
2. Bailey, D.G., Klaiber, M.J.: Zig-zag based single-pass connected components analysis. Journal of Imaging **5,45**, 1–26 (2019)
3. Bolelli, F., Allegretti, S., Baraldi, L., Grana, C.: Spaghetti labeling: Directed acyclic graphs for block-based connected components labeling. Transactions on Image Processing **PP**, 1–14 (2019)
4. Bolelli, F., Cancilla, M., Baraldi, L., Grana, C.: Toward reliable experiments on the performance of connected components labeling algorithms. Journal of Real-Time Image Processing (JRTIP) pp. 1–16 (2018)
5. Cabaret, L., Lacassagne, L.: What is the world's fastest connected component labeling algorithm ? In: IEEE International Workshop on Signal Processing Systems (SiPS). pp. 97–102 (2014)
6. Cabaret, L., Lacassagne, L., Etiemble, D.: Parallel Light Speed Labeling for connected component analysis on multi-core processors. Journal of Real-Time Image Processing (JRTIP) **15**(1), 173–196 (2018)
7. Galil, Z., Italiano, G.: Data structures and algorithms for disjoint set union problems. ACM Computing Survey **23,3**, 319–344 (1991)
8. Gray, S.B.: Local properties of binary images in two dimensions. Transactions on Computers **20, 5**, 551–561 (1971)
9. He, L., Chao, Y.: A very fast algorithm for simultaneously performing connected-component labeling and euler number computing. Transaction on Image Processing **24,9**, 2725–2735 (2017)
10. He, L., Chao, Y., Suzuki, K.: 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). pp. 3099–3102 (2012)
11. He, L., Chao, Y., Suzuki, K.: An algorithm for connected-component labeling, hole labeling and euler number computing. Journal of Computer Science and Technology **28,3**, 468–478 (2013)
12. He, L., Ren, X., Gao, Q., Zhao, X., Yao, B., Chao, Y.: The connected-component labeling problem: a review of state-of-the-art algorithms. Pattern Recognition **70**, 25–43 (2017)
13. He, L., Ren, X., Zhao, X., Yao, B., Kasuya, H., Chao, Y.: An efficient two-scan algorithm for computing basic shape features of objects in a binary image. Journal of Real-Time Image Processing **16**, 1277–1287 (2019)
14. Hennequin, A., Masliah, I., Lacassagne, L.: Designing efficient SIMD algorithms for direct connected component labeling. In: ACM Workshop on Programming Models for SIMD/Vector Processing (PPoPP). pp. 1–8 (2019)
15. Hennequin, A., Meunier, Q.L., Lacassagne, L., Cabaret, L.: 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). pp. 1–6 (2018)
16. Klaiber, M., Bailey, D., Simon, S.: A single cycle parallel multi-slice connected components analysis hardware architecture. Journal of Real-Time Image Processing (2016)
17. Lacassagne, L., Zavidovique, A.B.: Light Speed Labeling for RISC architectures. In: IEEE International Conference on Image Analysis and Processing (ICIP) (2009)

18. Lacassagne, L., Zavidovique, B.: Light Speed Labeling: Efficient connected component labeling on RISC architectures. Journal of Real-Time Image Processing (JRTIP) **6**(2), 117–135 (2011)
19. Lemaitre, F., Hennequin, A., Lacassagne, L.: How to speed connected component labeling up with simd rle algorithms. In: ACM Workshop on Programming Models for SIMD/Vector Processing (PPoPP) - to appear. pp. 1–8 (2020)
20. Manne, F., Patwary, M.: A scalable parallel union-find algorithm for distributed memory computers. In: Springer, L.. (ed.) Parallel Processing and Applied Mathematics. pp. 186–195 (2009)
21. Patwary, M., Blair, J., Manne, F.: Experiments on union-find algorithms for the disjoint-set data structure. In: Springer, L.. (ed.) International symposium on experimental algorithms (SEA). pp. 411–423 (2010)
22. Playne, D.P., Hawick, K.: A new algorithm for parallel connected-component labelling on GPUs. IEEE Transactions on Parallel and Distributed Systems (2018)
23. del Rio, F.D., Molina-Abril, H., Real, P.: 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). pp. 82–95. Springer (2019)
24. del Rio, F.D., Sanchez-Cuevas, P., Molina-Abril, H., Real, P.: Parallel connected-component-labeling based on homotopy trees. Pattern Recognition Letters **131**, 71–78 (2020)
25. Rosenfeld, A.: Digital topology. The American Mathematical Monthly **28, 8**, 621–360 (1979)
26. Rosenfeld, A., Platz, J.: Sequential operator in digital pictures processing. Journal of ACM **13,4**, 471–494 (1966)
27. Somasundaram, K., Kalaiselvi, T.: A method for filling holes in objects of medical images using region labeling and run length encoding schemes. In: National conference on image processing (NCIMP). pp. 110–115 (2010)
28. Tang, J.W., Shaikh-Husin, N., Sheikh, U.U., Marsono, M.N.: A linked list run-length-based single-pass connected component analysis for real-time embedded hardware. Journal of Real-Time Image Processing (2016)
29. Tarjan, R.: Efficiency of good but not linear set union algorithm. Journal of ACM **22,2**, 215–225 (1975)
30. Tarjan, R., Leeuwen, J.: Worst-case analysis of set union algorithms. Journal of ACM **31**, 245–281 (1984)
31. Wu, K., Otoo, E., Suzuki, K.: Optimizing two-pass connected-component labeling algorithms. Pattern Analysis and Applications **12**, 117–135 (2009)
32. Yao, B., He, L., Kang, S., Chao, Y., Zhao, X.: Bit-quad-based euler number computing. Transaction on Information and Systems **E100-D,9**, 2197–2204 (2017)
33. Zhao, H., Chen, Z.X.: A simple hole filling algorithm for binary cell images. In: Applied Mechanics and Materials. vol. 433, pp. 1715–1719. Trans Tech Publ (2013)