YACCLAB - Yet Another Connected Components Labeling Benchmark

Costantino Grana, Federico Bolelli, Lorenzo Baraldi, Roberto Vezzani
Dipartimento di Ingegneria “Enzo Ferrari”
Università degli Studi di Modena e Reggio Emilia
41125 Modena, Italy
Email: {name.surname}@unimore.it

***Abstract*—The problem of labeling the connected components (CCL) of a binary image is well-defined and several proposals have been presented in the past. Since an exact solution to the problem exists and should be mandatory provided as output, algorithms mainly differ on their execution speed. In this paper, we propose and describe YACCLAB, Yet Another Connected Components Labeling Benchmark. Together with a rich and varied dataset, YACCLAB contains an open source platform to test new proposals and to compare them with publicly available competitors. Textual and graphical outputs are automatically generated for three kinds of test, which analyze the methods from different perspectives. The fairness of the comparisons is guaranteed by running on the same system and over the same datasets. Examples of usage and the corresponding comparisons among state-of-the-art techniques are reported to confirm the potentiality of the benchmark.**

I. Introduction

The last 20 years have been really significant for both computer vision and image processing, and huge advancements have been made. Part of the responsibility for this huge progress may be credited to the broad access to public image and video datasets. Even if datasets have been blamed for narrowing the focus of research on object recognition, reducing it to a single benchmark performance number, it is now clear that the ability to compare different techniques on the same data allows the reader to chose which algorithm suits his needs best [1].

Performance must be quantified in order for objective comparisons to be made [2]. The question is: “Is there any reason not to use a common benchmark?” We are not alone in believing that the answer is a sound: “No.” One point which may remain open is how to perform a fair evaluation, and many algorithms seem to have been designed without explicit consideration of the statistical character of the input data at all: this makes it difficult to produce any quantitative prediction of performance [3].

But there are some specific aspects of image processing in which the expected result is known, and this lessens the burden of the evaluator, since, after checking that the result is correct, the main question left is to measure how fast an algorithm is. The problem of labeling the connected components of a binary image is such a problem, so one would expect every paper on the subject to focus on the same evaluation method and data. This is not the case. In recent years, many novel proposals have been published (and one of the authors of this paper is responsible for one of those) and almost none of them compared on the same data.

This paper tackles the problem of evaluating the speed of execution of different strategies to solve the Connected Components Labeling (CCL) problem on binary images.

When talking of “speed of execution”, one should answer to three questions:

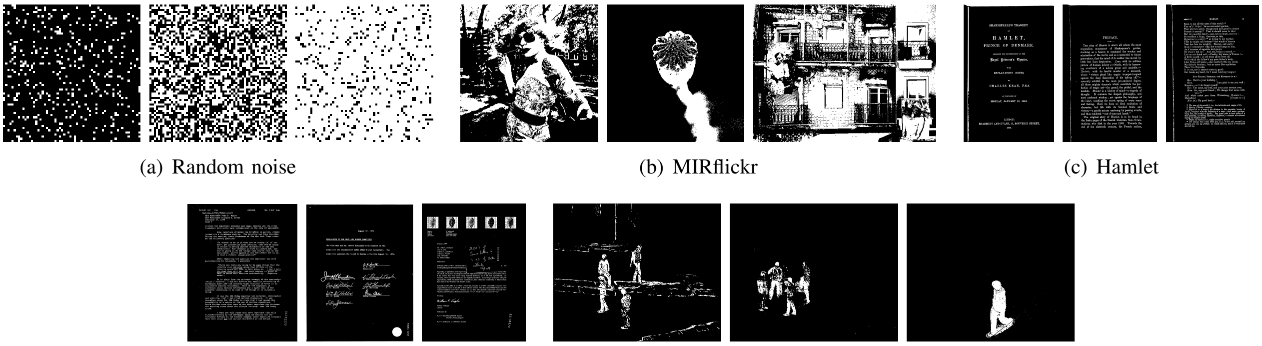
1) on what data?
2) on which machine?
3) with which implementation?

