What Is the World's Fastest Connected Component Labeling Algorithm?

Laurent Cabaret Lionel Lacassagne
Laboratoire de Recherche en Informatique
Univ Paris-Sud, F-91405 Orsay, France
email: firstname.name@lri.fr

***Abstract*—Optimizing connected component labeling is currently a very active research field. Some teams claim to have design the fastest algorithm ever designed. This paper presents a review of these algorithms and a enhanced benchmark that improve classical random images benchmark with a varying granularity set of random images in order to become closer to natural image behavior.**

INTRODUCTION

Binary Connected Component Labeling (CCL) algorithms deal with graph coloring and transitive closure computation. CCL algorithms play a central part in machine vision, because they often constitute a mandatory step between low-level image processing (filtering) and high-level image processing (recognition, decision). As such, CCL algorithms have a lot of applications and derivate algorithms like convex hull computation, hysteresis filtering or geodesic reconstruction.

Designing a new algorithm is challenging both from considering the overwhelming literature and from the very performance of best existing algorithms. Goals could be a faster algorithm on some class of computer architecture or minimizing the number of over-created labels or the smallest theoretical complexity. Yet another issue is to be most predictable.

Now, from the current state of the computing technology, reaching decent performances in actuality requires for CCL algorithms to take into account two specificities/capacities of current General Purpose Processors (GPP): the processor pipeline and its cache memories. That amounts to minimize conditional statements (like tests and comparisons) to reduce the number of pipeline stalls and limit random sparse (typically vertical) memory accesses, to lower cache misses.

As it is an intermediate level algorithm, it processes the output data coming from low level algorithms (binary segmentation, ...) and provides abstract input data to other intermediate or high level (decision) algorithms. Usually, such abstract data also called *features* are the boundary of bounding rectangle (for target tracking) and the first order statistical moments (surface, centroid, orientation, ...). So if a standalone CCL algorithm can be considered at first step, the couple "CCL + features computation" is the procedure to be actually evaluated at end.

Our contribution consists of three elements:

- an enhanced benchmark that incorporates random images with different granularities (fig. 1). That can be seen as a bridge between classical random images of density and data base images,
- a benchmark with all state-of-the-art algorithms, as previous articles or reviews on CCL *forget* some algorithms,
- an analysis of the duration of each stage (labeling, transitive closure, relabeling) that helps to understand the global performance of each algorithm and especially the feature computation part that is usually not addressed by other articles.

This paper is organized as follows: the first section describes the modern algorithms that claim to be the world fastest, the second section presents the benchmarks, the third section provide an analysis of algorithm's performances at an overall scale, the fourth section provide an in-depth algorithm's step duration analysis.

I. CONNECTED COMPONENT LABELING ALGORITHMS

Historical algorithms were designed by pioneers like Rosenfeld [14], Haralick [4] and Lumia [10] who designed pixel-based algorithms, and Ronse [13] for run-based algorithm. Modern algorithms derive from the historical ones and try to make improvements by replacing some components by a more efficient one. An extensive bibliography can be found in [5] and [16]. Except Contour Tracing algorithms [1] that is aesthetic but inefficient, all modern algorithms are two-passes (or less) algorithms, none is a data-dependent multi-pass algorithm. They share the same three steps:

- first labeling, that assigns a temporary/provisional label to each pixel and builds labels equivalence,
- label equivalences solving, that is to compute the transitive closure of the graph associated to the label equivalence table,
- final labeling, to replace temporary label by the final label (usually the smallest one of the component).

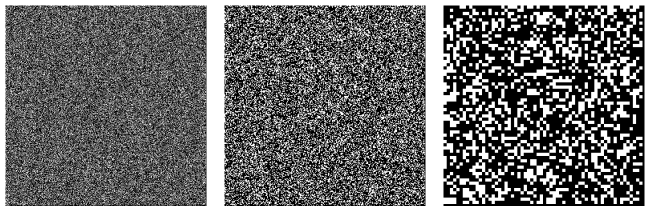
(a) $g = 1$ (b) $g = 2$ (c) $g = 8$

Fig. 1. random images with density = 35% at granularity $g \in 1, 2, 8$

They differ on three points: the mask topology, the number of tests for a given mask to find out the label and the equivalence management algorithm.

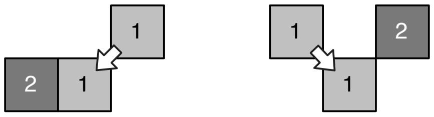

Fig. 2. minimal 8-connected basic patterns generating temporary labels: stair (left) and concavity (right)

Using $Rosenfeld$ mask (fig. 3), only two basic patterns trigger label creation (fig. 2), whatever the connectivity (here 8-connectivity). The first one is the *stair*. It is responsible for the unnecessary provisional label created by pixel-based algorithms like $Rosenfeld$'s one. The second one is the *concavity*. From the neighborhood that founds CCL, it is obvious that the label creation can not be avoided.

