Optimized Block-Based Connected Components Labeling With Decision Trees

Costantino Grana, *Member, IEEE*, Daniele Borghesani, and Rita Cucchiara, *Member, IEEE*

***Abstract*—In this paper, we define a new paradigm for eight-connection labeling, which employes a general approach to improve neighborhood exploration and minimizes the number of memory accesses. First, we exploit and extend the decision table formalism introducing OR-decision tables, in which multiple alternative actions are managed. An automatic procedure to synthesize the optimal decision tree from the decision table is used, providing the most effective conditions evaluation order. Second, we propose a new scanning technique that moves on a 2×2 pixel grid over the image, which is optimized by the automatically generated decision tree. An extensive comparison with the state of art approaches is proposed, both on synthetic and real datasets. The synthetic dataset is composed of different sizes and densities random images, while the real datasets are an artistic image analysis dataset, a document analysis dataset for text detection and recognition, and finally a standard resolution dataset for picture segmentation tasks. The algorithm provides an impressive speedup over the state of the art algorithms.**

***Index Terms*—Connected components labeling, decision tables, decision trees, optimization methods.**

I. Introduction

CONNECTED component labeling is a fundamental task in several image processing and computer vision applications, e.g., for identifying segmented visual objects or image regions. Thus a fast and efficient algorithm, able to minimize its impact on image analysis tasks, is undoubtedly very advantageous. Moreover, many applications where labeling is a necessary processing step often have to deal with high resolution images with thousands of labels: complex solutions for document analysis, multimedia retrieval, and biomedical image analysis would benefit the speedup of labeling considerably.

The research efforts in labeling techniques have a very long story, full of different strategies, improvements, and results. Some of these particular strategies were focused on taking advantage of the specific hardware architectures by that time, in terms of CPU and memory usage, trying to minimize the number of comparisons, the necessary sorts, the cost of the label management. Current computer architectures do not suffer anymore of many resource limitations and have new capabilities (in terms of memory capacity, CPU power, storage access speed): modern approaches can take advantage of the available resources and must only try to reduce memory access time, the main bottleneck of current computer systems.

Although the first algorithms for connected components labeling were proposed more than 50 years ago, only in the last years new strategies provided significant performance improvements, in particular with the introduction of the *Union-Find* approach for label equivalences resolution, array-based data structures and smarter neighborhood management.

This work aims at marking a new step forward. We propose a new methodology to consider a generic near-neighborhood task, as the connected components labeling, resuming the "old" condition-action paradigm which can be effectively described as a *single entry decision table*. Then we propose to further enhance this tool introducing the OR-*decision tables*, which enclose the possibility to represent more than one (*equivalent*) action for each set of conditions. An automatic procedure to select the most convenient alternative is proposed to get back a single-entry decision table, and finally a boolean optimization algorithm is adopted to automatically produce the optimal decision tree in terms of number of evaluations, thus access costs. Since this approach is fully automatic by design, it can be safely extended to each near-neighborhood task with a similar formalization. So, as matter of course, we approached raster-scan labeling in a block-wise (instead of pixel-wise) manner: our paradigm, applied as-it-is without any modifications, granted us significant performance improvements of the neighborhood exploration in terms of memory access times, compared to the state of the art. Moreover, we proved that our algorithm is able to outperform the state of the art both in high resolution images with thousands of labels and in standard resolution images with fewer labels.

Tests have been carried out on four different large datasets: a synthetic uniform noise dataset at different resolutions and densities, a digital library of high resolution replicas of an illuminated manuscript containing tenths of thousands of labels, a selection of digitized book pages publicly available on the Gutenberg Project website [1], and finally the Otsu binarized version of the MIRFlickr dataset [2]. Each dataset has a different amount of connected components, with peculiar patterns and at different resolutions, to test the algorithm in different situations. The OpenCV-compliant code and the random dataset are available online [3].

This paper is organized as follows. In Section II, we introduce the basic concepts and notation used throughout the paper. In Section III, we provide a historical overview of the different approaches to the problem of labeling, comparing their properties and performances. Section IV proposes an original view to the problem of labeling by means of decision tables and decision trees, focusing on reducing the cost of conditions testing,

Manuscript received July 15, 2009; revised January 23, 2010. First published March 11, 2010; current version published May 14, 2010. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Ying Wu.

The authors are with the Dipartimento di Ingegneria dell'Informazione, Università degli Studi di Modena e Reggio Emilia, Emilia 41125, Italy (e-mail: daniele.borghesani@unimore.it).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TIP.2010.2044963

then Section V details our 2×2 block neighborhood analysis. Finally, Section VI demonstrates the effectiveness of our approach with experiments on a wide variety of images, in comparison with other state of the art methods. Concluding remarks are given in Section VII.

II. Connected Components Labeling

In order to clearly present our solution for the labeling problem, it is convenient to fix the basic notations and definitions related to the concepts of neighborhood and connectivity.

Let us call I an image defined over a 2-D rectangular lattice $\mathcal{L}$, and $I(p)$ the value at pixel $p \in \mathcal{L}$, with $p = (p_x, p_y)$. The *four-neighborhood* and the *eight-neighborhood* of a pixel p can be, respectively, defined as

$$\mathcal{N}_4(p) = \{q \in \mathcal{L} | |p_x - q_x| + |p_y - q_y| \leq 1\} \quad (1)$$

$$\mathcal{N}_8(p) = \{q \in \mathcal{L} | \max(|p_x - q_x|, |p_y - q_y|) \leq 1\}. \quad (2)$$

In other words, $\mathcal{N}_4$ is the set of points with null or unitary *city block* distance ($\|\cdot\|_1$ norm), while $\mathcal{N}_8$ is the set of points with null or unitary *chessboard* distance ($\|\cdot\|_\infty$ norm). Thus, two pixels p and q are said to be *four-neighbors* if $q \in \mathcal{N}_4(p)$, which also implies $p \in \mathcal{N}_4(q)$, and they are said to be *eight-neighbors* if $q \in \mathcal{N}_8(p)$, which also implies $p \in \mathcal{N}_8(q)$. Furthermore, it is clear that $\mathcal{N}_4(p) \subset \mathcal{N}_8(p)$, i.e., if two pixels are four-neighbors, they are also eight-neighbors. We will write $\mathcal{N}$ to generically identify a neighborhood when either definition could be used.

Given a subset $\mathcal{S}$ of $\mathcal{L}$, we define the relation of *connectivity* $\Diamond$ between two pixels $p, q \in \mathcal{S}$ as

$$p \Diamond q \Leftrightarrow \exists \{s_i \in \mathcal{S} | s_1 = p, s_{n+1} = q, s_{i+1} \in \mathcal{N}(s_i), i = 1, \ldots, n\} \quad (3)$$

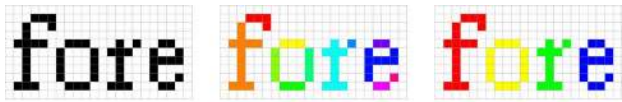
that is if it is possible to find a sequence of neighboring points of $\mathcal{S}$ starting from p and leading to q [4]. Thus we say that p *is connected to* q if the relation $p \Diamond q$ is satisfied. *Connectivity* is an equivalence relation, since the properties of *reflexivity*, *symmetry*, and *transitivity* hold. A subset $\mathcal{C}$ of $\mathcal{L}$, defined by a common property obtained from the pixel values, is called a *connected component* if $p \Diamond q, \forall p, q \in \mathcal{C}$, i.e., if any two points of the subset are connected.

Usually, labeling algorithms deal with binary images, i.e., images where points can only take binary values. Important or meaningful regions, such as the result of segmentation algorithms, are called *foreground* ($\mathcal{F}$), while the other pixels constitute the *background* ($\mathcal{B}$). Conventionally we will assign value 1 to foreground pixels and 0 to background pixels, so

$$\mathcal{F} = \{p \in \mathcal{L} | I(p) = 1\} \quad (4)$$

$$\mathcal{B} = \{p \in \mathcal{L} | I(p) = 0\}. \quad (5)$$

Clearly, $\mathcal{F} \cup \mathcal{B} = \mathcal{L}$ and $\mathcal{F} \cap \mathcal{B} = \emptyset$. Since the property of interest is normally to be part of the foreground with respect to the background, the common choice in binary images is to choose *eight-connectivity* for the foreground regions, and *four-connectivity* for background regions. This choice better matches our usual perception of distinct objects, as in Fig. 1. Accordingly to the Gestalt Theory of perception, our senses operate the closure property perceiving objects as a whole even if they are loosely



(a) (b) (c)

Fig. 1. (a) Examples of binary image depicting text, (b) its labeling considering four-connectivity, and finally, (c) eight-connectivity.

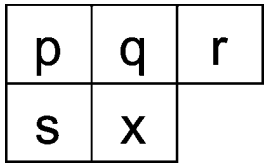


Fig. 2. Pixel mask $\mathcal{M}(x)$ used to compute the label of pixel x, and to evaluate possible equivalences in raster scan techniques.

connected as happens in the eight-connectivity case, so that we can easily read the letters Fig. 1(c).

Labeling algorithms take care of the assignment of a unique identifier (an integer value, namely *label*) to every connected component of the image, in order to give the possibility to refer to it in the next processing steps. It is common practice to reserve label 0 for background pixels. Analogously to the definition of image I, we also define the function $L : \mathcal{L} \rightarrow \mathbb{N}_0$, which maps a pixel to a label identifying the connected component to which it belongs. Depending on the search order and the region connectivity, during a labeling algorithm execution two pixels in the same connected component could be assigned provisionally different non zero labels: this implies that the two labels must be considered *equivalent*. Formally, given $p, q \in \mathcal{F}$, $p \Diamond q \Leftrightarrow L(p) \equiv L(q)$. We can define the *equivalence class* of a label $L(p)$ as

$$[L(p)] = \{\lambda \in \mathbb{N} | \lambda \equiv L(p)\}. \quad (6)$$

It can be observed that if $L(p) \equiv L(q)$ then $[L(p)] = [L(q)]$, further implying that $L(p) \in [L(q)]$ and $L(q) \in [L(p)]$.

