A Heuristic-Based Decision Tree for Connected Components Labeling of 3D Volumes

Maximilian Söchting
Hasso Plattner Institute
Faculty of Digital Engineering
University of Potsdam, Potsdam, Germany
Email: maximilian.soechting@student.hpi.de

Stefano Allegretti, Federico Bolelli
and Costantino Grana
Department of Engineering "Enzo Ferrari"
University of Modena and Reggio Emilia, Modena, Italy
Email: {*name.surname*}@unimore.it

***Abstract*—Connected Components Labeling represents a fundamental step for many Computer Vision and Image Processing pipelines. Since the first appearance of the task in the sixties, many algorithmic solutions to optimize the computational load needed to label an image have been proposed. Among them, block-based scan approaches and decision trees revealed to be some of the most valuable strategies.**

However, due to the cost of the manual construction of optimal decision trees and the computational limitations of automatic strategies employed in the past, the application of blocks and decision trees has been restricted to small masks, and thus to 2D algorithms.

With this paper we present a novel heuristic algorithm based on decision tree learning methodology, called Entropy Partitioning Decision Tree (EPDT). It allows to compute near-optimal decision trees for large scan masks. Experimental results demonstrate that algorithms based on the generated decision trees outperform state-of-the-art competitors.

I. Introduction

Connected Components Labeling (CCL) is used in many modern Computer Vision and Image Processing tasks, whenever object recognition and/or measurement is required.

CCL has a unique and exact solution, that every algorithm must provide as output. The task aims at producing a description of the objects inside binary images, by generating a symbolic image where each pixel of a single connected component is assigned a unique identifier, typically an integer number. In this scenario, an object is defined as the set of foreground pixels such that given two of them (p_1 and p_2) it is always possible to find a path of connected pixels, belonging to the same set, that leads from p_1 to p_2.

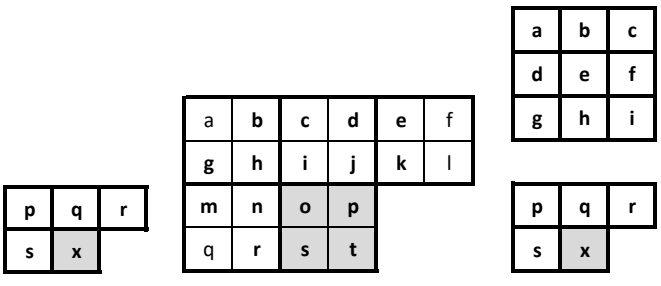
Although introduced many years ago [1], CCL is nowadays employed in several modern scenarios. Applications of such an algorithm include Object Tracking [2], Video Surveillance [3], Image Segmentation [4], [5], [6], Medical Imaging [7], [8], [9], [10], Document Restoration and Indexing [11], [12], [13], [14], and Graph Analysis [15], [16]. As a matter of fact, most state-of-the-art solutions on the aforementioned research fields exploit binary image processing algorithms as fundamental pre- or post-processing steps to get to the final results [9], [17], [18], [19] or to prepare training data [20].

The availability of efficient implementations is for this reason crucial and dozens of new algorithmic solution, both on sequential [21], [22], [23], [24], [25], [26] and parallel architectures [27], [28], [29], [30], [31], [32], [33], [34] have been published since the first proposal in 1966 [1].

Apart from the specific strategy involved, algorithms mainly differ on the total execution time required to complete the task. For this reason, an open-source framework that allows to fairly benchmark and compare new proposals on CCL has been released in 2016 [35]. The framework, named YACCLAB (Yet Another Connected Components Labeling Benchmark), has been employed by many authors since its first appearance [36], [37], [38], [39], [40], [41].

One of the cornerstones of sequential CCL algorithms has been released in [22] by Wu *et al.* with the Scan Array-based Union-Find algorithm (SAUF). The authors proved an optimal strategy, based on Decision Trees (DTrees), to reduce the average number of load/store operations during the scan of the input image. This scan was driven by the so called *Rosenfeld mask* (Fig. 1a). After the appearance of SAUF many authors proposed new solutions based on DTrees [23], [42], [43], [44], [45]. Among them, one of the major breakthroughs consists in the usage of a 2×2 block-based approach (Fig. 1b), with an automatic generation of the optimal decision tree associated to the scanning mask [42]. Strategies based on DTrees have demonstrated their effectiveness even when applied, with the necessary variations, to parallel architectures [46], [47].

