Optimized Connected Components Labeling with Pixel Prediction

Costantino Grana, Lorenzo Baraldi, Federico Bolelli

Dipartimento di Ingegneria "Enzo Ferrari"
Università degli Studi di Modena e Reggio Emilia
Via Vivarelli 10, Modena MO 41125, Italy
`name.surname@unimore.it`

Abstract. In this paper we propose a new paradigm for connected components labeling, which employs a general approach to minimize the number of memory accesses, by exploiting the information provided by already seen pixels, removing the need to check them again. The scan phase of our proposed algorithm is ruled by a forest of decision trees connected into a single graph. Every tree derives from a reduction of the complete optimal decision tree. Experimental results demonstrated that on low density images our method is slightly faster than the fastest conventional labeling algorithms.

Keywords: Connected Components Labeling; Binary Decision Trees.

1 Introduction

Connected Components Labeling (CCL) of binary images is an important and well-defined problem in image processing. With the labeling operation, a binary image is transformed into a symbolic image in which all pixels belonging to the same connected component are given the same label: this transformation is required whenever a computer program needs to identify independent components in an image, and is therefore a fundamental pre-processing task of many pattern analysis, computer vision and robot vision algorithms.

Given that an exact solution to the CCL problem exists, and should be provided as output, researchers have focused in the last years on optimizing existing CCL algorithms and on developing faster ones. The majority of existing algorithms scan the image and look at the neighborhood of a pixel through a mask (see Fig. 1 for an example), to assign provisional labels and collect label equivalences. This step, which can be expressed as a decision table [7], is done independently for each pixel, so that the pixels in the neighborhood are accessed multiple times during the scan phase.

In this work we propose a general paradigm to exploit already seen pixels during the scan phase, so to minimize the number of times a pixel is accessed. As shown in literature, the decision table which rules the scan step can be conveniently converted to an optimal binary decision tree [7], in which internal nodes represent conditions on mask's pixels, and leaves represents actions to be

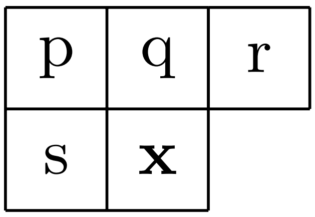


Fig. 1. The pixel mask used to compute the label of pixel x.

performed on the current pixel of the provisional image (x in Fig. 1), such as the creation of a new label, the assignment of an existing label to the pixel, or the merge of two existing labels. Usually, the same decision tree is traversed for each pixel of the input image, without exploiting values seen in the previous iteration, which, if considered, would result in a simplification of the decision tree for the pixel.

To go beyond this limitation, we compute a reduced decision tree for each possible set of known pixels; these reduced decision trees are then connected into a single graph, which rules the execution of the CCL algorithm on the whole image. This graph contains a starting tree, which should be accessed to process the first pixel of every row, and other trees, which are accessed to process the following pixels. Each leaf of a tree, which represent the action to be performed on a pixel, is connected to the root of a second tree which should be executed for the next pixel. The obtained graph can then be directly converted into running code.

The rest of this paper is organized as follows: in Section 2 we give an overview of existing Connected Components Labeling algorithms; Section 3 contains the description of the proposed strategy, which is then evaluated in Section 4. Finally, we draw the conclusions in Section 5.

2 Previous Works

The research efforts in labeling techniques have a very long story, where different strategies, improvements and results have been presented.

The algorithm described in [15, 16] is basically equivalent to the one in [9]. 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.

In [7] it was proved that different versions of the decision tree are equivalent to the previous one and that, when performing 8-connected labeling, the scanning process can be extended to Block Based scanning, that is scanning the image in 2×2 blocks (we will refer to this algorithm as BBDT, for Block Based with Decision Tree). Building the decision tree for that case is much harder, because of the large number of possible combinations. In [8] a proved optimal strategy to build the decision tree has been proposed, by means of a dynamic programming approach. The final decision tree is generated automatically by another program.

Another variation of Block Based analysis was proposed in [4], 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 testing standard.

He *et al.* [10] recently observed that BBDT still checks many pixels repeatedly, because after labeling one pixel, the mask moves to the next one, but many pixels in the current mask are overlapped to the previous ones, which may have already been checked. They thus propose a Configuration-Transition-Based (CTB) algorithm which uses a set of different configurations to represent the current *state* of the algorithm and employ it to make further decisions. This allows to save a number of accesses to pixels and thus to speed up the labeling process. The algorithm is specifically designed for the task and no provision to a general methodology is foreseen in the paper.