The majority of images are stored in raster scan order, so the most common technique for connected components labeling applies sequential local operations in that order, as first introduced in [4]. This is classically performed in the following three steps:

1) first image scan (provisional labels assignment and collection of label equivalences);
2) equivalences resolution (equivalence classes creation);
3) second image scan (final label assignment).

During the first step, $L(x) \leftarrow 0, \forall x \in \mathcal{B}$. Instead, for each pixel $x \in \mathcal{F}$, $L(x)$ is evaluated by only looking at the labels of its already processed neighbors. When using eight-connectivity, these pixels belong to the scanning *mask* $\mathcal{M}(x) \subset \mathcal{N}_8(x)$, shown in Fig. 2. More in detail, given x the pixel with coordinates (i, j) in the lattice identified as $x = l_{i,j}$, we can define $\mathcal{M}(x) = \{p = l_{i-1,j-1}, q = l_{i,j-1}, r = l_{i+1,j-1}, s = l_{i-1,j}\}$. As mentioned before, during the scanning procedure, the same connected component can be assigned different (*provisional*) labels, so all algorithms adopt some mechanism to keep track of the possible equivalences.

In the second step, all the provisional labels must be segregated into disjoint sets, or disjoint equivalence classes. As soon

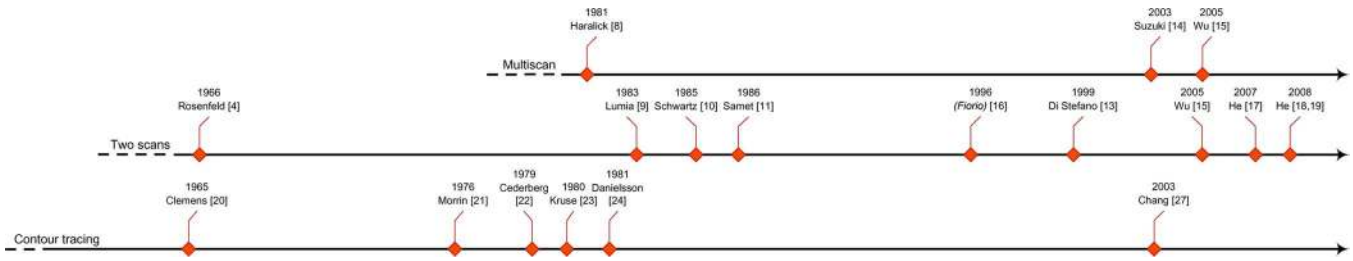


Fig. 3. Timeline showing the evolution of the labeling algorithms. Algorithms are referenced by first author name.

as an unprocessed equivalence is considered, a "merging" between classes is needed, that is some operation which allows to mark as equivalent all labels involved. Most of the recent optimizations introduced in modern connected components labeling techniques aim at increase the efficiency of this step.

Once the equivalences have been eventually solved, in the third step a second pass over the image is performed in order to assign to each foreground pixel the representative label of its equivalence class. Usually, the class representative is unique and is set to be the minimum label value in the class.

The *disjoint set union* problem (step 2) has been widely studied during the past decades [5]. The problem consists of maintaining a collection of disjoint sets under the operation of union. More precisely, the problem is to perform a sequence of operations of the following two kinds on disjoint sets.

Union (A, B): Combine the two sets A and B into a new set named A.

Find (x): Return the name of the unique set containing the element x.

The introduction of efficient *Union-Find* algorithms allows the inclusion of the equivalences resolution step directly into the first image scan, removing the need of collecting equivalences. This constitutes the basic structure of all modern labeling algorithms, which perform online equivalences resolution.

III. Evolution of Labeling Algorithms: A Review

In this section, we provide a historical view of the different literature works, discussing how they contributed to the current state of the art, and how much they are still suitable for modern architectures. This review does not aim to be comprehensive of all the proposals, but mainly aims at providing an overview of the more relevant research trends.

Two wide classes of labeling algorithms will not be covered by our analysis. The first one is the class of parallel algorithms which has been extensively studied up to the first half of the 1990's (see for instance [6]). These algorithms mainly address the massive data parallelism of 1980's architectures and do not readily apply to current common workstations parallelism, such as instruction level, thread level and so on. The second class comprehends algorithms defined for hierarchical image representations (for example quadtrees [7]) initially studied for accessing large images stored in secondary memory. We excluded them because the vast majority of images is currently stored in sequential fashion, since the large availability of main memory in modern computer architecture do not limit their full storage anymore.

In order to summarize this historical review, Fig. 3 proposes a temporal positioning of the presented algorithms, classified based on the methodology adopted for the scan over the image. In the first row we list the approaches based on iterated multiple scans of the image. The middle row lists the approaches that, starting from the pioneering work of Rosenfeld [4], exploit two scans only, while in the third row we list some of the algorithms that are based on contour tracing techniques, so exploiting a single scan over the image.

The first work proposed for image labeling dates back to Rosenfeld *et al.* in 1966 [4]. This algorithm can be considered the most classical approach to labeling, and it is based on a raster scan of the image. It produces an output image containing the labeling result, and it stores the "redundancies" (i.e., equivalences) of the labels in an equivalences table with all the neighborhood references. The redundancies are then solved processing the table by repeatedly using an unspecified sorting algorithm and removing redundant entries. Finally the resulting labels are updated to the output image with a further pass, exploiting the solved equivalences table. This method requires an adequate memory allocation for the final image and the equivalence table, and a high computational cost due to the repeated use of sorting algorithms.

To tackle these limitations, in particular the memory requirements, an improvement was proposed by Haralick *et al.* [8]. This algorithm does not use any equivalences table and no extra space, by iteratively performing forward and backward raster scan passes over the output image to solve the equivalences, exploiting only local neighborhood information. This technique, although requiring very little memory, clearly turns out to be computationally very expensive when the size of the binary image to analyze increases.

Lumia *et al.* [9] observed that both previous algorithms perform poorly on 1983 virtual memory computers because of page faults, so they proposed a mix of the two approaches trying to keep the equivalences table as small as possible and saving memory usage. In this algorithm a forward and a backward scan are sufficient to complete the labeling, but at the end of each row the collected equivalences are solved and another pass immediately updates that row labels. Therefore four passes over the data are indeed used by this algorithm. The technique to solve label equivalences was left unspecified.

Schwartz *et al.* [10] further explored this approach, in order to avoid the storage of the output image, which would have required too much memory. Thus they use a sort of *run length*-based approach (without naming it as such), which produces a compact representation of the label equivalences. In this way,

after a forward and a backward scan, they can output an auxiliary structure which can be used to infer a pixel label.

Samet *et al.* [11] were the first researchers who clearly named the equivalence resolution problem as the *disjoint-set union problem*, about 20 years ago. This is an important achievement, since a quasi linear solution for this problem is available: the so-called *Union-Find* algorithm, from the name of the basic operations involved. Also this algorithm is executed in two passes. The first pass creates an intermediate file consisting of image elements and equivalence classes while the second pass processes this file in reverse order, and assigns final labels to each image element. The proposal in [11] was definitely complex, since it also targeted quad-tree-based image representations and it was aimed at not keeping the equivalences in memory. Then in [12] a general definition of this algorithm for arbitrary image representations has been proposed in detail.

The Union-Find algorithm is the basis of most of the modern approaches for label resolution. As a new pixel is computed, the equivalence label is resolved: while the previous approaches generally performed first a collection of labels and at the end the resolution and the Union of equivalence classes, this new approach guarantees that at each pixel the structure is up to date.

A relevant paper in this evolution is [13] where Di Stefano *et al.* proposed an online label resolution algorithm with an array-based structure to store the label equivalences. The array-based data structure has the advantage to reduce the memory required and to speed up the retrieval of elements without the use of pointer dereferencing. They do not explicitly name their equivalences resolution algorithm as Union-Find, and their solution requires multiple searches over the array at every Union operation.

In 2003, Suzuki *et al.* [14] resumed Haralick's approach of the multiscan strategy over the image, but with the inclusion of a small equivalence array: they provided a linear-time algorithm that in most cases requires four passes. The label resolution is performed exploiting array-based data structures, and each foreground pixel takes the minimum class of the neighboring foreground pixels classes. An important addition to this proposal is provided in an appendix in the form of a lookup table (LUT) of all possible neighborhoods, which allows to reduce computational times and costs by avoiding unnecessary Union operations.

In 2005, Wu *et al.* in [15] defined an interesting optimization to reduce the number of labels, in order to increase the performance of Suzuki's approach. They exploited a decision tree to minimize the number of neighboring pixels to be visited in order to evaluate the label of the current pixel. In a eight-connected components neighborhood, among all the neighboring pixels, often only one of them is sufficient to determine the label of the current pixel. This work in particular inspired our proposal to define a systematic way to minimize the comparisons, thus the necessary *Union* and *Find* operations. In the same paper, the authors proposed another strategy to improve the Union-Find algorithm of Fiorio *et al.* [16] exploiting an array-based data structure. For each equivalence array a path compression is performed to compute the root, in order to directly keep the minimum equivalent label within each equivalence array, without requiring an additional stage as in Fiorio's technique.

In 2007, He (in collaboration with Suzuki) proposed another fast approach in the form of a two scan algorithm [17]. The data structure used to manage the label resolution is implemented using three arrays in order to link the sets of equivalent classes without the use of pointers. Adopting this data structure, two algorithms have then been proposed: in [18] a run-based first scan is employed, while in [19] a decision tree (similarly to [15]) optimizes the neighborhood exploration to apply merging only when needed.