Extending a 2D algorithm to 3D is just a matter of increasing the size of the scanning mask to cover also the z-direction. Unfortunately, the generation of optimal decision



(a) Rosenfeld-2D (b) Grana-2D (c) Rosenfeld-3D

Fig. 1. Example of 2D (a), (c), and 3D (c) scan masks. The 3D Rosenfeld mask contains 9 pixels in the previous slice (top) and five in the current one (bottom). Gray squares are the current pixels that needs to be labeled using the information extracted from white pixels.

trees becomes quickly unfeasible when the size of the scanning mask increases. In order to compensate the limitation of the optimal decision tree generation employed in previous works, this paper proposes a novel heuristic algorithm based on decision tree learning and named Entropy Partitioning Decision Tree (EPDT). The proposed solution allows to compute near-optimal decision trees for large scan masks, which was impossible with previous methods. The effectiveness of the proposed solution is demonstrated by applying the 2D block-based approach to a 3D scenario, and generating the corresponding decision tree.

The comparative evaluation reported in Section V demonstrates the effectiveness of our strategy. Source code of the algorithms generated through the proposed heuristic is publicly available in [48].

II. Related Work

As regards sequential solutions, paper of the last fifteen years demonstrated the superiority of *two scans* algorithms over other approaches, based on multiple scans of the image or contour tracing. A two scans CCL algorithm performs a first raster scan, in which each pixel is given a provisional label based on its neighborhood, and equivalences between labels are stored in a data structure. Then, a representative label is chosen for each connected component, and provisional labels are updated with the final ones during the second scan.

The first proposal of a two scans algorithm was presented by Rosenfeld and Pfaltz in the sixties [1]. In the first scan, the provisional label for the current pixel x is decided on the basis of the 4 already computed adjacent pixels out of the total 8, *i.e.* the three on the previous row and the one to the left of x. These pixel positions constitute the so called *Rosenfeld mask* (Fig. 1a). After this pioneering work, successive studies improved the efficiency of pixel neighborhood exploration and equivalences resolution. In particular, Samet and Tamminen [49] realized that the equivalence resolution problem can be described as a *disjoint-set union problem*, and therefore can be solved by means of the *Union-Find* algorithm [50]. The Union-Find algorithm provides two basic operations over a collection of disjoint sets of equivalent labels: *Find* retrieves the representative label of a set given an element belonging to it, and *Union* joins two sets together. When a pixel is examined during the first scan, possible new equivalences between label sets are resolved online, with a Union. Given its effectiveness and efficiency, Union-Find has been a staple in connected components labeling proposals thereafter, and authors proposed different implementations of it [22], [23].

Another substantial improvement was introduced by Wu *et al.* [22], in the form of a decision tree. They noticed that, when scanning the image using the Rosenfeld mask, the label for the current pixel can often be decided without the need to visit every neighbor in the mask. Thus, they built an optimal binary decision tree that minimizes the average number of pixel checks, by specifying a certain access order. In such a tree, each node is labeled with a pixel p of the mask, and its two children denote the next pixel to check in the two cases that p is background or foreground. Finally, each leaf contains the proper action to be performed for the current pixel x. He *et al.* [23] proposed a similar algorithm to [22] that employs a different but likewise optimal DTree, and implements the Union operation in a different way.

Then, Grana *et al.* [42] observed that foreground pixels in a 2×2 square always share the same label, and designed a block-based algorithm that works with block labels in the first scan, to reduce the amount of provisional labels and necessary Union-Find operations. The application of the Rosenfeld mask to blocks instead of pixels led to the definition of the *Grana mask* (Fig. 1b). Given the larger size of the mask, instead of building a decision tree by hand the authors generated an optimal DTree automatically, by means of a dynamic programming technique.