We answer the first question by providing a public dataset of binary images without any license limitations, or synthetically generated ones. We tried to cover different application scenarios for CCL algorithms such as motion analysis, document processing, and OCR.

We then provide an open-source C++ project with a very permissive license, so the answer to the second question is: “Yours.” Anyone will be able to take the provided algorithms and test them on his own setting, verifying any claim found in the literature (ours included). Another element of variability, which may not be so significant but still has a some impact, is the compiler used. So also that is taken care by the availability of the source code.

The final question is probably the hardest one, because providing source code is, unfortunately, not a common requirement for papers to be published. So our weakest answer is the third one: “with our implementation, if the authors did not provide source code.” A positive note is that being our project open source, any author believing we did him wrong is welcome to provide a better implementation than ours.

The evaluation framework is unimaginatively called Yet Another Connected Components Labeling Benchmark (YACCLAB in short), and the accompanying dataset is the YACCLAB Dataset. If this already convinced you, the project can be found at `https://github.com/prittt/YACCLAB`. Otherwise, Section II describes the dataset, Section III provides some details on how the framework works and how to extend it, and Section IV summarizes the algorithms we already provide in YACCLAB. Section V shows how the currently implemented algorithms perform on our machines and serves as a showcase for the automatically generated outputs. Finally, in Section VI, we draw the conclusions.



(a) Random noise (b) MIRflickr (c) Hamlet

(d) Tobacco800 (e) 3DPeS

Fig. 1. Sample images from the YACCLAB dataset.

II. The YACCLAB Dataset

Following a common practice in the literature, the YACCLAB dataset includes both synthetic and real images. All images are provided in 1 bit per pixel PNG format, with 0 (black) being background and 1 (white) being foreground. Images are organized by folders as follows.

A set of synthetic random noise images has been used in many papers [4], [5], [6]. We have selected and included in YACCLAB the publicly available set of [6], because it is the only one already published and already used in other works [7], [8], [9]. This allows the researchers to easily compare their results with the literature. The images contain black and white random noise with 9 different foreground densities (10% up to 90%), from a low resolution of 32×32 pixels to a maximum resolution of 4096×4096 pixels, allowing to test the scalability and the effectiveness of different approaches when the number of labels gets high. For every combination of size and density, 10 images are provided for a total of 720 images. The resulting subset allows to evaluate performance both in terms of scalability on the number of pixels and on the number of labels (density).

The second dataset included is composed by the Otsu-binarized version of the MIRflickr dataset [10], publicly available under a Creative Commons License. It contains 25,000 standard resolution images taken from Flickr. These images have an average resolution of 0.17 megapixels, there are few connected components (495 on average) and are generally composed of not too complex patterns, so the labeling is quite easy and fast. This subset serves again as a comparison with already published results.

Two other parts are included in the YACCLAB Dataset, in order to cover for Document Analysis applications. The first dataset is a set of 104 images scanned from a version of the Hamlet found on Project Gutenberg (`http://www.gutenberg.org`). The second one is the Tobacco800 Document Image Database. It is composed of 1290 document images and is a realistic database for document image analysis research as these documents were collected and scanned using a wide variety of equipment over time. Resolutions of documents in Tobacco800 vary significantly from 150 to 300 DPI and the dimensions of images range from 1200 by 1600 to 2500 by 3200 pixels [11], [12], [13]. Since CCL is one of the initial preprocessing steps in most layout analysis or OCR algorithms, these two subsets allow to test the algorithm performance in such scenarios.

The final set of images included in YACCLAB comes from 3DPeS (3D People Surveillance Dataset [14]), a surveillance dataset designed mainly for people re-identification in multi camera systems with non-overlapped fields of view. 3DPeS can be also exploited to test many other tasks, such as people detection, tracking, action analysis and trajectory analysis. The background models for all cameras are provided, so a very basic technique of motion segmentation has been applied to generate the foreground binary masks, i.e., background subtraction and fixed thresholding. The analysis of the foreground masks to remove small connected components and for nearest neighbor matching is a common application for CCL.