Recently, an open-source benchmark for CCL has been proposed by the authors of this paper: YACCLAB [6]. This project is intended to be a growing effort towards better reproducibility of CCL algorithms, and allows researchers to test new proposals and available implementations on standard datasets. Given that the source code of many CCL algorithms has not been released, YACCLAB also relies on implementations provided by the authors of the benchmark. In the following sections we will show that our variation is often faster than the algorithm from He [10], and even faster than [7] when tested on very low density images.

3 Method

We focus our analysis on 8-connected CCL and start by observing the neighborhood mask of Fig. 1, choosing one of the possible optimal decision trees we can obtain from it. Fig. 2 provides a visual representation of such a tree: the first thing to do is to check whether current pixel x is background or foreground (0 or 1). If $x = 0$, than we don't need to do anything (action 1) and simply move to the next pixel, otherwise we start looking its neighborhood. We start from q because it is connected to all other pixels: if $q = 1$ its label will be equivalent to that of all others, so we simply *assign* its label to x (action 4). If instead $q = 0$ we are in the case of potential *merge* of different previously unconnected components through x. To this aim, we need to check pixels p, s and r basically in any order we like with the only saving of avoiding checking p if $s = 1$ or viceversa, since p is always connected to s. Finally, if no pixel is foreground, i.e. $p = r = s = 0$, we create a *new label* (action 2). In Fig. 2 actions 3, 5, and 6 are *assign* p, r, and s respectively, while actions 7 and 8 are *merge* $p + r$ and $s + r$ respectively.

Following [10], we observe that pixels pixels x will be the next s, q will be the next p, pixels r will be the next q. So if during the tree traversal we made a choice on x, q or r we know the values of s, p and q at the next step. This means

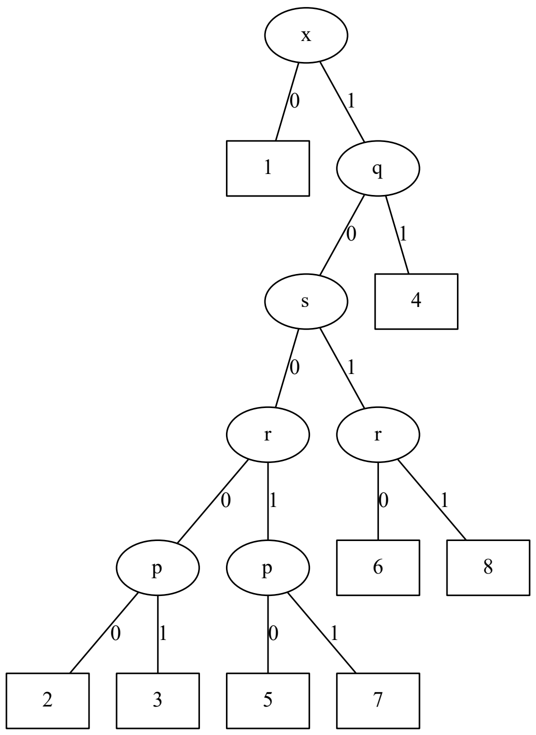


Fig. 2. One of the possible full decision trees obtainable from the mask of Fig. 1.

that the corresponding subtree may be substituted to the choice made on that pixel.

Let's start with the first thing we know for sure: at the beginning of a line pixels p and s are 0 (there is no foreground outside the image). So all choices do not need to check those, meaning that we can remove the right branch of s and use its left subtree instead. There $p = 0$, thus we remove that check and substitute it with actions 2 and 5. This gives reduced tree A of Fig. 3. Its meaning is very clear to understand: first check pixel x: if it is foreground check the previous line (q and r), otherwise move next.