Another important step towards performance optimization of the first scan was represented by *state prediction*, first introduced by He *et al.* [51]. Their first scan considers two possible computations for the current pixel, for the two cases that the previous one was foreground or background. In [24], authors enhanced the concept, condensing the information about previously accessed pixels in a state, and modeled the first scan as a finite state machine. Then, Grana *et al.* [45] applied state prediction to the DTree of [22], obtaining a forest of reduced DTrees, one for each possible state. In [52], Jang *et al.* applied prediction to a block-based algorithm, using a reduced 10 pixel mask. They identified seven possible states and manually built a specific DTree for each of them.

Following a different trend, Bolelli *et al.* [25], [53] defined the concepts of equal and equivalent DTrees, and noticed that the merging of equal and equivalent subtrees in the large DTree employed in [54] could sensibly reduce the overall number of nodes, at the cost of changing the tree into a Directed Rooted Acyclic Graph, or DRAG. This optimization, known as *code compression*, improves performance reducing the code size, and therefore increasing the instruction cache hit rate. Then, in [26], authors exploited code compression to combine the whole Grana mask and state prediction.

The study of 3D connected components labeling dates back to the 1980s. When considering three dimensions, the 2D 8-connectivity becomes 26-connectivity: each voxel v is adjacent to those composing a $3 \times 3 \times 3$ cube, with v as its center. A raster scan of a volume proceeds slice-by-slice, and each slice is scanned in the same way as a single image, but considering also connections along z axis. Lumia [55] published an extension of a 2D algorithm, which first labels each slice separately, and then runs a forward and backward scan on the volume to resolve equivalences between slices. The algorithm used for equivalences resolution is not specified in the paper. The solution proposed by Thurfjell *et al.* [56] is a two scans algorithm that uses a translation table for storing label equivalences, and exploits Union-Find operations for online equivalence resolution. It considers a 3D neighborhood, which adds to the Rosenfeld mask 9 pixels of the previous slice, arranged in a 3×3 square (Fig. 1c). We will call this extended mask, composed of 14 pixels, *3D Rosenfeld*

							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. 2. *OR*-decision table for the Rosenfeld mask. All the configurations with $x = 0$ lead to the same action, and are thus condensed in a single row.

mask. More recently, He *et al.* [57] applied a DTree to the 3D Rosenfeld mask, obtaining the Label-Equivalence Based (LEB) algorithm. The mask size makes unfeasible to find the optimal tree by hand, so the authors employed a strategy consisting in prioritizing the pixels with the highest number of neighbors. The method for solving equivalences is the same variation of Union-Find used in [23]. In the same paper, the authors also proposed a run based algorithm, named Run-Based Two-Scan (RBTS), which assigns provisional labels to pixels runs, storing them in a separate data structure. LEB and RBTS represent the current state-of-the-art for 3D CCL.

III. Preliminaries

A. Decision Tables

In a two scans CCL algorithm, the operation performed on each pixel during the first scan can be described as a *command execution metaphor* [42]: the values of pixels in the mask constitute a binary word, where 1 means foreground and 0 means background. These words represent commands, and each of them is associated to a corresponding action to be performed on the current pixel or group of pixels. Possible actions are:

- $no\,action$, when the current pixel/block is background;
- $new\,label$, assign to the current pixel/block a new provisional label, and must be performed when the rest of the mask is background;
- $assign$, copy the label of a foreground neighbor to the current pixel/block;
- $merge$, merge the equivalence classes of already labeled pixels connected through the current pixel/block, which usually is assigned the minimum of merged labels.

A characteristic of the CCL problem is interchangeable actions occurrence, each time multiple foreground neighbors share equivalent labels, and therefore each of those labels can be indifferently assigned to the current pixel. Thus, commands are actually linked to sets of equivalent actions. This relation can be conveniently represented in a *OR*-decision table, as Grana *et al.* observed in [42]. The *OR*-decision table for the Rosenfeld mask is reported in Fig. 2. The left part of each row is the command, in the form of a binary word where each bit corresponds to a mask pixel, and the right part displays the equivalent actions associated to the command, marked with 1.