As figure 5 and figure 6 show, the execution time is not directly correlated to the total amount of final labels, but to the number of stairs and concavities that generates equivalence building, so one way to improve CCL algorithms is to widen the label mask. That leads to block-based algorithms (fig. 3) like HCS$_2$ [7] and Grana [3] that respectively compute 2 and 4 labels from 6-pixel and 16-pixel neighborhood. One the opposite way, RCM [8] introduces a mask with only 3 neighbors in order to reduce the amount of tests. $Grana$ mask can detect some concavities and avoid label creation if these concavities are small enough to entirely fit in the mask. But the only way to prevent label creation from *stairs* is to use a run-based algorithm like HCS [6] or Light-Speed Labeling (LSL) [9] that first detect the pixel adjacency in the neighborhood before to assign a label to the run. The LSL uses a tricky line-relative labeling to generate RLC coding to directly find adjacent runs on the previous line whereas HCS has to perform a test on every pixel to decide to continue to propagate a label or to perform an equivalence.

The second point to enhance algorithm efficiency is to reduce the number of tests. A *decision tree* (DT) [16] reduces the amount of labels to test to find out the value to assign to the current label based on mask topology. For pixel-based algorithms, it decreases the complexity of the 8-connectivity to the 4-connectivity one. For block-based algorithm, DT is mandatory. Another way to reduce complexity is to perform *path-compression* (PC) [2]. It is a step added to the Union-Find algorithm to perform a transitive closure in climbing up to the root of the equivalence. It has been proven that PC make the Union-Find complexity to grow with the inverse of Ackermann function [15].

Finally the third point is the equivalence management algorithm. $Rosenfeld$ algorithm uses Union-Find algorithm and the associated table to store the equivalences. An alternative approach with three tables has been proposed by [5] now referenced as Suzuki equivalence tables. R holds the root of each set of components, T the *tail* of each equivalence and N the next-equivalent label. As table R holds the root of each label, the transitive closure of the equivalence table is not done at the end, but on the fly, at each equivalence.

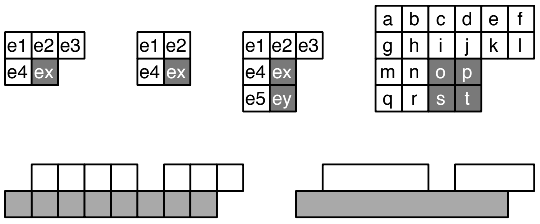

Fig. 3. masks: first line: $Rosenfeld$, RCM, HCS_2 and $Grana$, second line: HCS and LSL

The configurations of the 8-connected algorithms that are benchmarked are :

- $Rosenfeld$: original algorithm improved with DT+PC,
- $Suzuki$: $Rosenfeld$ mask with Suzuki tables management that we improved with DT,
- RCM: pixel-based algorithm with Suzuki management,
- HCS_2: block-based algorithm with Suzuki management,
- $Grana$: block-based algorithm with 128-stage decision tree,
- HCS: run-based algorithm with Suzuki management,
- LSL: run-based algorithm with Union-Find management with two variants: LSL_{STD} and LSL_{RLE}.

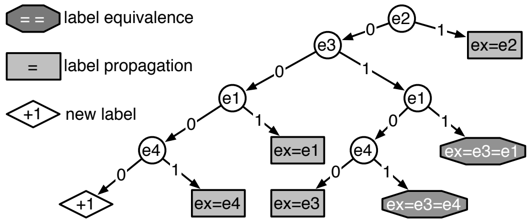

Fig. 4. 8-connected Decision Tree

II. Benchmarks

We present here, the images and processors used for benchmarking. We also provide a qualitative analysis of temporary labels creation.

Usually papers evaluate CCL performance first with random images (varying pixel density from 0% to 100%) for hard-to-label benchmark and secondly with image data base. But data base can be biased and then may favor some algorithms. As we want our benchmark to be as fair as possible (quite difficult with data-dependent algorithms) we decided to select Mersenne Twister MT19937 [11] to control random number generation and to extend random images by changing the pixel granularity. Initial random image has a granularity of 1. Then we create g-random images whose block of pixels have a size of $g \times g$ (Fig. 1), with $g \in [1 : 16]$.

The figure 5 provide the temporary labels distribution for granularity $g \in \{1, 4\}$ for pixel-based, run-based and $Grana$ algorithms (red, magenta and blue). The number of final labels (green), concavities (cyan) and stairs (orange) is also provided.