If we had a background pixel x before, we know that $s = 0$ so, again, we can remove its right branch thus obtaining the reduced tree B. If instead x was foreground and q was too, both $p = s = 1$ and we can remove the left branch of s getting reduced tree C. We keep going in this way and in many cases we obtain the same information from every leaf. Overall just two more different reduced trees are obtained (D and E in Fig. 3). It is noteworthy that, if in any tree we

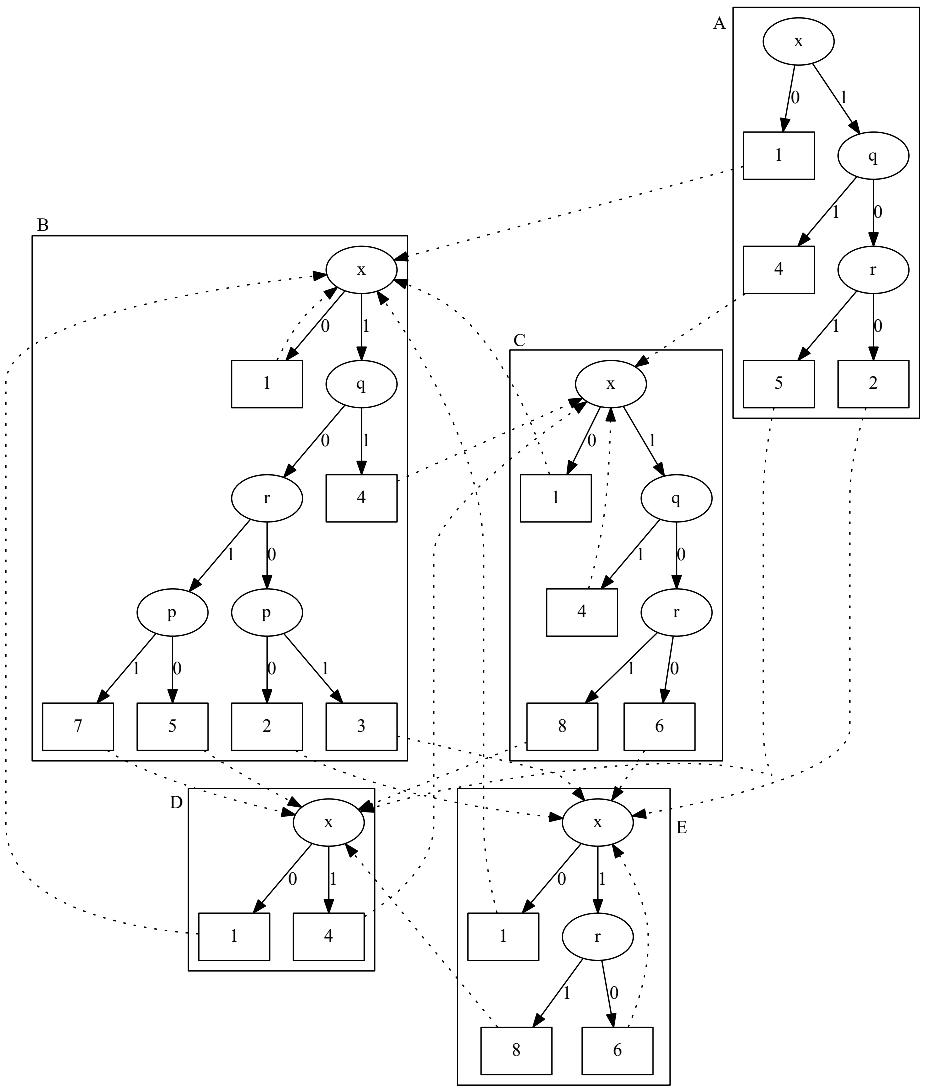


Fig. 3. The final graph of decision trees, obtained from the full decision tree of Fig. 2.

observed that $r = 1$, we move to tree D, which is simply a check on x: if it is foreground we assign the label of q without even the need of checking if it is foreground.

At every leaf of every tree, we know which tree shall be used next, so we mark that edge with a dotted line, to stress the difference with respect to the choices within the trees: inside every tree we just check the pixels, after performing the action in a leaf we need to advance the mask, check if the image row is finished and then proceed or break out of the current line.

```
...
tree_C:
        if (++c >= w - 1)
                goto break_C;
        if (condition_x) {
                if (condition_q) {
                        // action 4 - x <= q
                        goto tree_C;
                }
                else {
                        if (condition_r) {
                                // action 8 - x <= r + s
                                goto tree_D;
                        }
                        else {
                                // action 6 - x <= s
                                goto tree_E;
                        }
                }
        }
        else {
                // action 1 - do nothing
                goto tree_B;
        }
...
```

Fig. 4. Example code for reduced tree C. Here comments are used to indicate where the actions should be performed.