The *OR*-decision table is a particularly useful representation of the problem, because a DTree can be built starting from it. In particular, the dynamic programming approach described by Grana *et al.* [54] and derived from [58] takes as input the OR decision table and outputs an optimal DTree, *i.e.*, a DTree with the minimum expected path-length from root to leaves.

However, the generation of an optimal decision tree poses complexity problems as the size of the mask, and therefore the number of conditions, increases. Existing algorithms fail to generate optimal DTrees with more than 16 conditions; for this reason, no author has ever managed to apply the block-based approach to 3D algorithms.

B. Decision Tree Learning

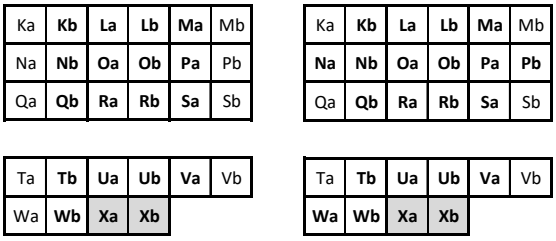
Let us consider a population of samples, each composed of an array of discrete features, and each belonging to a certain class. A decision (or classification) tree is a tree where each node is labeled with a feature, and has as many children nodes as the different values for that feature. Finally, leaves are labeled with class names. Such a tree serves to determine the class of any sample from the population: this can be done by traversing the tree from root to leaf, choosing at each node the edge corresponding to the actual feature value of the sample.

The construction of a classification tree starts from an available labeled dataset, and usually proceeds top-down, choosing for the root node the feature that best splits the dataset according to a certain metric [59]. The process is repeated recursively on every node, considering only the subset of the dataset that reached it, and stops when every sample of the subset belongs to the same class. A common metric for evaluating splits is the Information Gain (IG), employed in the ID3 tree generation algorithm [60], and in its extension C4.5 [61]. The information gain represents how much uncertainty can be reduced by splitting on a certain feature. The uncertainty of a set S is measured by its entropy $H(S)$, defined as:

$$H(S) = \sum_{c \in C} -p(c) \log_2 p(c) \tag{1}$$

In the formula, C is the set of classes in S, and $p(c)$ is the inferred probability of class c, calculated as the number of samples in class c over the total number of elements in S. Information Gain $IG(S, f)$ is defined as the difference between the entropy of S and the weighted average entropy of its subsets, determined splitting on feature f.

$$IG(S, f) = H(S) - \sum_{t \in T} p(t) H(t) \tag{2}$$



(a) 19c (b) 22c

Fig. 3. 3D scan masks based on $2 \times 1 \times 1$ blocks. These masks consider respectively (a) 19 and (b) 22 conditions, which are highlighted in bold. Both masks spread over two slices: 19c mask contains 7 pixels on the current slice (bottom) and 12 pixels in the upper one (top), while 22c mask contains 8 pixels on the current slice (bottom) and 14 pixels in the upper one (top). Gray squares identify the current pixels that need to be labeled using the information extracted from their neighbors. Non-bold pixels are ignored, as they cannot cause any connection that is not already detectable through other pixels.

Where T is the partition of S produced by the split, and $p(t)$ is the proportion of the cardinality of t to that of S. For each node, the feature yielding the largest information gain is chosen for the split. This process is a greedy heuristic, which performs a best-first search for locally optimal entropy values. In our specific case, each split will divide the original set S into two subsets T_0 and T_1, with $p(T_0) = p(T_1) = 0.5$:

$$IG(S, f) = H(S) - 0.5 \cdot H(T_0) - 0.5 \cdot H(T_1) \qquad (3)$$

IV. Outline of the Proposed Algorithm

A. 3D Block-Based Masks

As discussed in previous works [24], [42], using a block-based approach allows to reduce the number of pixel look-ups in the first scan, and the amount of label merges required to complete the task. This leads to a reduction in the number of load/store operations required, and to a consequent performance improvement. Therefore, applying a block-based approach also to the 3D space is very likely to achieve a considerable improvement.

When considering 3D volumes, it is possible to define many different block-based masks with increasing complexity. If we think about blocks of voxels, the most simple and promising options are $2 \times 1 \times 1$, $2 \times 2 \times 1$ and $2 \times 2 \times 2$ blocks.