First, if we compare run-based and pixel-based label distribution, we can see that run-based curve has always the same behavior (close to the final label curve), contrary to the pixel-based curve. The reason is that the amount of concavities is proportionally constant (from one granularity to another one) to the number of final label. For $g \geq 2$, it appears that the amount of stairs becomes bigger than concavities, then the pixel-based also proportionally generates more temporary labels than for $g = 1$. That is the reason why run-based algorithms have a better execution time when g is growing: they avoid more and more label creation.

Concerning $Grana$ algorithm, it generates quite the same number of temporary labels for $g = 1$ than pixel-based ones. For $g = 2$ it comes closer to run-based algorithms as its wide mask avoids many temporary labels. But for $g \geq 4$, its wide mask does not avoid label creation, as 4-pixel wide stairs and concavities are beyond the pixel's neighborhood.

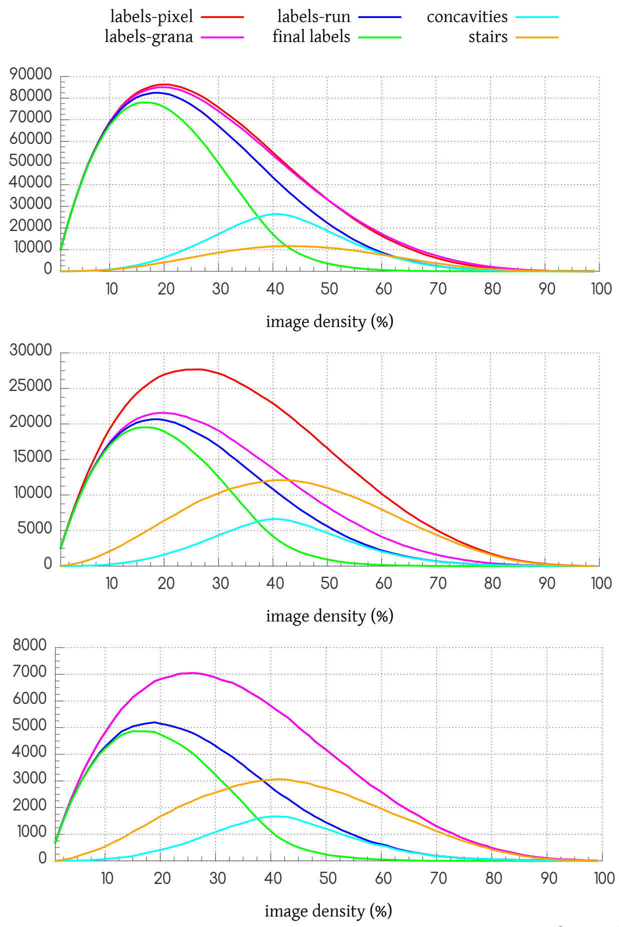

Fig. 5. Distribution of labels, concavities and stairs vs for $g \in \{1, 2, 4\}$

The Standard Image Data-Base (SIDBA) has been used for natural image labeling. Gray-scaled images have been automatically binarized with Otsu algorithm [12]. For both random images and natural ones, we provide the *cpp* (cycle per pixel) of each algorithm, with or without *features* computation (FC). The features extracted for each component are: the bounding box $([x_{min}, x_{max}] \times [y_{min}, y_{max}])$ and the first statistical moments (S, S_x and S_y). The benchmarks (except fig.8) were done on a Sandy-Bridge i7-2400 running a Debian 7.5 /64-bit linux and with ICC 14.0.1.