III. The YACCLAB Project

YACCLAB contains an open source C++ project which runs and tests CCL algorithms on the collection of datasets described in the previous section. Beside running a CCL algorithm and testing its correctness, YACCLAB performs three more kinds of test: average run-time test, density test and size test, in which the performance of the algorithms are evaluated with images of increasing density and size.

To check the correctness of an implementation, the output of an algorithm is compared with that of the OpenCV CCL function, which is assumed to be a correct reference point. Notice that 8-connectivity is always used. A colorized version of the input images can also be produced, to visually check the output and investigate possible labeling errors.

Average run-time tests, instead, execute an algorithm on every image of a dataset. The process can be repeated more times in a single test, to get the minimum execution time for each image: this allows to get more reproducible results and overlook delays produced by other running processes. It is also possible to compare the execution speed of different algorithms on the same dataset: in this case, selected algorithms are executed sequentially on every image of the dataset.

TABLE I
YACCLAB CONFIGURATION PARAMETERS.

Parameter name	Description
input_path	Folder on which datasets are placed
output_path	Folder on which result are stored
write_n_labels	Whether to report the number of Connected Components in output files
Correctness tests	
check_8connectivity	Whether to perform correctness tests
check_list	List of datasets on which CCL algorithms should be checked
Density and size tests	
ds_perform	Whether to perform density and size tests
ds_colorLabels	Whether to output a colorized version of input images
ds_testsNumber	Number of runs
ds_saveMiddleTests	Whether to save the output of single runs, or only a summary of the whole test
Average execution time tests	
at_perform	Whether to perform average execution time tests
at_colorLabels	Whether to output a colorized version of input images
at_testsNumber	Number of runs
at_saveMiddleTests	Whether to save the output of single runs, or only a summary of the whole test
averages_tests	List of algorithms on which average execution time tests should be run
Algorithm configuration	
CCLAlgorithmsFunc	List of available algorithms (function names)
CCLAlgorithmsName	List of available algorithms (display names for charts)

Results are presented in three different formats: a plain text file, histogram charts, either in color or in gray-scale, and a LaTeX table, which can be directly included in research papers.

Finally, density and size tests check the performance of different CCL algorithms when they are executed on images with varying foreground density and size. To this aim, a list of algorithms selected by the user is run sequentially on every image of the test_random dataset. As for run-time tests, it is possible to repeat this test for more than one run. The output is presented as both plain text and charts. For a density test, the mean execution time of each algorithm is reported for densities ranging from 10% up to 90%, while for a size test the same is reported for resolutions ranging from 32×32 up to 4096×4096. A showcase will be presented in Section V.

A configuration file placed in the installation folder lets the user specify which kind of test should be performed, on which datasets and on which algorithms. A complete description of all configuration parameters is reported in Table I.

YACCLAB has been designed with extensibility in mind, so that new resources can be easily integrated into the project. A CCL algorithm is coded with a `.h` header file, which declares a function implementing the algorithm, and a `.cpp` source file which defines the function itself. The function must follow a standard signature: its first parameter should be a const reference to an OpenCV `Mat1b` matrix, containing the input image, and its second parameter should be a reference to a `Mat1i` matrix, which shall be populated with computed labels. The function must also return an integer specifying the number of labels found in the image, included background's one. For example:

```
int MyLabelingAlgorithm(const cv::Mat1b& img,
        cv::Mat1i &imgLabels);
```

Once an algorithm has been added to YACCLAB, it is ready to be tested and compared to the others. To include the newly added algorithm in a test, it is sufficient to include its function name in the `CCLAlgorithmsFunc` parameter (see Table I) and a display name in the `CCLAlgorithmsName` parameter. We look at YACCLAB as a growing effort towards better reproducibility of CCL algorithms, so implementations of new and existing labeling methods are welcome.