3.1 Implementation

Moving from theory to practice is just a matter of translating every branch with a conditional statement and every dotted link with an unconditional jump. At the beginning of every tree an increment of the current pixel position has to be made along with a check for the end of the row. A sample of the code in C language for tree C is provided in Fig. 4. The real implementation of the actions is omitted and substituted with comments. Conditions are indicated with a macro which shall be defined by the specific implementation.

One thing should be noted: the *end of row* check stops one pixel before the end of the row. In this way we can make specialized versions for all the trees for the last pixel case, where we know that $r = 0$. This simple trick allows us to save an end of line check at every use of pixel r, since we already know that we are at least one pixel inside the image.

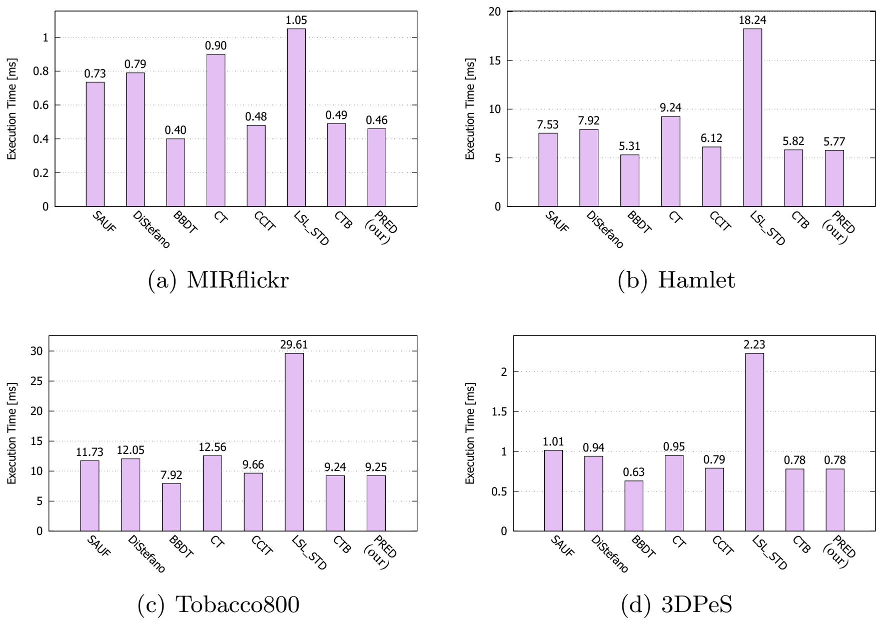


Fig. 5. Average results on a i7-4790K CPU @ 4.00 GHz with Windows and Microsoft Visual Studio 2013 (lower is better).

4 Experimental Evaluation

Tests were performed on a Windows PC with an Intel Core i7-4790K CPU @ 4.00 GHz and Microsoft Visual Studio 2013. All algorithms reported in this Section were included in YACCLAB. Tests were repeated ten times: for each image only the minimum execution time was considered (in order to reduce/avoid the influence of other tasks on the final results). Charts and tables report average times for every dataset or density/size considering only the minimum execution time on every image.

In the following, we use acronyms to refer to the available algorithms: CT is the Contour Tracing approach by Fu Chang *et al.* [3], CCIT is the algorithm by Wan-Yu Chang *et al.* [4], DiStefano is the algorithm in [5], BBDT is the Block Based with Decision Trees algorithm by Grana *et al.* [7], LSL_STD is the Light Speed Labeling algorithm by Lacassagne *et al.* [12], SAUF is the Scan Array Union Find algorithm by Wu *et al.* [16], which is the algorithm currently included in OpenCV, CTB is the Configuration-Transition-Based approach described in [10]. Our method is denoted as PRED.