III. Results and Analysis

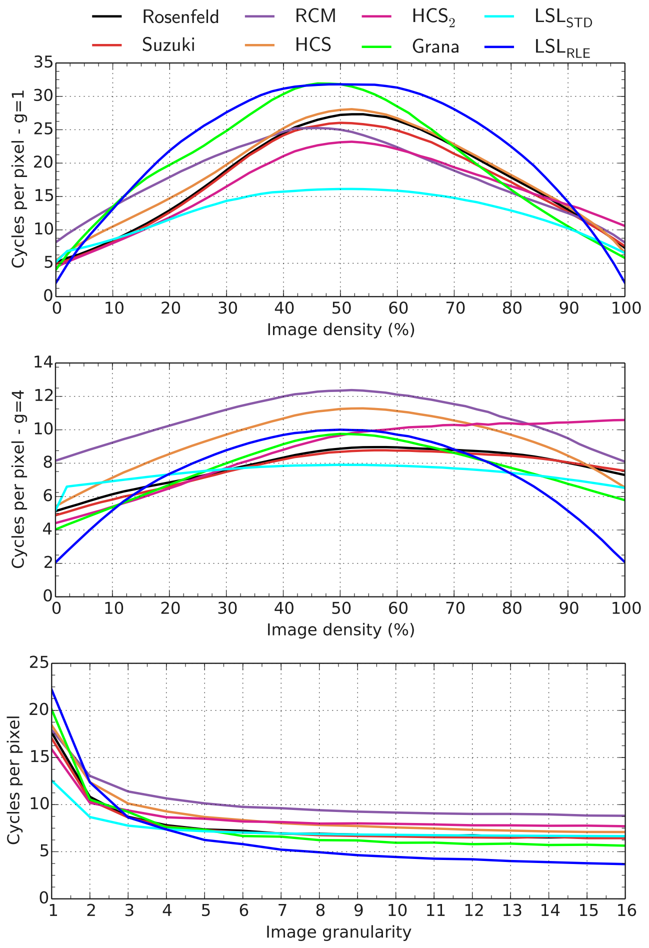

Fig. 6. *cpp* for granularity $g \in \{1, 4\}$ and average *cpp* vs granularity

a) Density behavior:

Figure 6 shows us that algorithm curves, except HCS_2, are symmetrical about their maximum value. The abscissas of the maximum values are contained in the $[45\%; 55\%]$ area depending on algorithm. Concavities and stairs are an explanation (fig. 5), they lead to temporary label creation and labels merging, they also increase the probability of more tests in the decision tree (e.g., stair imposes to traverse all the DT graph until the label creation node "+1" (fig. 4)) and doing so increase *cpp*. As described in [6], HCS_2 algorithm make no usage of decision tree and so, it needs to load the neighborhood's labels for each pixel to label. Doing so, it is not able to reduce *cpp* when density grows above 50%.

One can observe that when the number of stairs and concavities decrease (g comes higher) the density curves tend to flatten.

b) Granularity influence: Table I and figure 6 describe the behavior of algorithms with different granularity images.

TABLE I
AVERAGE *cpp* ACCORDING TO GRANULARITY WITH/WITHOUT FC

	granularity				
	$g = 1$	$g = 2$	$g = 4$	$g = 8$	$g = 16$
algorithms	*Without FC*				
$Rosenfeld$	*17.65*	*10.83*	*7.81*	*6.91*	*6.52*
$Suzuki$	*17.04*	*10.47*	*7.68*	*6.76*	*6.44*
RCM	*17.93*	*13.05*	*10.66*	*9.41*	*8.81*
HCS	*18.42*	*12.38*	*9.28*	*7.84*	*7.09*
HCS_2	*15.88*	*10.19*	*8.66*	*7.99*	*7.70*
$Grana$	*20.10*	*10.54*	*7.60*	*6.23*	*5.64*
LSL_{STD}	**12.55**	**8.68**	*7.38*	*6.87*	*6.65*
LSL_{RLE}	*22.21*	*12.37*	**7.36**	**4.94**	**3.68**
algorithms	*With FC*				
$Rosenfeld$	*31.84*	*21.99*	*17.45*	*15.62*	*14.68*
$Suzuki$	*31.14*	*21.63*	*17.21*	*15.35*	*14.49*
RCM	*32.00*	*24.06*	*20.03*	*17.87*	*16.84*
HCS	*31.91*	*22.90*	*18.36*	*16.05*	*14.96*
HCS_2	*30.10*	*21.66*	*18.77*	*17.28*	*16.51*
$Grana$	*34.34*	*22.00*	*17.76*	*15.55*	*14.46*
LSL_{STD}	**13.28**	**8.10**	*6.13*	*5.33*	*4.96*
LSL_{RLE}	*16.77*	*8.34*	**4.95**	**3.52**	**2.73**

The main trend is that when g grows *cpp* drops. First quickly $[\times 0.53; \times 0.73]$ for $g \in [1{:}2]$, and then slowly $[\times 0.30; \times 0.77]$ for $g \in [2{:}16]$. One can notice that LSL_{RLE} is the most accelerated when granularity grows while LSL_{STD} is the most regular. It comes from their construction as explained in [9]. LSL_{RLE} is obviously inefficient for $g = 1$ because of its run-length encoding kernel. RCM performs better on $g = 1$ than other granularities, this is due to the smaller number of tests it performs compared to $Rosenfeld$ which is an efficient strategy for unstructured data.

LSL algorithms are the most efficient: LSL_{STD} is the fastest for $g \in [1 : 3]$ and LSL_{RLE} is the fastest for $g \in [4 : 16]$ followed by $Grana$, $Suzuki$, LSL_{STD}, $Rosenfeld$, HCS, HCS_2 and RCM.

c) Features influence: When FC is activated, LSL_{STD} and LSL_{RLE} outperform all others algorithms (table I and fig. 7). This is mostly due to the efficiency of run-length FC that saves comparisons and memory accesses – compared to pixel algorithms [9]. Moreover for LSL algorithms, features are computed on-the-fly and before relabeling instead of after relabeling for pixel-based algorithms, the relabeling pass is unnecessary for LSL. The ranking is LSL_{RLE} and LSL_{STD}, $Suzuki$, $Grana$, $Rosenfeld$, HCS, HCS_2 and RCM.