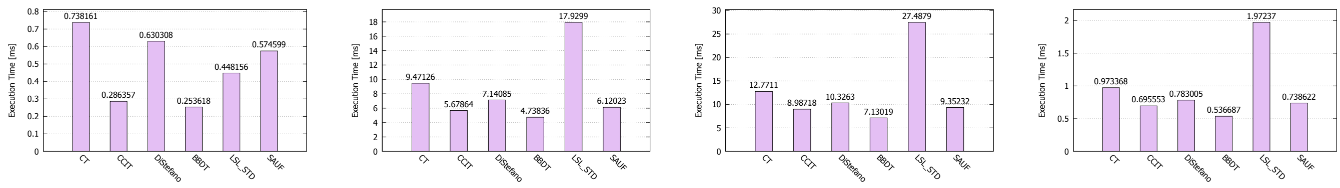
IV. Available algorithms

Since version 3.0, OpenCV included CCL features. The algorithm implemented is the one described in [15], [16], which is basically equivalent to the one in [4]. It uses a pixel based scanning with online equivalence resolution by means of a union find technique with path compression, plus a decision tree for accessing only the minimum number of already scanned labeled pixels. This is the reference algorithm implemented in YACCLAB and the source code comes directly from the OpenCV repository. It has been simplified just by removing the 4-connection case and any reference to blob features computations. Also the use of `InputArray` and `OutputArray` was removed in order to get a simpler interface. All of these modifications do not impact on the algorithm execution times, neither negatively nor positively.

In [6] we proved that different versions of the decision tree are equivalent to the previous one and extended it to Block Based scanning, that is scanning the image in 2×2 blocks. Building the decision tree for that case is much harder, because of the large number of possible combinations. In [17] we proposed a proved optimal strategy to build the decision tree by means of a dynamic programming approach. In YACCLAB we provide an implementation of the algorithm. The only difference with the originally available algorithm is the compliance with the new OpenCV funcion interface and the use of the same strategy found in most OpenCV modules to get the line pointer and then use it instead of the templated `operator()`, which is really handy in designing algorithms, but also known to slow down things considerably. The final decision tree is generated by another program, so a caveat to the reader is mandatory: the code is definitely ugly!

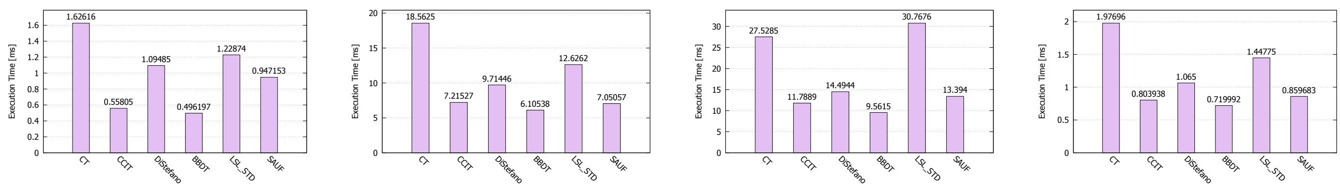
Another variation of Block Based analysis was proposed in [9], which is reported to be faster than the previous one. We also include this algorithm thanks to the availability of the source code on the authors web pages, making the signature compliant with our standards.

In [18] a different take on labeling was proposed, called Light Speed Labeling. The paper has a well described pseudocode for the algorithm, and in the journal version [19], further analysis has been performed, proposing also some variations and stating that it is the fastest algorithm available. One important note is that *Light Speed Labeling is not always correct*: failures in exactly solving all equivalences is reported



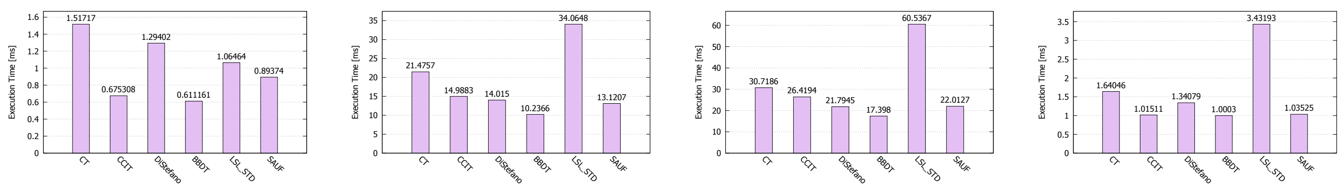
(a) MIRflickr (b) Hamlet (c) Tobacco800 (d) 3DPeS

Fig. 2. Average run-time tests on a i7-4790 CPU @ 3.60 GHz with Windows.