Another group of researchers has taken a radically different approach to this problem, starting from Clemens [20], which in his Ph.D. thesis was one of the first to provide a link between the concept of connected components labeling and contour tracing. He described an *hexagon tracing routine* (implemented in hardware) able to extract the outer contours of a character, remove the interior with a mathematical morphology approach, and further tracing the inner edges. Strictly speaking, his proposal is not a labeling algorithm, but provides the basis later employed for this task.

In 1976, Morrin [21] developed a binary image compression technique, which is composed by raster scanning and contour following technique. As soon as the raster scan encounters a boundary the algorithm starts to follow it, peeling off one layer of pixels after another until the object is exhausted. Raster scan is then resumed. Only the first boundary trace is stored as a contour. While effective and requiring a minimum amount of auxiliary memory, the multiple contour following steps are rather time consuming.

Cederberg [22] in 1979 proposed a raster scan approach, which is able to produce a set of partial contours, max points, and min points. These local information allow to later reconstruct the complete contour. In his work a solution for producing an ordered tree of contour inclusions is also provided, and this could be employed to assign different labels to the various connected components, but no detail is given on the computational complexity for this specific task.

In 1980, Kruse [23] proposed a fast stack-based algorithm for segmentation of connected components in binary images. In his terminology, segmentation is a sort of superset of labeling, in which not only every foreground connected component is given a different label, but also every background connected component is distinguished. Segmentation may be obviously used to obtain labeling if needed. His approach again uses a raster scan plus contour following routine. After encountering the first object pixel, the algorithm starts following the contour, and during this stage it *tags* the pixels having a background pixel on the right, then the raster scan is resumed. When a labeled pixel is encountered, its label is pushed on a stack (we are "entering" a connected component). Later, when we meet a tagged pixel, we know that we are "exiting" that component and we can pop the stack. The combined use of stack information and tagging allows to completely reconstruct the original image components.

Danielsson [24] in 1981 further improved this approach avoiding the need of both the stack and the tagging, by substituting the tag with a special temporary "0" label assigned to the first background pixel, immediately to the right of a contour point, which would have been tagged by Kruse's algorithm.

Both these algorithms have been extended to the nonbinary case [25], [26].

Later the most relevant work on this branch of research is given by Chang *et al.* [27] in 2003. Their approach strongly resembles Danielsson's and it is based on a single pass over the image exploiting contour tracing. Their technique clockwise tags all pixels in both the contour and the immediately external background in a single operation. When during the raster scan an untagged boundary is found, a counter clockwise contour tracing is performed for internal contours. This technique proved to be very 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. Moreover the algorithm can naturally output the connected components contours, if needed.

As far as we write, the algorithm presented in 2008 by He *et al.* [19] represents the state of the art for connected components analysis. This proposal is based on a raster scan over the image and it embraces the Union-Find approach for equivalences resolution, performed online as soon as the equivalences are found. There are two key novelties in this algorithm.

The first novelty is the fast technique implemented to perform the Union-Find, described in [17]. It is based on a set of three arrays in order to link the sets of equivalent classes without the use of pointers. An *rl_table* array contains information about the representative label of each class, a n_label array contains the index of the next equivalent label, thus providing a linked list structure, finally a t_label array contains the index of the last label of the list. This array-based structure turns out to be very effective, combining the performances of arrays with the benefits of a list-like structure in order to solve equivalences without scanning an entire array of equivalences. The second novelty is the optimization performed for the neighborhood computation. Accessing in a clever way the labels of the neighboring pixels, the number of *resolve* operations (the name used in [19] for the *union* operation) to perform are minimized, avoiding to solve equivalences already solved by previous steps of the algorithm. In this way, authors significantly improved performance, since these actions are the most time consuming computations within the algorithm. In this paper, we adopted the same efficient data structure for label resolution, but we mainly focus on the neighborhood computation proposing a whole new way to speed up the process.

IV. Decision Tables and Decision Trees

The procedure of collecting labels and solving equivalences may be described by a *command execution metaphor*: the current and neighboring pixels provide a binary command word, interpreting foreground pixels as 1s and background pixels as 0s. A different action must be taken based on the command received.

We may identify four different types of actions: *no action* is performed if the current pixel does not belong to the foreground, a *new label* is created when the neighborhood is only composed of background pixels, an *assign* action gives the current pixel the label of a neighbor when no conflict occurs (either only one pixel is foreground or all pixels share the same label), and finally a *merge* action is performed to solve an equivalence between two or more classes and a representative is assigned to the current pixel. The relation between the commands and the corresponding actions may be conveniently described by means of a *decision table* [28].

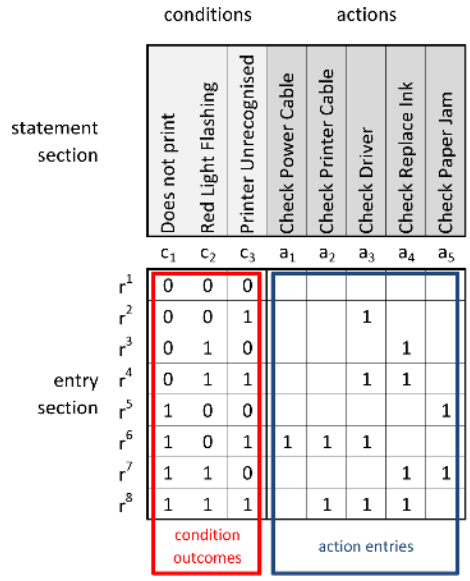


Fig. 4. Decision table example, showing a hypothetical troubleshooting checklist for solving printing failures. Note that we use a vertical layout, which is more suitable when dealing with a large number of conditions.

A *decision table* is a tabular form that presents a set of conditions and their corresponding actions. A decision table is divided into four quadrants: an example is provided in Fig. 4. The statement section reports a set of conditions which must be tested and a list of actions to perform. Each combination of condition entries (*condition outcomes*) is paired to an *action entry*. In the action entries, a column is marked, for example with a "1", to specify whether the corresponding action is to be performed. If the conditions outcomes may only be true or false, the table is called *limited entry decision table* [29]. These will be the tables type used throughout this manuscript.

More formally, we call $c_1, \ldots, c_L$ the list of conditions. If we call S the *system status* (the lights on a printer, the service quality, the current pixel neighborhood, etc., ...), a condition is a function of S which returns a boolean value. The list of actions is identified by $a_1, \ldots, a_M$, where an action is a procedure or operation which can be executed. Every row in the entry section is called a rule $\mathbf{r}^1, \ldots, \mathbf{r}^N$, which is a pair of Boolean vectors of condition outcomes o_j^i and action entries e_k^i, denoting with i the rules index, with j the conditions index, and with k the actions index. A *decision table* may thus be described as

$$\mathcal{DT} = \{\mathbf{r}^1, \ldots, \mathbf{r}^N\} = \{(\mathbf{o}^1, \mathbf{e}^1), \ldots, (\mathbf{o}^N, \mathbf{e}^N)\}. \quad (7)$$

The straightforward interpretation of a decision table is that the actions a_k corresponding to *true* entries e_k^i should be performed if the outcome $\mathbf{o}^i$ is obtained when testing the conditions. Formally, given the status S, we write

$$\mathbf{c}(S) = \mathbf{o}^i \Leftrightarrow o_j^i = c_j(S), \forall j = 1, \ldots, J \quad (8)$$

						assign				merge										
x	p	q	r	s	no action	new label	x = p	x = q	x = r	x = s	x = p+q	x = p+r	x = p+s	x = q+r	x = q+s	x = r+s	x = p+q+r	x = p+q+s	x = p+r+s	x = q+r+s	x = p+q+r+s
0	-	-	-	-	1																
1	0	0	0	0		1															
1	1	0	0	0			1														
1	0	1	0	0				1													
1	0	0	1	0					1												
1	0	0	0	1						1											
1	1	1	0	0							1										
1	1	0	1	0								1									
1	1	0	0	1									1								
1	0	1	1	0										1							
1	0	1	0	1											1						
1	0	0	1	1												1					
1	1	1	1	0													1				
1	1	1	0	1														1			
1	1	0	1	1															1		
1	0	1	1	1																1	
1	1	1	1	1																	1

Fig. 5. Initial decision table providing a different action for every pixel configuration. To produce a more compact visualization, we reduce redundant logic by means of the indifference condition "−," whose values do not affect the decision and always result in the same action. In the condition section the pixel letter means that we have to test if that pixel belongs to the foreground. In the action section, the "+" operator is used to indicate a *merge* between the labels of pixels indicated, while the "=" means that pixel x is assigned any of the labels of the right operands.

so

$$\texttt{if } \mathbf{c}(S) = \mathbf{o}^i \texttt{then execute} \{a_k | e_k^i = 1\}_{k=1,\ldots,K}. \quad (9)$$

The execute operation applied to a set of actions $\{a_k\}$, as in (9), classically requires the execution of **all** the actions in the set, that is all actions marked with 1s in the action entries vector: we call this behavior an AND-decision table. For our problem we define a different meaning for this operation. We define an OR-decision table, in which **any** of the actions in the set may be performed in order to satisfy the corresponding condition.

Note that this situation does not imply that the actions are redundant, in the sense that two or more actions are always equivalent. In fact, the result of doing any action in the execution set is the same only when a particular condition is verified.

A. Modeling Raster Scan Labeling With Decision Tables

In order to describe the behavior of a labeling algorithm with a decision table, we need to define the conditions to be checked and the corresponding actions to take. For this problem, as we already mentioned, the conditions are given by the fact that the current pixel x and the four neighboring ones in mask $\mathcal{M}(x)$ belong to the foreground. The conditions outcomes are given by all possible combinations of five Boolean variables, leading to a decision table with 32 rules. The actions belong to four classes: *no action*, *new label*, *assign*, and *merge*. Fig. 5 shows a basic decision table with these conditions and actions.