FC increases every other algorithms *cpp* depending on g (even if the number of pixels is constant for a given density) from $+7.8$*cpp* up to $+14.2$*cpp*. Those variations are explained by the structure of the image (fig. 5): if granularity is low there is more labels than if granularity is high, so more sparse and distant memory accesses will happen, that lead to different values of cache hits or cache misses.

Table II provides relative *cpp* ratios between the fastest algorithm and the other ones for different g values. The ratio increases with g. When FC is activated (considering structured images), LSL_{RLE} is $\times 3.48$ faster than $Suzuki$ for $g = 4$. This ratio increases to $\times 5.30$ for $g = 16$.

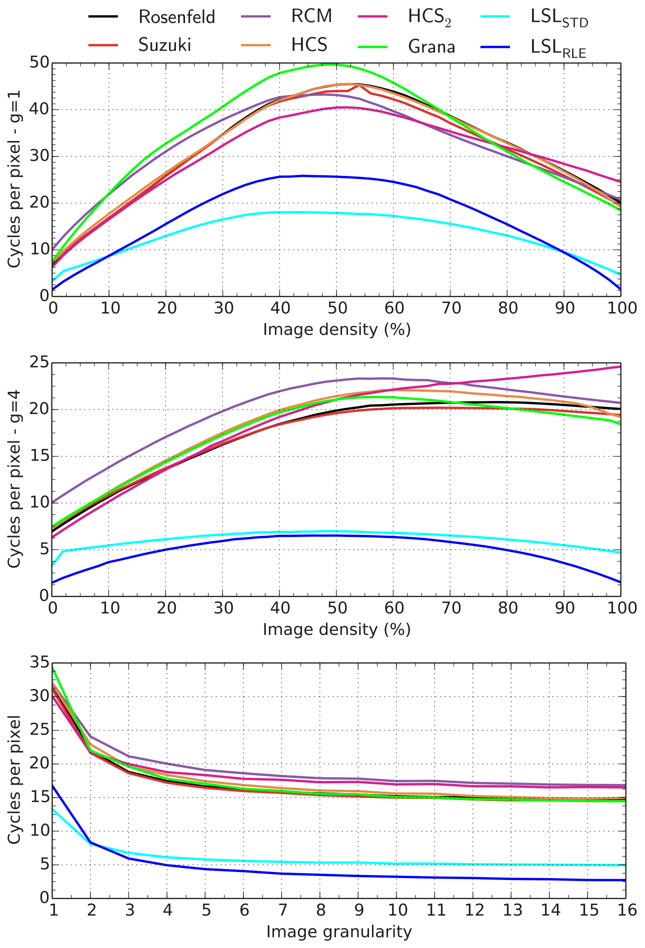

Fig. 7. *cpp* over density for granularity $g \in \{1, 4\}$ and average *cpp* over granularity with FC

TABLE II
cpp RATIO BETWEEN THE FASTEST ALGORITHM AND THE OTHER ONES

	granularity				
	$g = 1$	$g = 2$	$g = 4$	$g = 8$	$g = 16$
algorithms	*without FC*				
$Rosenfeld$	*1.41*	*1.25*	*1.06*	*1.40*	*1.77*
$Suzuki$	*1.36*	*1.21*	*1.04*	*1.37*	*1.75*
RCM	*1.43*	*1.50*	*1.45*	*1.90*	*2.39*
HCS	*1.47*	*1.43*	*1.26*	*1.59*	*1.92*
HCS_2	*1.26*	*1.17*	*1.18*	*1.62*	*2.09*
$Grana$	*1.60*	*1.21*	*1.03*	*1.26*	*1.53*
LSL_{STD}	**1.00**	**1.00**	*1.00*	*1.39*	*1.81*
LSL_{RLE}	*1.77*	*1.42*	**1.00**	**1.00**	**1.00**
algorithms	*with FC*				
$Rosenfeld$	*2.40*	*2.71*	*3.53*	*4.44*	*5.38*
$Suzuki$	*2.34*	*2.67*	*3.48*	*4.36*	*5.31*
RCM	*2.41*	*2.97*	*4.05*	*5.08*	*6.17*
HCS	*2.40*	*2.83*	*3.71*	*4.56*	*5.48*
HCS_2	*2.27*	*2.67*	*3.79*	*4.91*	*6.05*
$Grana$	*2.58*	*2.72*	*3.59*	*4.42*	*5.30*
LSL_{STD}	**1.00**	**1.00**	*1.24*	*1.51*	*1.82*
LSL_{RLE}	*1.26*	*1.03*	**1.00**	**1.00**	**1.00**

d) Real case images: SIDBA natural images database benchmark confirm random images conclusion.