(a) MIRflickr (b) Hamlet (c) Tobacco800 (d) 3DPeS

Fig. 3. Average run-time tests on a Xeon CPU E5-2609 v2 @ 2.50GHz with Linux.



(a) MIRflickr (b) Hamlet (c) Tobacco800 (d) 3DPeS

Fig. 4. Average run-time tests on a Intel Core Duo @ 2.8GHz with OSX.

by the authors in about 2% of the cases. To our knowledge, no public implementation exists, so we are the first ones to really make it available in YACCLAB. This is probably due to two small mistakes in the pseudo code of [19]: a w was called n because of a change in notation between the two papers and a "step 2" was missing in a "for" loop. We implemented the standard version –this is the name used in the paper– of LSL, limiting the optimization again to just using raw row-pointers.

In order to demonstrate the importance of the algorithm used to solve the label's equivalences, we include an implementation of [20] which uses a two scan procedure with an online label resolution algorithm, using an array-based structure to store the label equivalences. This technique requires multiple searches over the array at every Union operation, leading to a clear non-linear behavior with respect to the number of labels.

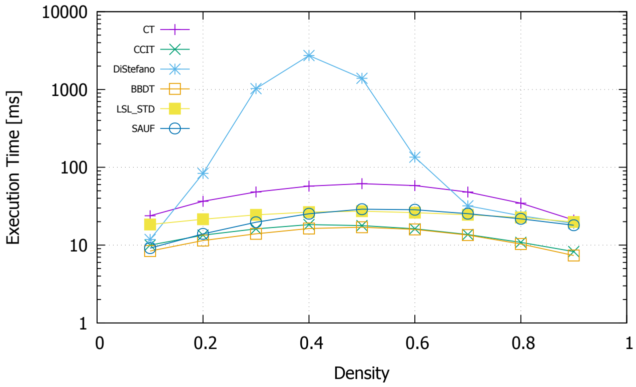
As a representative of the contour tracing type of algorithms we include [21] proposed in 2003. This approach clockwise tags all pixels in both the contour and the immediately external background area in a single operation. Then, during the raster scan, an untagged boundary is found, a counter clockwise contour tracing is performed for internal contours. This technique proved to have a linear complexity with respect to the number of labels and run quite fast, also because the filling of the connected components (label propagation after contour following) is cache-friendly for images stored in a raster scan order.

V. Current results

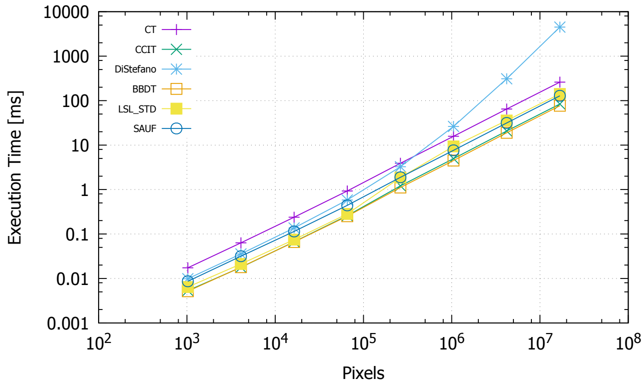
To make a first performance comparison and to showcase automatically generated charts and tables, we have run each algorithm in YACCLAB on all datasets and in three different environments: a Windows PC with a i7-4790 CPU @ 3.60 GHz and Microsoft Visual Studio 2013, a Linux workstation with a Xeon CPU E5-2609 v2 @ 2.50GHz and GCC 5.2, and a Intel Core Duo @ 2.8 GHz running OS X with X Code 7.2.1. Average run-time tests, as well as density and size tests, were repeated 10 times, and for each image the minimum execution time was considered.

In the following, we use acronyms to refer to the available algorithms: CT is the Contour Tracing approach by Fu Chang *et al.* [21], CCIT is the algorithm by Wan-Yu Chang *et al.* [9], DiStefano is the algorithm in [20], BBDT is the Block Based with Decision Trees algorithm by Grana *et al.* [6], LSL_STD is the Light Speed Labeling algorithm by Lacassagne *et al.* [19], SAUF is the Scan Array Union Find algorithm by Wu *et al.* [16], which is the algorithm currently included in OpenCV.