The action entries are obtained applying the following considerations:

1) *no action* if $x \in \mathcal{B}$;
2) $L(x) =$ *new label* if $x \in \mathcal{F} \wedge \mathcal{M}(x) \subset \mathcal{B}$;
3) $L(x) = L(y)$ if $x \in \mathcal{F} \wedge \exists! \, y \in \mathcal{M}(x) | y \in \mathcal{F}$;
4) $L(x) = \text{merge}\left(\{L(y) | y \in \mathcal{M}(x) \cap \mathcal{F}\}\right)$ otherwise.

Using these considerations the equivalences are solved and a representative (provisional) label is associated to the current pixel x. The process then moves ahead to the next pixel and the next neighborhood accordingly.

							assign				merge	
x	p	q	r	s	no action	new label	x = p	x = q	x = r	x = s	x = p+r	x = r+s
0	-	-	-	-	**1**							
1	0	0	0	0		**1**						
1	1	0	0	0			**1**					
1	0	1	0	0				**1**				
1	0	0	1	0					**1**			
1	0	0	0	1						**1**		
1	1	1	0	0			*1*	**1**				
1	1	0	1	0							**1**	
1	1	0	0	1			*1*			**1**		
1	0	1	1	0				**1**	*1*			
1	0	1	0	1				**1**		*1*		
1	0	0	1	1								**1**
1	1	1	1	0			*1*	**1**	*1*			
1	1	1	0	1			*1*	**1**		*1*		
1	1	0	1	1							*1*	**1**
1	0	1	1	1				**1**	*1*	*1*		
1	1	1	1	1			*1*	**1**	*1*	*1*		

Fig. 6. Resulting OR-decision table for labeling. Bold 1's are selected with the procedure described in Section IV-C.

First, *merge* operations have a higher computational cost with respect to an *assign*, so we should reduce at the minimum the number of these operations in order to improve the performance of labeling. Similarly a *merge* between two labels is computationally cheaper than a *merge* between three labels. Thus, exploiting the OR-decision table formalism, we can substitute whenever is possible all *merge* operations with equivalent *assign* operations. In the matter of facts, merging an equivalence class with itself returns the same class again: for example when $p, q \in \mathcal{F}$ and $L(p) \equiv L(q)$, the *merge* operation has no effect and assigning a representative label from the merge outcome or any of $L(p)$ or $L(q)$ has the same result. So in these cases all the action entries of $L(x) = L(p)$, of $L(x) = L(q)$ and of $L(x) = \text{merge}\left(\{L(p), L(q)\}\right)$ should be set to 1.

The problem with this reasoning is of course that we would need to add a condition for checking if $L(p) \equiv L(q)$, complicating enormously the decision process, since every condition doubles the number of rules. But, is this condition really necessary? No, because we can further notice that if we exploit an algorithm with online equivalences resolution, p and q **cannot have different labels**. Since they are eight-connected, if both of them are foreground, during the analysis of q a label equivalent to $L(p)$ would have been assigned to $L(q)$. This allows us to always remove merge operations between eight-connected pixels, substituting them with assignments of the involved pixels labels.

Extending the same considerations throughout the whole rule set, we obtain an effective "compression" of the table, as shown in Fig. 6. To obtain the table, when an operation could be substituted with a cheaper one, the more costly was removed from the table. Most of the *merge* operations are avoided, obtaining an OR-decision table with multiple alternatives between *assign* operations, and only in a single case between *merge* operations. Moreover the reduction leads also to the exclusion of many unnecessary actions (for example, the merge between p and q) without affecting the algorithm outcome.

Summarizing, connected components labeling based on OR-decision tables means to retrieve the condition outcome given the current status $\mathbf{c}(S) = \mathbf{o}^i$ and select one action

among the alternatives a_k corresponding to $e_k^i = 1$, with $k = 1, \ldots, K$. More details regarding the heuristic adopted to select the final single action will be discussed in Section IV-C.

B. Reducing the Cost of Conditions Testing: Decision Trees

The definition of decision tables requires all conditions $c_1, \ldots, c_L$ to be tested in order to select the corresponding action to be executed. Testing the conditions of the decision table has a cost which is related to the number of conditions and to the computational cost of each test. If we assume that each test has the same cost, which is true in our application, the only parameter which can be optimized is the number of conditions to be tested.

There are a number of cases in which not all conditions must be tested in order to perform the corresponding action. For example in the first row of the decision table of Fig. 6, if $x \in \mathcal{B}$ all the other conditions are useless, since the outcome will always be *no action*. This straightforward observation suggests that the order with which the conditions are verified impacts on the number of tests required, thus on the total cost of testing.

What we are now looking for is the optimal ordering of conditions tests, which effectively produces a sequence of tests, depending on the outcome of previous tests. This is well represented by an optimal *decision tree*: the sequence requiring the minimum number of tests corresponds to the decision tree with the minimum number of nodes. The transformation of the decision table in an optimal decision tree has been deeply studied in the past and we use the dynamic programming technique proposed by Schumacher [30], which guarantees to obtain an optimal solution.

One of the basic concepts involved in the creation of a simplified tree from a decision table is that if two branches lead to the same action the condition from which they originate may be removed. With a binary notation, if both the condition outcomes 10110 and 11110 require the execution of action 4, we can write that 1–110 requires the execution of action 4, thus removing the need of testing condition 2, with the use of a dash implies that both 0 or 1 may be substituted in that condition, representing the concept of *indifference*. The saving given by the removal of a test condition is called *gain* in the algorithm, and we conventionally set it to 1.

The conversion of a decision table (with n conditions) to a decision tree can be interpreted as the partitioning of an n-dimensional hypercube (n-cube in short) where the vertexes correspond to the 2^n possible rules. Including the concept of indifferences, a t-cube corresponds to a set of rules and can be specified as an n-vector of t dashes and $n - t$ 0's and 1's. For example, 01-0- is the 2-cube consisting of the four rules $\{01000, 01001, 01100, 01101\}$. In summary, Schumacher's algorithm proceeds in steps as follows.

- *Step 0:* All 0-cubes, that is all rules, are associated to a single corresponding action and a starting gain of 0; this means that if we need to evaluate the complete set of conditions, we do not get any computational saving.
- *Step t*: All t-cubes are enumerated. Every t-cube may be produced by the merge of two $(t-1)$-cubes in t different ways (for example 01-0- may be produced by the merge of $\{01-00, 01-01\}$ or of $\{0100-, 0110-\}$). For each of these ways of producing the t-cube (denoted as s in the following formulas) we compute the corresponding gain G_s as

$$G_s = G_s^0 + G_s^1 + \delta\left[A_s^0 - A_s^1\right] \tag{10}$$

where G_s^0 and G_s^1 are the gains of the two $(t-1)$-cubes in configuration s, and A_s^0 and A_s^1 are the corresponding actions to be executed. δ is the Kronecker function that provides a unitary gain if the two actions are the same or no gain otherwise, modeling the fact that if the actions are the same we "gain" the opportunity to save a test. The gain assigned to the t-cubes is the maximum of all G_s, which means that we choose to test the condition allowing the maximum saving.
Analogously we have to assign an action to the t-cube. This may be a real action if all rules of the t-cube are associated to the same action, otherwise it is 0, a conventional way of expressing the fact that we need to branch to choose which action to perform. In formulas

$$A = A_s^0 \cdot \delta[A_s^0 - A_s^1] \tag{11}$$

where s may be chosen arbitrarily, since the result is always the same.

The algorithm continues to execute Step t until $t = n$, which effectively produces a single vector of dashes. The tree may be constructed by recursively tracing back through the merges at each t-cube. A leaf is reached if a t-cube has an action $A \neq 0$.

C. Action Selection in OR-Decision Tables

To produce an optimal tree, the described algorithm [30] requires a decision table where every rule leads to *a single action*, that we will call *single action decision table*. This requirement forces us to convert the previously described decision tables into this representation. Starting from an AND-decision table, a single action decision table is straightforward to obtain: for every distinct row of action entries $\mathbf{e}^i$ we can define a *complex action* in the form of the set of actions $A_l = \{a_k | e_k^i = 1\}_{k=1,\ldots,K}$. The execution of A_l requires the execution of all actions in A_l. Now we can associate to every condition outcome an integer index, which points to the corresponding complex action.

Algorithm 1 Greedy selection of the actions to perform in OR-decision tables

$\mathcal{I} = \{1, \ldots, K\}$ ▷ Define actions indexes set
while $\mathcal{I} \neq \varnothing$ **do**
  $k^* \leftarrow \arg\max_{k \in \mathcal{I}} \sum_{i=1}^{N} e_k^i$ ▷ Find most frequent action
  for $i = 1, \ldots, N$ ▷ Remove equivalent actions
    if $e_{k^*}^i = 1$ **then**
      $e_k^i \leftarrow 0, \forall k \neq k^*$
    end if
  end for
  $\mathcal{I} = \mathcal{I} - \{k^*\}$ ▷ This action has been done
end while

Instead in OR-decision tables only one of the different alternatives provided in $\mathbf{e}^i$ must be selected. While an arbitrary selection does not change the result of the algorithm, the optimal tree derived from a decision table implementing these arbitrary choices may be different. How do we select the best combination of actions, in order to minimize the final decision tree? Exhaustive search quickly becomes infeasible when the number of conditions increases, thus we propose an heuristic greedy procedure.