We give the results for each algorithm (table III) with min, average and max values for processing time and *cpp*, for direct comparison with others articles results. With FC, LSL_{RLE} is first followed by LSL_{STD}, $Grana$, $Suzuki$, HCS, $Rosenfeld$, RCM and HCS_2. One can notice that LSL_{STD} is extremely stable in execution time on all images: the variation is 0.12 ms (and 0.21 ms with FC).

TABLE III
EXECUTION TIME AND *cpp* ON SIDBA WITH/WITHOUT FC

	time (ms)			*cpp*		
	min	avg	max	min	avg	max
algorithms	*Without FC*					
$Rosenfeld$	*0.93*	*1.08*	*1.20*	*6.62*	*7.64*	*8.51*
$Suzuki$	*0.90*	*1.04*	*1.16*	*6.36*	*7.35*	*8.18*
RCM	*1.02*	*1.19*	*1.34*	*7.21*	*8.43*	*9.52*
HCS	*0.87*	*1.06*	*1.24*	*6.15*	*7.48*	*8.79*
HCS_2	*1.08*	*1.23*	*1.43*	*7.63*	*8.75*	*10.15*
$Grana$	*0.82*	*1.01*	*1.23*	*5.81*	*7.18*	*8.68*
LSL_{STD}	*0.93*	*0.98*	*1.05*	*6.58*	*6.96*	*7.45*
LSL_{RLE}	**0.36**	**0.67**	**1.02**	**2.55**	**4.72**	**7.24**
algorithms	*With FC*					
$Rosenfeld$	*1.85*	*2.12*	*2.44*	*13.10*	*14.98*	*17.28*
$Suzuki$	*1.81*	*2.07*	*2.37*	*12.85*	*14.67*	*16.76*
RCM	*1.96*	*2.24*	*2.53*	*13.91*	*15.86*	*17.95*
HCS	*1.78*	*2.09*	*2.37*	*12.61*	*14.81*	*16.77*
HCS_2	*1.93*	*2.27*	*2.70*	*13.68*	*16.09*	*19.16*
$Grana$	*1.75*	*2.07*	*2.37*	*12.42*	*14.67*	*16.78*
LSL_{STD}	*0.65*	*0.76*	*0.86*	*4.63*	*5.37*	*6.11*
LSL_{RLE}	**0.26**	**0.45**	**0.66**	**1.85**	**3.17**	**4.69**

e) Architecture influence - From Conroe to Haswell: Figure 8 presents the performance of the algorithms (*cpp*) on six Intel processors for SIDBA data base. The turning point was the Nehalem with the abandon of the FSB bus and its replacement by more efficient busses (DMI on core-i7 and QPI on Xeon). If *cpp* is quite constant, new processors are faster thanks to a higher clock frequency. From an algorithmic point of view, results are very similar on all processors. The algorithms can be split into three groups of performance: first $\{LSL_{STD}, LSL_{RLE}\}$ then $\{Rosenfeld, Suzuki, HCS, Grana\}$ and finally $\{RCM, HCS_2\}$.

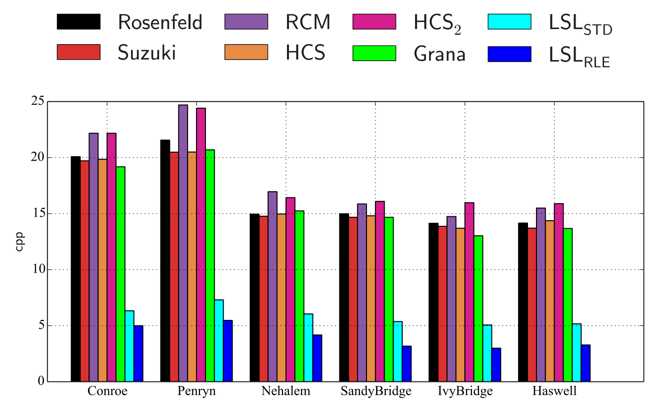

Fig. 8. Average *cpp* for six Intel processors for SIDBA with FC

IV. IN-DEPHT ALGORITHM TIME SLICING ANALYSIS

a) Random Images: In order to well understand the time distribution between each steps of CCL algorithm, we monitored them (fig. 9 & fig. 10). One can see that labeling and features computations parts are very similar for all pixel-based algorithms, and have quite the same duration that is by far longer that the relabeling part.

The LSL_{STD} version has a quite *constant* relabeling part like pixel-based versions, as it is done one pixel by one pixel (data independent).

The FC part is by far smaller for both LSL thanks to run-length coding. First min and max operations are done only twice, for the beginning and the end of a run, instead of as many times as there are pixels in the run. Secondly, the statistical moment can be calculated with the begin and the end indexes. For a given run of interval $[j_0, j_1]$, $S = j_1 - j_0 + 1$ and $S_x = \phi(j_1) - \phi(j_0 + 1)$, $S_y = i \times S$, with ϕ the first Bernoulli polynomial.