As already mentioned, one of the core principles on which block-based CCL algorithms are based on is that all pixels/voxels inside a block are connected. This means that blocks with more that 2 pixels/voxels per edge would have no benefit.

Considering that a 3D mask would contain 14 blocks in total (9 on the previous plane, 5 on the current plane), the full masks of the aforementioned block configurations would contain $14 \times (2 \times 1 \times 1) = 28$, $14 \times (2 \times 2 \times 1) = 56$, and $14 \times (2 \times 2 \times 2) = 112$ voxels respectively. However, the idea is to slowly approach bigger masks, due to the explosion in complexity. Therefore, we start by considering masks containing only essential voxels, by simply removing those that are not direct neighbors of any x voxel. Thus, for the $2 \times 1 \times 1$ block configuration a mask with 19 conditions, hereinafter called 19c, can be generated (Fig. 3a). The mask

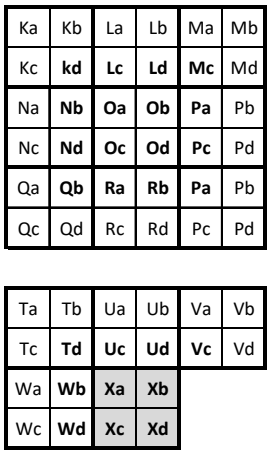


Fig. 4. 3D scan masks based on $2 \times 2 \times 1$ blocks. This mask considers 26 conditions, which are highlighted in bold. The mask spreads over two slices: 10 pixels on the current slice (bottom) and 16 pixels in the upper one (top).

based on $2 \times 1 \times 1$ blocks could be extended up to 28 pixels. Anyway, the corner voxels Ka, Mb, Qa, Sb, Ta, Vb are never directly responsible for $merges$ or $assignments$. They can thus be ignored to reduce the DTree generation complexity with no other side effects (Fig. 3b); the resulting mask is called 22c. A similar consideration can be applied to the other block configurations. As an example, the simplified version of the $2 \times 2 \times 1$ block-based mask (26c) is reported in Fig. 4.

B. Learning Decision Trees from OR-Decision Tables

With some adjustments, it is possible to apply the principles of decision tree learning to binary image processing. As discussed in Section III-A, binary image processing algorithms can be described through the command execution metaphor by means of an OR-decision table. Considering this table as a "dataset" and treating the actions as the *result classes*, it is possible to apply decision tree learning. This approach will allow the generation of a DTree that maps commands to actions, ideally checking the minimum number of conditions.

However, classifying the rule data poses a challenge: given three rules with actions $\{3\}$, $\{3, 4\}$ and $\{4\}$, which *classes* exist? Rule 1 and 2 are "equivalent", because they have a common intersection. However, rule 2 and 3 are also "equivalent", but not rule 1 and 3, *i.e.* equivalence in this case is not transitive. If all rules remaining after a *split* are of the same class, a leaf is created. When only the two rules $\{\{3\}, \{3, 4\}\}$ or $\{\{3, 4\}, \{4\}\}$ remain after a *split*, a leaf can be created. However, the rules $\{\{3\}, \{3, 4\}, \{4\}\}$ can not be reduced to a single action. Hence, taking unique action combinations as classes, *e.g.* through a bitmapped representation, would not resolve cases like $\{\{3\}, \{3, 4\}\}$ correctly and would require a large amount of memory due to the high number of classes. Therefore, a heuristic for the classification has been employed: for rules with more than one action, the action with the previous highest occurrence, *i.e.* the *temporal* "most popular" action, determines the "class" of this rule. Another heuristic can be applied as well: calculating all single action occurrences

TABLE I

ENTROPY AND INFORMATION GAIN FOR THE GENERATION OF THE SAUF TREE USING THE TEMPORAL POPULARITY CLASSIFIER. EACH ROW CORRESPONDS TO ONE NODE OF THE TREE. VALUES HAVE BEEN ROUNDED FOR VISUALIZATION REASONS. BOLD VALUES IDENTIFY THE ENTROPY AND THE IG OF THE CONDITION CHOSEN FOR THE CURRENT NODE IN THE DECISION TREE. THE FINAL RESULTING TREE IS REPORTED IN FIG. 5A.