In accordance with the issues of boolean optimization in combinatorial logic, the rationale behind our approach is that the more rules require the execution of the same action, the more likely it will be to find large k-cubes covering that action. We propose a greedy approach: the number of occurrences of each action entry is counted; iteratively the most common one is selected, and for each rule where this entry is present all the other entries are removed, until no more changes are required. In case two actions have the same number of entries, we arbitrarily chose the one with lower index. The resulting table after applying this process is shown in Fig. 6, with bold faces 1's. The following Algorithm 1 formalizes the procedure.

The described approach does not always lead to an optimal selection, but the result is often optimal or nearly optimal, based on many different experiments. This is particularly true when the distribution of the actions frequencies is strongly non uniform. For example, from the original OR-decision table in Fig. 6, it is possible to derive 3456 different decision tables, by selecting all permutations of equivalent actions. Using Algorithm 1 only two actions are chosen arbitrarily, leading to four possible equivalent decision trees. All of these have the same number of nodes and are optimal (in this case we were able to test all of the 3456 possibilities). One of these trees is the one described by He *et al.* in [19].

In his proposal, He *et al.* summarize the alternatives in a truth table, then employ a Karnaugh map to provide a synthesis of the logic function under which the *resolve* operation may be avoided. This logic function requires all nearby pixels, so his approach is to manually derive an optimal ordering on the conditions to be checked, giving a short circuit exit in some cases.

In conclusion, we provided an algorithmic solution to the optimal neighborhood exploration problem, which is equivalent to the state of the art. Nevertheless, with respect to previous approaches, our solution has an important added value: it can be naturally extended to larger problems, without requiring any empirical workaround. In the following, we introduce a novel approach to neighborhood exploration, which takes advantage of the described technique.

V. 2×2 Block Neighborhood Analysis

The availability of the previously described technique allows us to enlarge our neighborhood exploration window, with the aim to further speed up the connected components labeling process. As previously reported in [31], the key idea of our proposal starts from two very straightforward observations: 1) when using eight-connection, the pixels of a 2×2 square are all connected to each other and 2) a 2×2 square is the largest set of pixels in which this property holds. This implies that all foreground pixels in a the block will share the same label at the end of the computation. For this reason, we propose to scan the image moving on a 2×2 pixel grid applying, instead of the classical neighborhood of Fig. 2, an extended mask of five 2×2 blocks, as shown in Fig. 7.

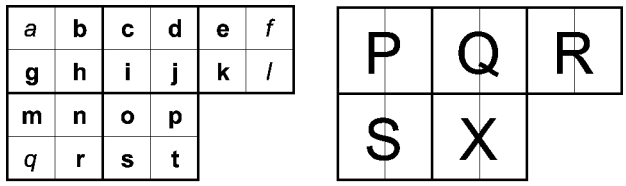


Fig. 7. Mask used for 2×2 block-based labeling is shown. (a) Gives the identifiers of the single pixels employed in the algorithm (a, f, l, and q are not used), while (b) provides the blocks identifiers.

Scanning the image with this larger grain has the advantage to allow the labeling of four pixels at the same time. The number of provisional labels created during the first scan is roughly reduced by a factor of four, and we need to apply much less unions, since labels equivalence is implicitly solved within the blocks. Moreover a single label is stored for the whole block.

On the other hand, the neighborhood to consider now is much larger. The standard procedure (that is to consider all the pixels in the neighborhood) greatly increases computational time due to the number of memory accesses and merge operations required. Likewise a manual approach for an effective neighborhood exploration is unfeasible since we must deal with much more than five pixels for each labeling operation, and the amount of combinations to explore is enormous. But the general procedure described in the previous section is designed to provide an effective way to face the optimization in this situation.

The new scanning procedure may require also the same pixel to be checked multiple times, but the impact of this problem is greatly reduced by our optimized pixel access scheme. Finally, a second scan requires to access again the original image to check which pixels in the block require their label to be set. Overall the advantages will be shown to largely overcome the additional work required in the following stage.

Employing all pixels in the new mask of Fig. 7, we would need to work with 20 pixels: for this reason, the decision table would have $L = 20$ conditions, and $N = 2^{20}$ possible configurations of condition outcomes. However, we can notice that not all those pixels are necessary to compute labeling information. In particular pixels a, f, l, q do not provide eight-connection between blocks of the mask and can be ignored. We thus need to deal with $L = 16$ pixels (thus conditions), for a total amount of 2^{16} possible combinations.

Since manually specifying the action entries for all 65 536 combinations is impractical, we choose not to directly deal with the condition outcomes but abstracting the relations between blocks. For this reason, given two blocks X and Y, we introduce the concept of *block connectivity* $\bowtie$, defined as

$$X \bowtie Y \Leftrightarrow \exists x \in X \cap \mathcal{F}, y \in Y \cap \mathcal{F} | x \in \mathcal{N}(y). \qquad (12)$$

Block connectivities provide sufficient information to perform labeling: the connectivity between two blocks implies that all

foreground pixels of the two blocks share the same label. For the sake of clarity, we will call *pixel based decision table* (PBDT) the decision table defined over the 16 pixels conditions, and *block based decision table* (BBDT) the one defined over the block neighborhoods.

Similarly to the scanning mask in Fig. 2, we call each block with the corresponding uppercase letter: $P = \{a,b,g,h\}$, $Q = \{c,d,i,j\}$, $R = \{e,f,k,l\}$, $S = \{m,n,q,r\}$, $X = \{o,p,s,t\}$, as shown in Fig. 7. Block X corresponds to the current block under analysis. Specifically, we define the following conditions:

c_1: $X \bowtie P \stackrel{\text{def}}{=} h \in \mathcal{F} \wedge o \in \mathcal{F}$;
c_2: $X \bowtie Q \stackrel{\text{def}}{=} (i \in \mathcal{F} \vee j \in \mathcal{F}) \wedge (o \in \mathcal{F} \vee p \in \mathcal{F})$;
c_3: $X \bowtie R \stackrel{\text{def}}{=} k \in \mathcal{F} \wedge p \in \mathcal{F}$;
c_4: $X \bowtie S \stackrel{\text{def}}{=} (n \in \mathcal{F} \vee r \in \mathcal{F}) \wedge (o \in \mathcal{F} \vee s \in \mathcal{F})$.

Further analysis evidences that blocks have some interdependencies: to completely describe the pixels connectivities, we must consider not only the connectivity relation between the current block X and each individual neighboring block, but also all the connectivity relation between the blocks. In this new perspective we define the following four additional conditions:

c_5: $P \bowtie Q \stackrel{\text{def}}{=} (b \in \mathcal{F} \vee h \in \mathcal{F}) \wedge (c \in \mathcal{F} \vee i \in \mathcal{F})$;
c_6: $Q \bowtie R \stackrel{\text{def}}{=} (d \in \mathcal{F} \vee j \in \mathcal{F}) \wedge (e \in \mathcal{F} \vee k \in \mathcal{F})$;
c_7: $S \bowtie P \stackrel{\text{def}}{=} (g \in \mathcal{F} \vee h \in \mathcal{F}) \wedge (m \in \mathcal{F} \vee n \in \mathcal{F})$;
c_8: $S \bowtie Q \stackrel{\text{def}}{=} i \in \mathcal{F} \wedge n \in \mathcal{F}$.

A last condition which needs to be considered is whether block X contains any foreground pixel:

c_9: $X \in \mathcal{F} \stackrel{\text{def}}{=} o \in \mathcal{F} \vee p \in \mathcal{F} \vee s \in \mathcal{F} \vee t \in \mathcal{F}$.

We have eventually defined nine Boolean conditions, with a total amount of 512 combinations, which allow us to convey the same knowledge of the PBDT, with an affordable action entries definition.

For each condition outcome $\mathbf{o}^i$ in the BBDT, we can count the amount of its occurrences $O_{\mathbf{o}^i}$ in the pixel-based one. $O_{\mathbf{o}^i} = 0$ denotes a condition outcome that turns out to be impossible in practice, so we can remove the corresponding rule from the BBDT. Only 192 condition outcomes are effectively possible.

To construct the BBDT we start considering that whenever condition c_9 is not satisfied *no action* should be performed, and when $X \subset \mathcal{F}$, $X \not\bowtie P$, $X \not\bowtie Q$, $X \not\bowtie R$, and $X \not\bowtie S$ a *new label* should be created. When instead only one of $X \bowtie P$, $X \bowtie Q$, $X \bowtie R$, or $X \bowtie S$ is verified, we must perform an *assign* operation. What is important here is that this does not imply exactly the assignment of the neighboring block label to $L(X)$: we can assign $L(X)$ the label of any block directly or indirectly connected to X, i.e., the neighbor or any of its neighbors. For example, in Fig. 17, the condition outcomes 1 0001 0001, that is the case in which $X \bowtie S$ and $S \bowtie Q$ but $X \not\bowtie Q$, we can arbitrarily choose to perform the action $L(X) \leftarrow L(S)$ or $L(X) \leftarrow L(Q)$, which translates to the action entry shown.

The same approach may be applied to all other combinations, explicitly solving the connected component problem between the blocks. The labels of the connected components are then merged if any of the composing blocks is a neighbor of X. As before, the labels to be assigned or merged may be arbitrarily chosen from the any block of every connected component.

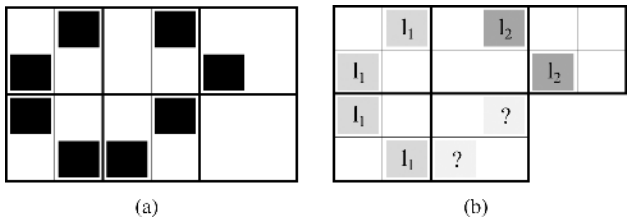


Fig. 8. Example of a complex merging situation: (a) the binary image and (b) the two sets of neighboring blocks with a common label.