For LSL_{RLE}, the relabeling is not constant and can be more time consuming than pixel-based relabeling because of the run-lenght encoding needs to decompress the information and so performs more memory accesses.

As all the information is in the equivalence table and in the features structures, the relabeling is not required for LSL and so it is not taken into account for the calculus of the algorithm duration in tables and figures of the section III and for figures 12 and 11. The same modification can be done for pixel algorithms, but, as far as we known, the other authors have never mentioned this modification as they do not address FC optimization in their articles and only focus on labeling part.

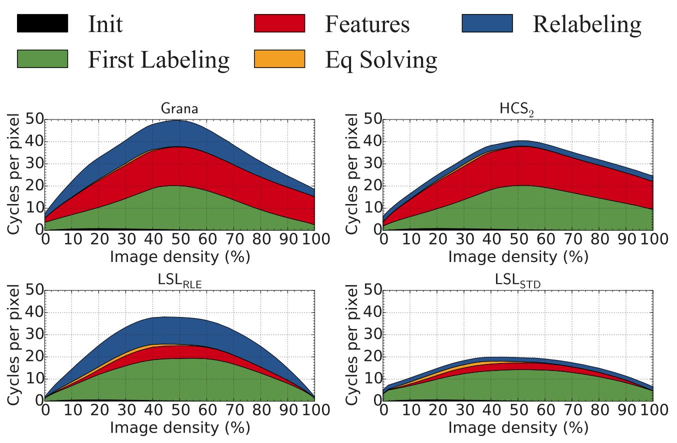

Fig. 9. Algorithms time-slicing for $g = 1$

When g grows, the labeling and relabeling parts of LSL_{STD} are very similar to pixel-based algorithms while as there are fewer regions, these parts for LSL_{RLE} become faster. For pixel-based algorithms, as the duration of these parts decreases, FC becomes the main part of the total computation time, whereas for LSL versions FC is so fast that it is almost invisible on the graph. For that reason, when FC is taken into account (in a real application), no pixel-based algorithm can match run-length-based algorithm performance like LSL_{RLE}.

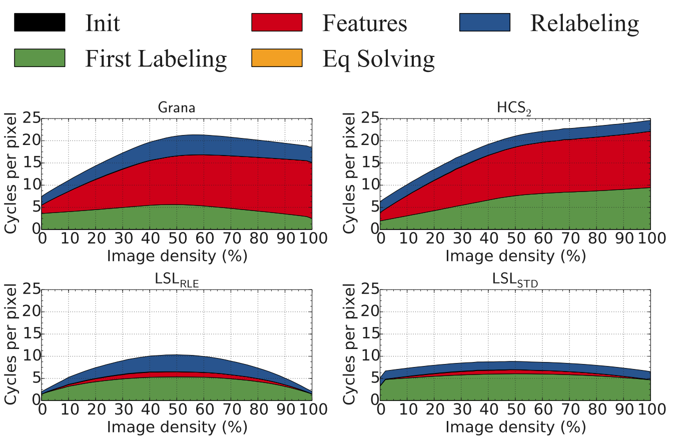

Fig. 10. Algorithms time-slicing for $g = 4$

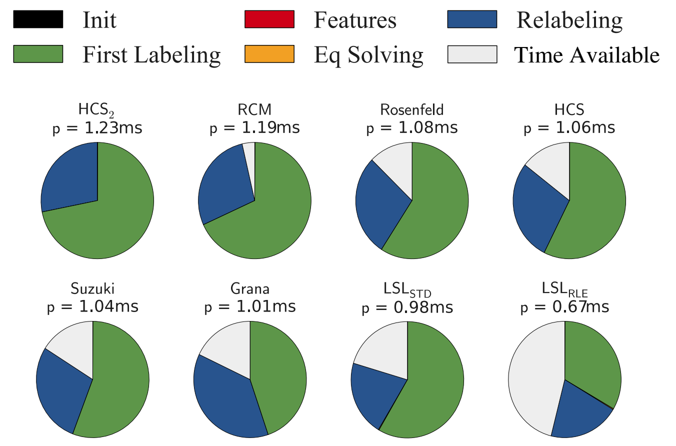

Fig. 11. Average processing time p on SIDBA images without FC, slower is the reference

b) Real case images: Pie-graphs show the time repartition for SIDBA images (full pie for the slowest algorithm). The average SIDBA *cpp* compared to random images of granularity is in the interval $[4 \rightarrow 8]$ for pixel algorithm and $[8 \rightarrow 16]$ for LSL versions. So random images with a granularity of 1 are not representative of real use cases, they are just useful to find the synthetic/theoretical worst case. But for a practical case, random images with $g \geq 4$ are more suitable. As for random images with high granularity, FC is invisible for LSL algorithms, whereas it can be half of the whole processing time for pixel algorithms (RCM, HCS_2).