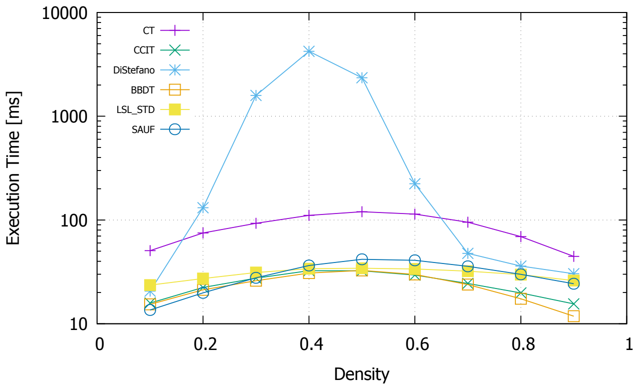
Figures 2, 3, and 4 report mean run-times for each platform and dataset. As it can be noticed, YACCLAB provides an



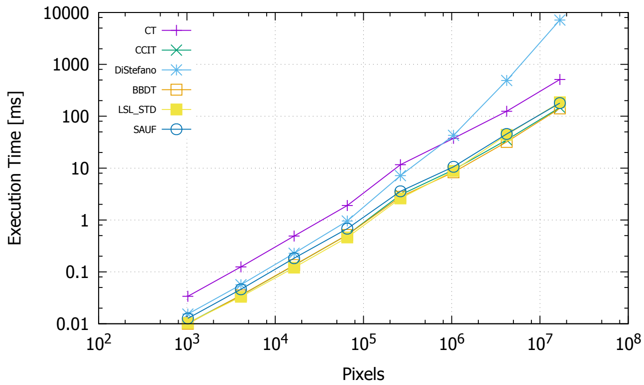
(a) Density test on a i7-4790 CPU @ 3.60 GHz with Windows.



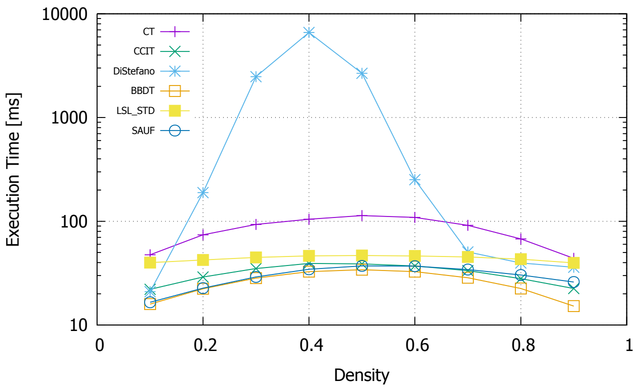
(b) Size test on a i7-4790 CPU @ 3.60 GHz with Windows.



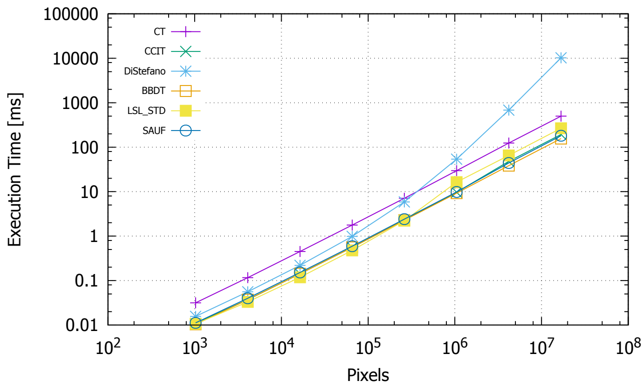
(c) Density test on a Xeon CPU E5-2609 v2 @ 2.50GHz with Linux.



(d) Size test on a Xeon CPU E5-2609 v2 @ 2.50GHz with Linux.



(e) Density test on a Intel Core Duo @ 2.8GHz with OS X.



(f) Size test on a Intel Core Duo @ 2.8GHz with OS X.