An example will clarify the concept. Fig. 8(a) depicts a possible pixel configuration in which two disjoint sets of labels are connected to X. In particular $X \in \mathcal{F}$ and $X \bowtie R$ and $X \bowtie S$. Moreover $Q \bowtie R$ and $S \bowtie P$. The corresponding condition outcomes in Fig. 17 is 1 0011 0110, which leads to four possible choices for the merging

$$L(X) \leftarrow \text{merge}(\{L(P), L(Q)\})$$
$$L(X) \leftarrow \text{merge}(\{L(P), L(R)\})$$
$$L(X) \leftarrow \text{merge}(\{L(Q), L(S)\})$$
$$L(X) \leftarrow \text{merge}(\{L(R), L(S)\}).$$

These choices are obtained selecting one block from the component with label l_1 in Fig. 8(b) and the other from the component with label l_2. The output of merge will be different, but the equivalence class will be the same.

By applying these considerations to all 192 condition outcomes, the OR-decision table in Fig. 17 is obtained. In order to convert this table to a decision tree we need to produce a single entry decision table by selecting a single nonzero action entry for every rule. Since $O_{\mathbf{o}^i}$ is the probability to observe in the PBDT a pixel configuration corresponding to the condition outcome $\mathbf{o}^i$, we slightly modify the greedy technique of Section IV-C in order to directly apply it to the BBDT. The line 3 of Algorithm 1 thus becomes

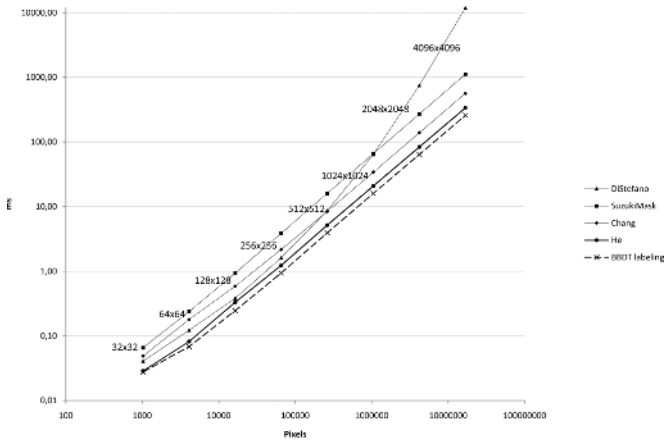
$$k^* \leftarrow \arg\max_{k \in \mathcal{I}} \sum_{i=1}^{N} e_k^i \cdot O_{\mathbf{o}^i}. \tag{13}$$

In this way, a greater importance is assigned to the actions that have a higher impact in the decision table, and are likely to provide a more effective grouping of 1s.

After the application of Algorithm 1 to the BBDT, we can produce the 65 536 rules PBDT, which contains a single action to perform given any possible pixel configuration. The Schumacher's algorithm is finally applied to this decision table, producing an optimal tree containing 210 nodes, with 211 leaves sparse over 14 levels. The code implementing the sequence of these conditions was automatically generated and an OpenCV compliant version is available online [3].

VI. Results

Connected components labeling is a well-defined problem that always yields to the same result: whatever algorithm has

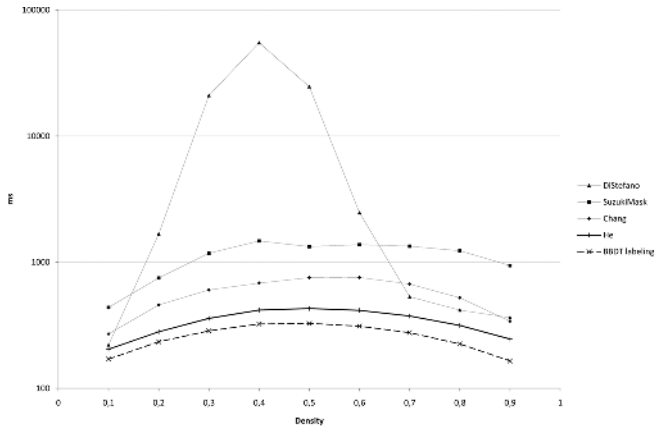


(a)

Pixels	DiStefano	SuzukiMask	Chang	He	BBDT labeling
1024	0.04	0.07	0.05	0.03	**0.03**
4096	0.12	0.24	0.18	0.08	**0.07**
16384	0.38	0.94	0.59	0.33	**0.24**
65536	1.61	3.86	2.17	1.23	**0.94**
262144	8.64	16.05	8.41	5.16	**3.93**
1048576	64.32	65.93	34.00	20.83	**15.98**
4194304	749.50	270.99	138.97	83.79	**64.08**
16777216	11817.45	1117.70	562.03	337.74	**257.39**

(b)

Fig. 9. Performance of the algorithms scaling the size of the image to label, expressed in milliseconds. The reported value is the average time obtained considering all images at all densities with that size.



(a)

Density	DiStefano	SuzukiMask	Chang	He	BBDT labeling
0.1	220.39	437.77	269.76	203.93	**170.26**
0.2	1675.87	750.24	458.26	280.65	**233.90**
0.3	20979.69	1177.14	602.74	358.43	**285.36**
0.4	55052.34	1474.81	682.86	417.17	**323.05**
0.5	24648.12	1328.63	753.90	429.66	**327.13**
0.6	2468.44	1378.83	756.97	414.91	**311.12**
0.7	531.95	1338.90	671.13	374.28	**275.98**
0.8	416.81	1232.17	522.46	315.00	**224.76**
0.9	363.46	940.79	340.20	245.65	**164.91**

(b)

Fig. 10. Performance of the algorithms varying the label densities, expressed in milliseconds. The resolution used for this chart was 4096×4096.

Fig. 11. Sample collection of random images, in this case shown at 64×64 resolution, to which a variation on the threshold is performed in order to produce different densities of labels.

to outcome the same number of labeled connected components; differences over the corresponding label values can be standardized in a second time with a common enumeration procedure. The way in which the image is scanned and the neighborhood is evaluated, and the type of data structures exploited for equivalences resolution produce a number of approaches that distinguish themselves only in terms of computational time required. In this work, we state that our proposal (block-based decision tree labeling, *BBDT* in short), provides the most efficient way to scan the images and evaluate the connectivities, and in this section we are going to show several results in different application fields.

In order to propose a valuable comparison with the state of the art, we used several large and very dissimilar datasets. We will examine the more important and effective representative of each general approach for labeling analyzed in the historical overview proposed in Section III. In particular, we suggest a comparison between the following approaches.

- Suzuki *et al.* [14] as more recent representative of the Haralick's multiscan approach, in particular with the LUT optimization proposed to speedup the process.
- Di Stefano *et al.* [13] as a straightforward Union-Find-based approach with no particular optimizations included except for the array-based data structures.
- Chang *et al.* [27] as more recent and faster representative of the contour tracing-based techniques.
- He *et al.* [19] as the more recent thus effective representative of the classical two scans approach, which has been proposed in 2008 as the fastest labeling algorithm presented so far in literature.

For each of these algorithms, the minimum time over five runs is kept in order to remove possible outliers due to other task performed by the operating system. All algorithms of course produced the same number of labels and the same labeling on all images. The tests have been performed on a Intel Core 2 Duo E6420 processor, using a single core for the processing. The code is written in C++ and compiled on Windows using Visual Studio 2008.

A. Synthetic Dataset

Analogously to many recent works [19], [18], we produced a dataset of black and white random noise square images with nine different foreground densities, from a low resolution of 32×32 pixels to a maximum resolution of 4096×4096 pixels. Unlike past works on this subject, we also generated high resolution images to prove the scalability and the effectiveness of our approach when the number of labels gets really high. For every combination of size and density, 10 images were produced for a total of 720 images. The dataset is available at [3].

The resulting dataset gives us the possibility to evaluate the performances of our approach and the other selected algorithms, both in terms of scalability on the number of pixels and in terms of scalability on the number of labels (density). An example of density variation is provided in Fig. 11.

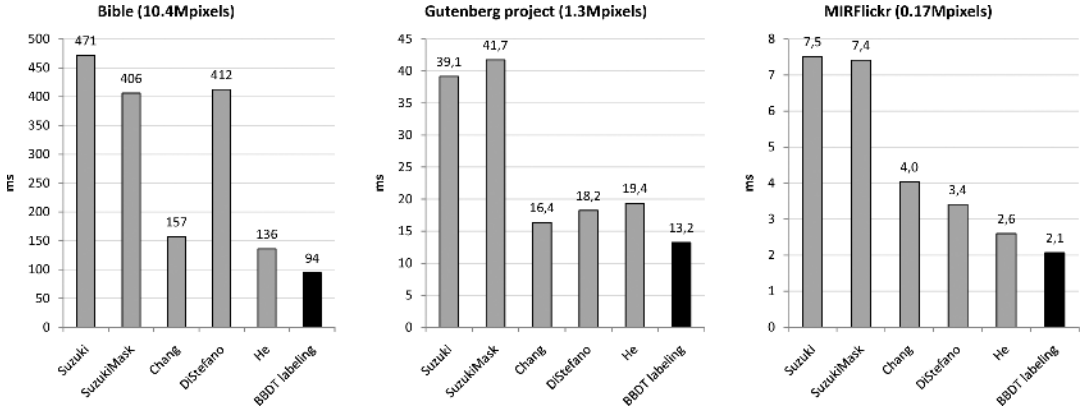
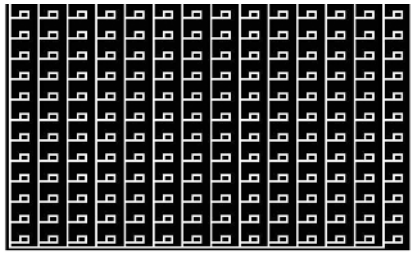


Fig. 12. Connected components labeling results on the Bible, the Gutenberg Project and MIRFlickr datasets, expressed in milliseconds.