Node	**Depth**	$H(S)$	**p**			**q**			**r**			**s**			**x**		
			$H(T_0)$	$H(T_1)$	IG	$H(T_0)$	$H(T_1)$	IG	$H(T_0)$	$H(T_1)$	IG	$H(T_0)$	$H(T_1)$	IG	$H(T_0)$	$H(T_1)$	IG
1	0	2.2	2.0	1.4	0.5	2.3	1.5	0.3	1.9	2.1	0.2	2.1	2.1	0.1	**0.0**	**2.4**	**1.0**
2	1	2.4	**2.0**	**0.8**	**1.0**	2.5	1.0	0.7	1.8	2.3	0.4	2.2	2.2	0.2			
3	2	2.0				**2.0**	**0.0**	**1.0**	1.5	1.5	0.5	1.5	1.5	0.5			
4	2	0.8				**1.0**	**0.0**	**0.3**	0.0	1.0	0.3	0.8	0.8	0.0			
5	3	2.0							**1.0**	**1.0**	**1.0**	1.0	1.0	1.0			
6	3	1.0							**0.0**	**0.0**	**1.0**	1.0	1.0	0.0			
7	4	1.0										**0.0**	**0.0**	**1.0**			
8	4	1.0										**0.0**	**0.0**	**1.0**			

in a separate pass, and always classifying based on the *global* "most popular" action. If there is a tie in popularity, the action with the lower id is taken in both classifiers. The global classifier has been found to be the most effective at obtaining high-quality decision trees, close to the optimal decision trees. The temporal popularity classifier performs slightly worse, but it is much faster since it does not require an additional pass for counting global rules. Using the temporal classifier, the first two rules of $\{\{3\},\{3,4\},\{4\}\}$ would be therefore classified in class 3, and the third rule in class 4, resulting in an entropy of 0.92 ($P_3 = \frac{2}{3}$, $P_4 = \frac{1}{3}$).

In order to demonstrate the application of decision tree learning to CCL and to binary image processing problems in general, a small example based on the Rosenfeld mask is discussed in the following of this Section. The example is based on IG and *temporal* popularity classifier, and the step-by-step execution is reported in Table I.

The Rosenfeld mask (Fig. 1a) has five conditions, and thus 32 different mask configurations (Fig. 2). For the first node, there are 5 conditions on which the split can be performed: p, q, r, s and x. Splitting on these conditions results in the information gain values displayed in the first row of Table I. Since x has the highest information gain, a condition node with x is created, and two child nodes are added. Because all rules with $x = 0$ result in action 1 ($no\,action$), the entropy is zero and the left child node is a leaf. On the other hand, rules with $x = 1$ do not have a common intersection and the entropy is greater than zero. This results in a non-leaf node. The procedure is repeated in the child nodes until all leaves are created (Fig. 5a). As can be noticed from the comparison with the optimal DTree (Fig. 5b) obtained using the algorithm presented in [54], in this specific case the heuristic-generated tree contains only one extra node.

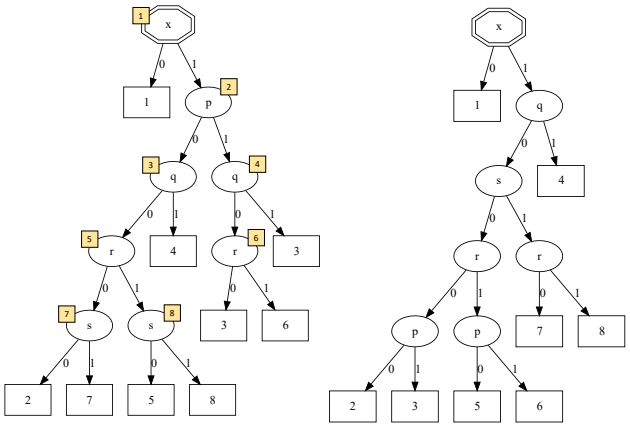
V. EXPERIMENTAL RESULTS

Algorithms generated with the proposed strategy are evaluated using YACCLAB [35], [48], [62], a widely used [39], [41] open source *C++* benchmarking framework for CCL algorithms. Experimental results discussed in the following are obtained on an Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz with Windows 10.0.17134 (64 bit) OS and MSVC 19.15.26730 compiler. All the algorithms have been compiled for x64 architecture with optimizations enabled.