Fig. 5. Size and Density tests.

effective way to compare CCL algorithms on heterogeneous datasets and environments. Same results can be reported in tabular form, as in Tables II, III, and IV. Execution times vary from platform to platform, depending on the processor performance as well as on the optimization carried out by the compiler. However, with the implementations currently included in YACCLAB, the method proposed in [6] always shows the best performance.

Beside run-time tests, size and density test can be automatically executed on the synthetic dataset. Figures 5(b), 5(d), and 5(f) report execution times on all synthetic images and platforms with respect to different image sizes. A linear dependency of execution time with respect to the number of pixels is highlighted for all algorithms, except from Di Stefano's, which shows, as expected, a worse performance when the number of pixels is high.

Figures 5(a), 5(c), and 5(e), finally, report execution times on the syntethic dataset, with respect to different levels of foreground density. All algorithms show an increased execution time on middle densities, because the number of

TABLE II
AVERAGE RESULTS IN MS ON A I7-4790 CPU @ 3.60 GHZ WITH WINDOWS.

	CT	CCIT	DiStefano	BBDT	LSL	SAUF
MIRflickr	0.73	0.28	0.63	0.25	0.44	0.57
Tob800	12.77	8.98	10.32	7.13	27.48	9.35
3DPeS	0.97	0.69	0.78	0.53	1.97	0.73
Hamlet	9.47	5.67	7.14	4.73	17.93	6.12

TABLE III
AVERAGE RESULTS IN MS ON A XEON CPU E5-2609 V2 @ 2.50GHZ WITH LINUX.

	CT	CCIT	DiStefano	BBDT	LSL	SAUF
MIRflickr	1.62	0.55	1.09	0.49	1.22	0.94
Tob800	27.52	11.78	14.49	9.56	30.76	13.39
3DPeS	1.97	0.80	1.06	0.72	1.44	0.86
Hamlet	18.56	7.21	9.71	6.10	12.62	7.05

TABLE IV
AVERAGE RESULTS IN MS ON A INTEL CORE DUO @ 2.8GHZ WITH OS X.

	CT	CCIT	DiStefano	BBDT	LSL	SAUF
MIRflickr	1.51	0.67	1.29	0.61	1.06	0.89
Tob800	30.71	26.41	21.79	17.39	60.53	22.01
3DPeS	1.64	1.01	1.34	1.00	3.43	1.03
Hamlet	21.47	14.98	14.01	10.23	34.06	13.12

labels and merges between equivalence classes is higher. Di Stefano's algorithm produces the worst performance in the middle densities.

VI. CONCLUSION

In this paper we described two contributions to the image processing community: a comprehensive dataset for comparing Connected Components Labeling Algorithms and a portable open source C++ project to test different algorithms on top of it. No new algorithms were proposed, but this tool allows any new improvement to be evaluated uniformly with respect to existing proposals.

We are strongly in favor of testing paper results with source code. It is not a lack of trust in the other researchers, but a realistic need to compare. Instead of forcing everybody to reimplement other algorithms' code from scratch, and then draw wrong conclusions because they overlooked a peculiarity (probably insufficiently stressed in the original paper), we support the fact that the inventor knows its creature best. So, we welcome all contributions to YACCLAB in terms of new algorithms, novel datasets for applications we did not consider, or better implementations of what we already included. It is not likely that the code we are providing is the best you can write, so review and improvement is mandatory.

Further work is needed to systematically test these algorithms on different machines to point out weaknesses and strengths of the various proposals. And if you are going to sport that your algorithm is the fastest in the world, please make the source code available in YACCLAB.

REFERENCES

[1] A. Torralba and A. A. Efros, "Unbiased Look at Dataset Bias," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2011, pp. 1521–1528.

[2] A. F. Clark and P. Courtney, "Databases for Performance Characterization," in *Performance Characterization in Computer Vision*. Springer, 2000, pp. 29–40.

[3] N. A. Thacker, A. F. Clark, J. L. Barron, J. R. Beveridge, P. Courtney, W. R. Crum, V. Ramesh, and C. Clark, "Performance characterization in computer vision: A guide to best practices," *Computer vision and image understanding*, vol. 109, no. 3, pp. 305–334, 2008.