Fig. 13. Sample collection of Otsu binarized version of the three real dataset.



(a)

	Chang	He	BBDT labeling
bigPattern	768.67	382.67	233.75
smallPattern	1115.98	495.04	294.55

(b)

Fig. 14. Pattern specifically designed to stress all the algorithms based on contour tracing technique.

Fig. 9 shows how the different algorithms behave with images with increasing sizes. The reported value is the average time obtained considering all images at all densities with that size. A linear dependency of time with respect to the number of pixels is highlighted for all algorithms except for Di Stefano's approach which is fast only when the number of pixels is relatively low. Our approach proved to be scalable and able to outperform all the others in each experiment with the increasing image size.

The second experiment proposed in Fig. 10 highlights the behavior of the algorithms varying the label densities. In this representation, the worst case is reached around the middle densities, because the number of labels and merges between equivalence classes is higher. Lower densities present more sparse labels and consequently less merges, while higher densities present highly connected components with simpler merges. Our approach evidences the best performance among all the densities. Note that Di Stefano's algorithms produces as expected the worst performance in the middle densities.

B. Real Datasets

To test the effective performance of the algorithms, we also used three datasets composed of real world images, corresponding to three possible applications of labeling (see Fig. 13).

1) Borso d'Este Holy Bible: This bible is one of the most important illuminated manuscript of the Italian Renaissance. We are involved in a project of text detection and image segmentation aimed at detecting the most valuable pictures within the bible pages, and the connected components labeling is one of the processing steps. In particular, the dataset exploited in this work is composed by the Otsu-binarized[1] versions of 615 images of high resolution (3840×2886) pages, with Gothic text, pictures, and floral decorations. This dataset gives us the possibility to test the connected components labeling capabilities with very complex patterns at different sizes, with an average resolution of 10.4 megapixels and 35 359 labels, providing a challenging dataset which heavily stresses the algorithms.

2) Gutenberg Project: This dataset is composed by 6105 high resolution scans of books taken from the Gutenberg Project [1], with an average amount of 1.3 millions of pixels to analyze and 2568 components to label. This is a typical application of document analysis and character recognition where labeling is the necessary starting step. The connected components identifies words, sub-words or characters.

3) MIRflickr: This dataset is composed by the Otsu-binarized version of the MIRflickr dataset [2], publicly available under a Creative Commons License, containing 25 000 standard resolution images taken by Flickr. These images are smaller (the average resolution is 0.17 megapixels), there are fewer connected components (495 on average) and generally less complex, so the labeling is easier to accomplish.

[1]The Otsu thresholding has been chosen only as an automatic and consistent way to produce a binarized version the image.

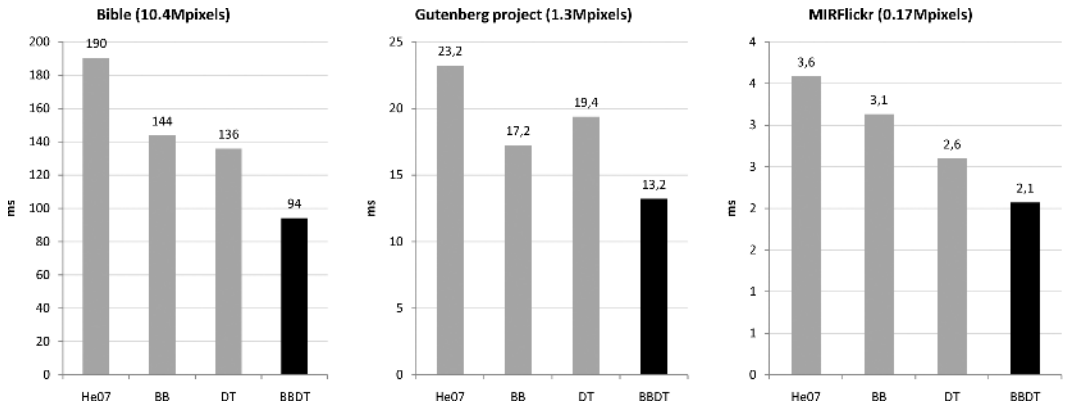


Fig. 15. Direct comparison between the two He's approaches (*He07* and *DT* in the charts) and the two main evolutions of our approach, first with only block-based optimization (*BB*), then with also the decision tree optimization (*BBDT*).

Performance tests in terms of the average time required to label an image are shown in Fig. 12. As mentioned before, we executed five runs keeping the minimum time sampled, then we compute the average of the minimum times for all images within each dataset. These tests show how our approach can outperform all the other proposals on every dataset, starting from high resolution images with thousands of labels down to standard images with few labels. The speed-up with respect to the second best algorithm is between 23% and 29%. It is also interesting to notice that, in presence of a limited number of labels, He's approach is not always the second best.

The contour tracing algorithm proposed by Chang rises up as a good competitor on each dataset. Anyway the performances of this approach can be seriously compromised with particularly elaborate patterns, whose contours are difficult to follow. An example is provided in Fig. 14. This pattern, connected in this manner, creates a very complex single connected component that forces the contour tracing to follow the entire image in the most time consuming way. We build two artificial images using this pattern, with a challenging resolution of 7000×5000; in particular, while the first image contains a pattern size suitable for the typical mask of two-scans labeling approaches, the second image used a larger one. The results show that our approach still outperforms all the others, employing respectively 233.75 and 294.55 ms to complete the labeling. Overall, in the first image our algorithm is 38.9% faster than He's algorithm and 73.6% faster than Chang's, while in the second image we perform 40.5% faster than best than He's and 71% faster than Chang's.

C. Incremental Contributions

In order to provide a deeper understanding of the relative contributions (in terms of performance improvements) of the two main novelties of this work, we also include a comparison of our final algorithm (*BBDT* in the charts) against the following approaches.

- The first He's approach (*He07*), which highlights the benefits of the Union-Find algorithm for labels resolution implemented with the set of three arrays as referred in [17].
- He's state-of-the-art approach as proposed in [18], that is the previous one with the addition of the decision tree optimization (*DT*).

Algorithm	Total accesses	Label image accesses	Binary image accesses
Chang	47.13	26.45	20.67
He	43.88	28.36	**15.52**
Our Approach	**34.04**	**17.42**	16.62

Fig. 16. Analysis of memory accesses required by the connected components computation. The numbers are given in millions of accesses.

- The block-based approach as proposed in [31] (*BB*), with the aforementioned algorithm for label resolution.

As reported in Fig. 15, we can highlight how the use of the block-based technique, applied side-by-side to the labels resolution technique in [17], guarantees competitive performances in comparison with the first He's approach itself (performance gain of 24.2% with the challenging Bible dataset). Later introducing the decision tree optimization, both He's and our approaches get a significant performance improvement. It is nevertheless interesting to notice that while the state-of-the-art approach in [18] gets a 28.4% performance boost using decision trees (DT *versus* He07), our approach (despite being far more complex in terms of tree structure) gets an higher performance boost (34.7%) over previous algorithm without decision trees (BBDT *versus* BB).

D. Memory Access Requirements

To understand the reason of the good performance of our proposal, we analyzed the memory accesses of each algorithm. In particular, we focused on a comparison with the two more representative algorithms in terms of memory access and thus speed, that is He's and Chang's approaches.

We performed these tests on the Bible dataset, and computed the average number of accesses to the label image (i.e., the image containing the provisional and then the final labels for the connected components), the average number of accesses to the binary image to be labeled and finally the sum of the two contributions. As shown in Fig. 16, the reason of the great performances of our approach is mainly due to a significantly lower number of accesses to memory. In particular, due to the optimization in the neighborhood computation and the 2×2 scanning approach, we can access much less frequently to the label image in order to extract the label of a particular block (thus group of pixels), maintaining quite as much as He's accesses