Figure 5 and Table 1 report mean run-times for each dataset. As it can be noticed, YACCLAB provides an effective way to compare CCL algorithms on heterogeneous datasets:

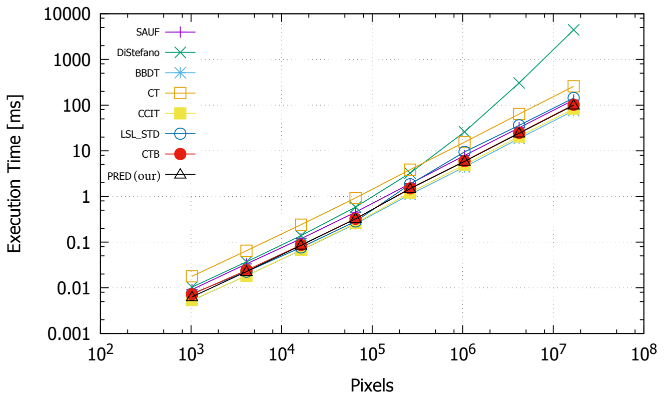
Table 1. Average results in ms on a i7-4790K CPU @ 4.00 GHz with Windows and Microsoft Visual Studio 2013 (lower is better).

	SAUF	DiStefano	BBDT	CT	CCIT	LSL_STD	CTB	PRED (our)
MIRflickr	0.735	0.795	**0.403**	0.902	0.481	1.052	0.491	0.458
Hamlet	7.531	7.921	**5.314**	9.245	6.118	18.242	5.819	5.769
Tobacco800	11.728	12.047	**7.924**	12.561	9.663	29.608	9.237	9.252
3DPeS	1.014	0.945	**0.625**	0.953	0.791	2.234	0.778	0.778

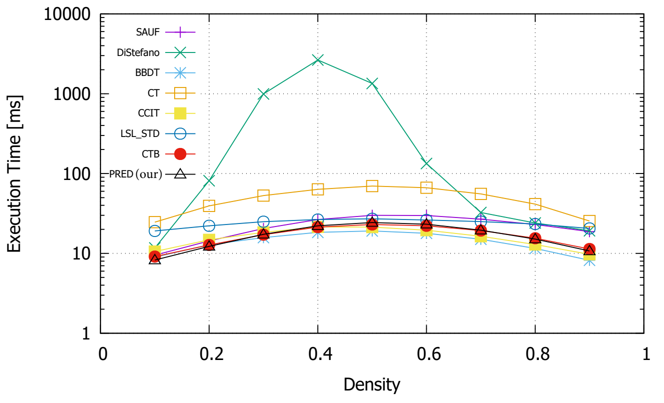
- *MIRflickr*, Figure 5(a): this dataset is composed by the Otsu-binarized version of the MIRflickr dataset [11]. It contains 25,000 standard resolution images taken from Flickr. These images have an average resolution of 0.18 megapixels, there are few connected components and are generally composed of not too complex patterns, so the labeling is quite easy and fast.
- *Hamlet*, Figure 5(b): a set of 104 images scanned from a version of the Hamlet found on Project Gutenberg (http://www.gutenberg.org). Images have an average amount of 2.71 million of pixels to analyze and 1,447 components to label.
- *Tobacco800*, Figure 5(c): it is composed of 1,290 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 1,200 by 1,600 to 2,500 by 3,200 pixels [1, 13, 14].
- *3DPeS*, Figure 5(d): it comes from 3DPeS (3D People Surveillance Dataset [2]), a surveillance dataset designed mainly for people re-identification in multi-camera systems with non-overlapped fields of view. The images of this dataset have an average resolution of 0.41 megapixels.

Beside run-time tests, size and density test were executed on a synthetic dataset: random noise images with 9 different foreground densities (10% up to 90%), from a low resolution of 32×32 pixels to a maximum resolution of $4,096 \times 4,096$ pixels. For every combination of size and density, 10 images were provided for a total of 720 images.

- *Size*, Figure 6(a): highlights a linear dependency of execution time with respect to the number of pixels. This is true for all algorithms except Di Stefano's one, which shows, as expected, a worse performance when the number of pixels is high.
- *Density*, Figure 6(b): reports density tests performed on the syntethic dataset. Like almost all others algorithms, PRED shows an increased execution time on middle densities, because the number of labels and merges between equivalence classes is higher and also branch prediction can affect negatively the execution times. Di Stefano's algorithm produces the worst performance in the middle densities instead LSL_STD is the only one that demonstrated a quasi-linear trend, probably due to the lower number of conditional statements.



(a) Size



(b) Density

Fig. 6. Density and size tests on a i7-4790K CPU @ 4.00 GHz with Windows and Microsoft Visual Studio 2013 (lower is better).