V. Conclusion and Future Works

In this paper, we proposed a new detailed benchmark procedure for evaluating connected component labeling algorithms, with granularity steps that are complemented with the use of a standard database. This benchmark procedure, applied to a selection of State-of-the-Art algorithms, confirms that LSL algorithms are still the world's fastest CCL algorithms.

As we focus on real utilization of CCL – that implies to extract some features – we performed an in-depth analysis of the labeling parts duration to explain why run-length based algorithms outperform pixel-based ones. The experimentation shows that all pixel-based algorithms have a very similar behavior (labeling, transitive closure and relabeling parts) both on random images and images from data-bases. Moreover, the features computation is a significant time-consuming part for pixel-based algorithms whereas for LSL, features computation is very fast and efficient (for image granularity ≥ 4) thanks to run-length encoding. Future work will consider the parallelization of these algorithms as modern processors are multicore.

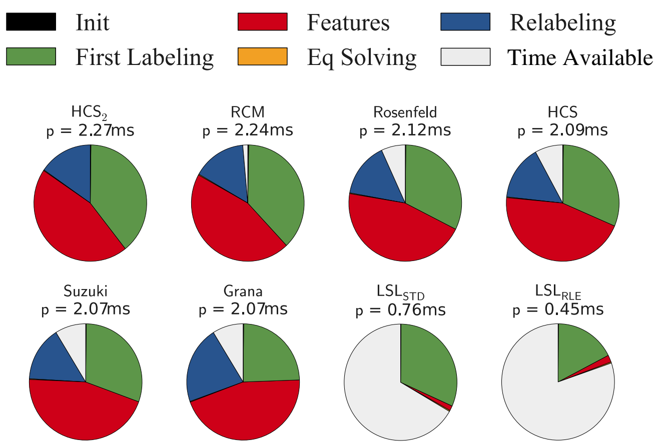

Fig. 12. Average processing time p on SIDBA images with FC, slower is the reference

References

[1] F. Chang and C. Chen. A linear-time component-labeling algorithm using contour tracing technique. *Computer Vision and Image Understanding*, 93:206–220, 2004.

[2] T. Cormen, C. Leiseirson, R. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2001.

[3] C. Grana, D.Borghesani, and R. Cucchiara. Fast block based connected components labeling. In *ICIP*, pages 4061–4064. IEEE, 2009.

[4] R. Haralick and L. Shapiro. *Computer and Robot Vision*. Addison-Wesley ISBN 0-201-56943-4, 1981.

[5] L. He, Y. Chao, and K. Suzuki. A run-based two-scan labeling algorithm. In *ICIAR*, pages 131–142. LNCS, 2007.

[6] L. He, Y. Chao, and K. Suzuki. An efficient first-scan method for label-equivalence-based labeling algorithms. *Pattern Recognition Letters*, 31(1):28–35, 2010.

[7] L. He, Y. Chao, and K. Suzuki. A new two-scan algorithm for labeling connected components in binary images. In W. Congress, editor, *Proceedings of the World Congress on Engineering*, volume 2, pages p1141–1146, 2012.

[8] U. Hernandez-Belmonte, V. Ayala-Ramirez, and R. Sanchez-Yanez. Enhancing ccl algorithms by using a reduced connectivity mask. In Springer, editor, *Mexican Conference on Pattern Recognition*, pages 195–203, 2013.

[9] L. Lacassagne and B. Zavidovique. Light speed labeling: efficient connected component labeling on risc architectures. *Journal of Real-Time Image Processing*, 6(2):117–135, 2011.

[10] R. Lumia, L. Shapiro, and O. Zungia. A new connected components algorithms for virtual memory computers. *Computer Vision, Graphics and Image Processing*, 22-2:287–300, 1983.

[11] M. Matsumoto and T. Nishimura. Mersenne twister: A 623-dimensionally equidistributed uniform pseudorandom number generator. *Transactions on Modeling and Computer simulation*, 8(1):3–30, 1998.

[12] N. Otsu. A threshold selection method from gray-level histograms. *Transactions on System, Man and Cybernetics*, 9:62–66, 1979.

[13] C. Ronse and P. Dejvijver. Connected components in binary images: the detection problems. In *Research Studies Press*, 1984.

[14] A. Rosenfeld and J. Platz. Sequential operator in digital pictures processing. *Journal of ACM*, 13,4:471–494, 1966.

[15] R. E. Tarjan. Efficiency of a good but not linear set union algorithm. *J. ACM*, pages 215–225, 1975.

[16] K. Wu, E. Otoo, and A. Shoshani. Optimizing connected component labeling algorithms. *Pattern Analysis and Applications*, 2008.