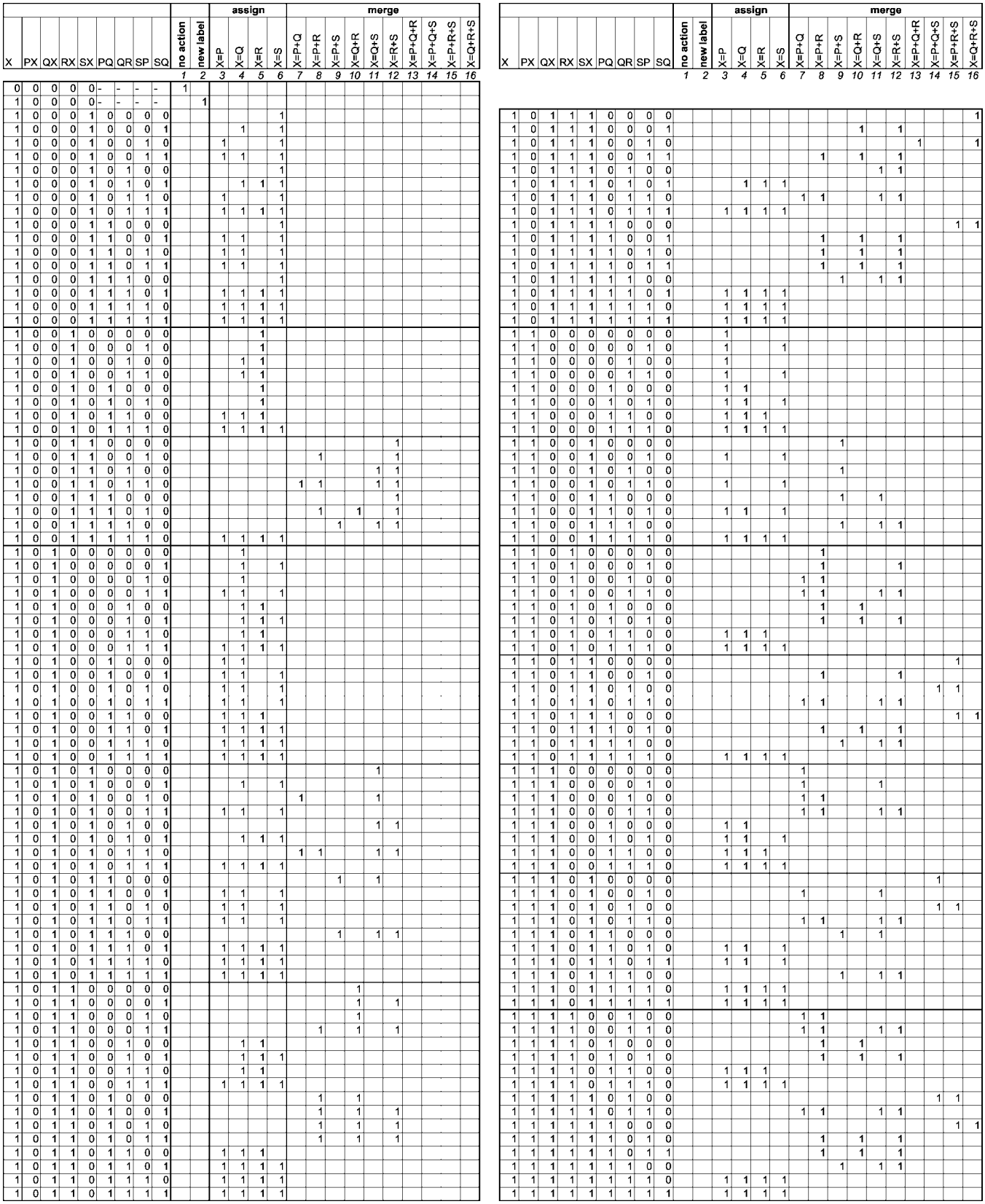


Fig. 17. Complete BBDT obtained using block connectivities. It is possible to note that we produced an OR-decision table which cannot be optimized by brute force.

to the original binary image. Globally, we can save 22% of accesses with respect to He's approach and the 27% of accesses with respect to Chang's approach.

VII. Conclusion

In this paper, we presented a novel approach to connected components analysis able to improve the performance of all ex-

isting approaches between 23% and 29% on average. Firstly, an effective modeling of the problem by means of decision tables is proposed, with the introduction of the OR-decision table to formalize the situation in which multiple alternative actions could be performed. A greedy procedure to reduce this table into a single entry decision table is proposed, and finally an automatic decision tree synthesis is implemented to obtain the optimal arrangement of conditions to verify. In order to speed up the neighborhood computation, a neighborhood scanning optimization is performed enlarging the scanning mask of pixels to 2×2 blocks. The proposed modeling methodology is particularly effective even in this case where the number of combination is very high. The experimental results evidence how our approach is faster than all other techniques proposed in literature.

REFERENCES

[1] Project Gutenberg Literary Archive Foundation, Salt Lake City, UT, "Project Gutenberg," 2010. [Online]. Available: http://www.gutenberg.org

[2] M. J. Huiskes and M. S. Lew, "The MIR flickr retrieval evaluation," presented at the ACM Int. Conf. Multimedia Inf. Retrieval (MIR), New York, 2008 [Online]. Available: http://press.liacs.nl/mirflickr/

[3] University of Modena and Reggio Emilia, Modena, Italy, "Labeling Image Lab: an impressively fast labeling routine for Open," 2010. [Online]. Available: http://imagelab.ing.unimore.it/imagelab/labeling.asp

[4] A. Rosenfeld and J. L. Pfaltz, "Sequential operations in digital picture processing," *J. ACM*, vol. 13, no. 4, pp. 471–494, 1966.

[5] Z. Galil and G. F. Italiano, "Data structures and algorithms for disjoint set union problems," *ACM Comput. Surveys*, vol. 23, no. 3, pp. 319–344, 1991.

[6] Y. Han and R. A. Wagner, "An efficient and fast parallel-connected component algorithm," *J. ACM*, vol. 37, no. 3, pp. 626–642, 1990.

[7] H. Samet, "Connected component labeling using quadtrees," *J. ACM*, vol. 28, no. 3, pp. 487–501, 1981.

[8] R. Haralick, "Some neighborhood operations," in *Real Time Parallel Computing: Image Analysis*. New York: Plenum Press, 1981, pp. 11–35.

[9] R. Lumia, L. G. Shapiro, and O. A. Zuniga, "A new connected components algorithm for virtual memory computers," *Comput. Vision, Graph., Image Process.*, vol. 22, no. 2, pp. 287–300, 1983.

[10] J. Schwartz, M. Sharjr, and A. Siegel, "An efficient algorithm for finding connected components in a binary image," New York Univ., Robotics Research Tech. Rep. 38, 1985.

[11] H. Samet and M. Tamminen, "An improved approach to connected component labeling of images," in *Proc. Int. Conf. Comput. Vision Pattern Recog.*, 1986, pp. 312–318.

[12] M. B. Dillencourt, H. Samet, and M. Tamminen, "A general approach to connected-component labeling for arbitrary image representations," *J. ACM*, vol. 39, no. 2, pp. 253–280, 1992.

[13] L. Di Stefano and A. Bulgarelli, "A simple and efficient connected components labeling algorithm," in *Proc. Int. Conf. Image Anal. Process.*, 1999, pp. 322–327.

[14] K. Suzuki, I. Horiba, and N. Sugie, "Linear-time connected-component labeling based on sequential local operations," *Comput. Vision Image Understand.*, vol. 89, pp. 1–23, 2003.

[15] K. Wu, E. Otoo, and A. Shoshani, "Optimizing connected component labeling algorithms," in *Proc. SPIE Conf. Med. Imag.*, 2005, vol. 5747, pp. 1965–1976.

[16] C. Fiorio and J. Gustedt, "Two linear time union-find strategies for image processing," *Theoretical Comput. Sci.*, vol. 154, pp. 165–181, 1996.

[17] L. He, Y. Chao, and K. Suzuki, "A linear-time two-scan labeling algorithm," in *Proc. Int. Conf. Image Process.*, 2007, vol. 5, pp. 241–244.

[18] L. He, Y. Chao, and K. Suzuki, "A run-based two-scan labeling algorithm," *IEEE Trans. Image Process.*, vol. 17, no. 5, pp. 749–756, May 2008.

[19] L. He, Y. Chao, K. Suzuki, and K. Wu, "Fast connected-component labeling," *Pattern Recog.*, vol. 42, no. 9, pp. 1977–1987, Sep. 2008.

[20] J. K. Clemens, "Optical character recognition for reading machine applications," Ph.D. dissertation, Massachusetts Inst. Technol., Cambridge, Sep. 1965.

[21] T. H. Morrin, "Chain-link compression of arbitrary black-white images," *Comput. Graph. Image Process.*, vol. 5, no. 2, pp. 172–189, 1976.

[22] R. L. T. Cederberg, "Chain-link coding and segmentation for raster scan devices," *Comput. Graph. Image Process.*, vol. 10, no. 3, pp. 224–234, 1979.

[23] B. Kruse, "A fast algorithm for segmentation of connected components in binary images," in *Proc. 1st Scandinavian Conf. Image Anal.*, Lund, Sweden, Jan. 1980.

[24] P.-E. Danielsson, "An improvement of Kruse's segmentation algorithm," *Comput. Graph. Image Process.*, vol. 17, no. 4, pp. 394–396, 1981.

[25] B. Kruse, "A fast stack-based algorithm for region extraction in binary and nonbinary images," in *Signal Process.; Theories Appl.*, M. Kunt and F. de Coulon, Eds. Amsterdam, The Netherlands: North-Holland, Jan. 1980, pp. 169–173.

[26] P.-E. Danielsson, "An improved segmentation and coding algorithm for binary and nonbinary images," *IBM J. Res. Developm.*, vol. 26, no. 6, pp. 698–707, 1982.

[27] F. Chang and C. Chen, "A component-labeling algorithm using contour tracing technique," in *Proc. Int. Conf. Document Anal. Recog.*, 2003, pp. 741–745.

[28] L. J. Schutte, "Survey of decision tables as a problem statement technique," Comput. Sci. Dept., Purdue Univ., CSD-TR 80, 1973.

[29] L. T. Reinwald and R. M. Soland, "Conversion of limited-entry decision tables to optimal computer programs i: Minimum average processing time," *J. ACM*, vol. 13, no. 3, pp. 339–358, 1966.

[30] H. Schumacher and K. C. Sevcik, "The synthetic approach to decision table conversion," *Commun. ACM*, vol. 19, no. 6, pp. 343–351, 1976.

[31] C. Grana, D. Borghesani, and R. Cucchiara, "Fast block based connected components labeling," in *Proc. IEEE Int. Conf. Image Process.*, Cairo, Egypt, Nov. 2009.

Costantino Grana (M'07) received the Ph.D. degree in information engineering from the University of Modena and Reggio Emilia, Italy, in 2004.

He is currently an Assistant Professor with the University of Modena. His research interests comprise multimedia information analysis, focusing on image and video concept detection, and historical document analysis.

Daniele Borghesani received the M.S. degree in computer science from the University of Modena and Reggio Emilia, Italy, in 2006, where he is currently pursuing the Ph.D. degree in information engineering.

His current research regards document analysis and content-based image retrieval, focused on Renaissance illuminated manuscripts.

Rita Cucchiara (M'98) received the Laurea degree in electronic engineering and the Ph.D. degree in computer engineering from the University of Bologna, Italy, in 1989 and 1993, respectively.

She is a Full Professor with the University of Modena and Reggio Emilia, Emilia, Italy, where she heads the ImageLab Laboratory. Her current research interests include pattern recognition and computer vision for video surveillance and multimedia.