It is important to underline that all numerical values reported in the graphs and tables also consider times needed to define and initialize data structures, except for the binary input matrix. Indeed, if an algorithm needs to save more information to compute correct labeling, these must be considered in the total amount of execution time.

5 Conclusions

In this paper we presented a novel approach for performing Connected Components Labeling, which employs a reproducible strategy able to avoid repeatedly checking the same pixels multiple times. Experimental results are very promising and even if the current algorithm is not able to beat BBDT algorithm on the real datasets, it was faster on the synthetic one for low density cases. Moreover, it was able to surpass the performance of CTB which is currently the second best accordingly to the YACCLAB benchmark. We plan to apply the same optimization strategy also to the BBDT algorithm, but in this case the tree reduction cannot be performed *by hand*, given the enormous size of the decision tree.

The source code of the described method has been included in the YACCLAB benchmark, so that it will be possible to check the real performance on different machines and compare it with future proposals. We strongly believe that given the maturity of the problem and the subtlety involved in the implementation, it should be mandatory to allow the community to reproduce the results without forcing everyone to reimplement every proposal.

References

1. Agam, G., Argamon, S., Frieder, O., Grossman, D., Lewis, D.: The Complex Document Image Processing (CDIP) Test Collection Project. Illinois Institute of Technology (2006), http://ir.iit.edu/projects/CDIP.html
2. Baltieri, D., Vezzani, R., Cucchiara, R.: 3DPeS: 3D People Dataset for Surveillance and Forensics. In: Proceedings of the 2011 joint ACM workshop on Human gesture and behavior understanding. pp. 59–64. ACM (2011)
3. Chang, F., Chen, C.J., Lu, C.J.: A linear-time component-labeling algorithm using contour tracing technique. Computer Vision and Image Understanding 93(2), 206–220 (2004)
4. Chang, W.Y., Chiu, C.C., Yang, J.H.: Block-based connected-component labeling algorithm using binary decision trees. Sensors 15(9), 23763–23787 (2015)
5. Di Stefano, L., Bulgarelli, A.: A Simple and Efficient Connected Components Labeling Algorithm. In: International Conference on Image Analysis and Processing. pp. 322–327. IEEE (1999)
6. Grana, C., Bolelli, F., Baraldi, L., Vezzani, R.: YACCLAB - Yet Another Connected Components Labeling Benchmark. In: 23rd International Conference on Pattern Recognition. ICPR (2016)
7. Grana, C., Borghesani, D., Cucchiara, R.: Optimized Block-based Connected Components Labeling with Decision Trees. IEEE Transactions on Image Processing 19(6), 1596–1609 (2010)
8. Grana, C., Montangero, M., Borghesani, D.: Optimal decision trees for local image processing algorithms. Pattern Recognition Letters 33(16), 2302–2310 (2012)
9. He, L., Chao, Y., Suzuki, K.: A Run-Based Two-Scan Labeling Algorithm. IEEE Transactions on Image Processing 17(5), 749–756 (2008)
10. He, L., Zhao, X., Chao, Y., Suzuki, K.: Configuration-Transition-Based Connected-Component Labeling. IEEE Transactions on Image Processing 23(2), 943–951 (2014)
11. Huiskes, M.J., Lew, M.S.: The MIR Flickr Retrieval Evaluation. In: MIR '08: Proceedings of the 2008 ACM International Conference on Multimedia Information Retrieval. ACM, New York, NY, USA (2008), http://press.liacs.nl/mirflickr/
12. Lacassagne, L., Zavidovique, B.: Light speed labeling: efficient connected component labeling on risc architectures. Journal of Real-Time Image Processing 6(2), 117–135 (2011)

13. Lewis, D., Agam, G., Argamon, S., Frieder, O., Grossman, D., Heard, J.: 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. pp. 665–666. ACM (2006)
14. The Legacy Tobacco Document Library (LTDL). University of California, San Francisco (2007), http://legacy.library.ucsf.edu/
15. Wu, K., Otoo, E., Suzuki, K.: Two Strategies to Speed up Connected Component Labeling Algorithms. Tech. Rep. LBNL-59102, Lawrence Berkeley National Laboratory (2005)
16. Wu, K., Otoo, E., Suzuki, K.: Optimizing two-pass connected-component labeling algorithms. Pattern Analysis and Applications 12(2), 117–135 (2009)