[4] L. He, Y. Chao, and K. Suzuki, "A Run-Based Two-Scan Labeling Algorithm," *IEEE Transactions on Image Processing*, vol. 17, no. 5, pp. 749–756, 2008.

[5] L. He, Y. Chao, K. Suzuki, and K. Wu, "Fast connected-component labeling," *Pattern Recognition*, vol. 42, no. 9, pp. 1977–1987, 2009.

[6] C. Grana, D. Borghesani, and R. Cucchiara, "Optimized Block-based Connected Components Labeling with Decision Trees," *IEEE Transactions on Image Processing*, vol. 19, no. 6, pp. 1596–1609, 2010.

[7] P. Sutheebanjard and W. Premchaiswadi, "Efficient scan mask techniques for connected components labeling algorithm," *EURASIP Journal on image and Video Processing*, vol. 2011, no. 1, pp. 1–20, 2011.

[8] L. He, X. Zhao, Y. Chao, and K. Suzuki, "Configuration-Transition-Based Connected-Component Labeling," *IEEE Transactions on Image Processing*, vol. 23, no. 2, pp. 943–951, 2014.

[9] W.-Y. Chang, C.-C. Chiu, and J.-H. Yang, "Block-based connected-component labeling algorithm using binary decision trees," *Sensors*, vol. 15, no. 9, pp. 23 763–23 787, 2015.

[10] M. J. Huiskes and M. S. Lew, "The MIR Flickr Retrieval Evaluation," in *MIR '08: Proceedings of the 2008 ACM International Conference on Multimedia Information Retrieval*. New York, NY, USA: ACM, 2008. [Online]. Available: http://press.liacs.nl/mirflickr/

[11] G. Agam, S. Argamon, O. Frieder, D. Grossman, and D. Lewis, "The Complex Document Image Processing (CDIP) Test Collection Project," Illinois Institute of Technology, 2006. [Online]. Available: http://ir.iit.edu/projects/CDIP.html

[12] D. Lewis, G. Agam, S. Argamon, O. Frieder, D. Grossman, and J. Heard, "Building a test collection for complex document information processing," in *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, 2006, pp. 665–666.

[13] "The Legacy Tobacco Document Library (LTDL)," University of California, San Francisco, 2007. [Online]. Available: http://legacy.library.ucsf.edu/

[14] D. Baltieri, R. Vezzani, and R. Cucchiara, "3DPeS: 3D People Dataset for Surveillance and Forensics," in *Proceedings of the 2011 joint ACM workshop on Human gesture and behavior understanding*. ACM, 2011, pp. 59–64.

[15] K. Wu, E. Otoo, and K. Suzuki, "Two Strategies to Speed up Connected Component Labeling Algorithms," Lawrence Berkeley National Laboratory, Tech. Rep. LBNL-59102, 2005.

[16] ——, "Optimizing two-pass connected-component labeling algorithms," *Pattern Analysis and Applications*, vol. 12, no. 2, pp. 117–135, 2009.

[17] C. Grana, M. Montangero, and D. Borghesani, "Optimal decision trees for local image processing algorithms," *Pattern Recognition Letters*, vol. 33, no. 16, pp. 2302–2310, 2012.

[18] L. Lacassagne and B. Zavidovique, "Light Speed Labeling for RISC architectures." in *ICIP*, 2009, pp. 3245–3248.

[19] ——, "Light speed labeling: efficient connected component labeling on risc architectures," *Journal of Real-Time Image Processing*, vol. 6, no. 2, pp. 117–135, 2011.

[20] L. Di Stefano and A. Bulgarelli, "A Simple and Efficient Connected Components Labeling Algorithm," in *International Conference on Image Analysis and Processing*. IEEE, 1999, pp. 322–327.

[21] F. Chang, C.-J. Chen, and C.-J. Lu, "A linear-time component-labeling algorithm using contour tracing technique," *Computer Vision and Image Understanding*, vol. 93, no. 2, pp. 206–220, 2004.