Exploiting the DTree learning strategy presented in Section III-B, we generated three different algorithms: EPDT_19c, EPDT_22c, and EPDT_26c. They exploit a two scan approach based on the $2 \times 1 \times 1$ (19c, 22c) and the $2 \times 2 \times 1$ (26c) block-based masks. Obtained DTree have also been compressed as described in [25] to minimize code footprint. The proposed heuristic solutions have been compared to state-of-the-art algorithms for 3D CCL: Label-Equivalence-Based (LEB) and Run-Based Two-Scan (RBTS), by He *et al.* [57].

Three different 3D datasets are included in YACCLAB, and have been used to carry out the following comparison evaluation. One is synthetically generated, and the other two come from the medical domain, which is the main applicationfield for 3D CCL.

Hilbert contains six volumes of $128\times128\times128$ pixels, filled with the 3D Hilbert curve obtained at different iterations (1 to



(a) EPDT (b) Optimal DTree

Fig. 5. (a) decision tree generated using *IG* and temporal popularity classifier for the rules given in Fig. 2. The entropy values used for the creation of this tree are reported in Table I, with yellow numbers indicating the respective node numbers. (b) one of the optimal decision trees that can be generated with the algorithm described in [42] starting from the decision table of Fig. 2. Ellipsis represent the conditions to be checked, and rectangles (leaves) contain the actions to be performed, which are identified by integer numbers. (1) $x \leftarrow no\,action$, (2) $x \leftarrow new\,label$, (3) $x \leftarrow p$, (4) $x \leftarrow q$, (5) $x \leftarrow r$, (6) $x \leftarrow s$, (7) $x \leftarrow p + r$, (8) $x \leftarrow r + s$.

6) of the construction method. The Hilbert curve is a fractal space-filling curve, and it represents a challenging test case for the CCL task. *Mitochondria* is the Electron Microscopy Dataset, which contains binary sections taken from the CA1 hippocampus [63]. Volumes are composed by 165 slices, with a resolution of 1024×768 pixels. *OASIS* is a dataset of medical MRI data, taken from the Open Access Series of Imaging Studies (OASIS) project [64]. It consists of 373 volumes of $256 \times 256 \times 128$ pixels, binarized with the Otsu threshold.

Table II reports average execution times of the considered algorithms over the three datasets. In order to discover which label equivalence solving method works best for the EPDT algorithms, four different options have been considered: standard Union-Find (UF), Union-Find with Path Compression (UFPC) [22], Three Table Array (TTA) [23], and interleaved Rem algorithm with SPlicing (RemSP) [65]. For what concerns LEB and RBTS, instead, the TTA solver is always employed since it delivers the best results according to [57].

In order to achieve a deeper understanding of execution time distribution, two more tests have been performed. Only one label solver has been considered for EPDT algorithms: RemSP, which, according to Table II, on average carries out the best performance. Bar charts of Fig. 6 report average execution time split between the three steps composing the algorithms: memory allocation and deallocation, first scan, and second scan. Finally, Table III compares the amount of average memory accesses on the binary image, labels image and equivalence vector of pixel- and block-based algorithms. RBTS, having a considerably different memory access pattern due to its run-based nature, has been excluded.

The comparison of raw execution times in Table II demonstrates that the best EPDT algorithm is 22c. It also outperforms LEB, the current state-of-the-art, on Hilbert and Mitochondria, with an average speed-up of $1.25\times$ and $1.15\times$ respectively, while obtaining the same performance on OASIS.

As can be noticed, EPDT_19c is better than EPDT_26c, but worse than EPDT_22c in all test cases. In order to explain this performance gap, several factors must be taken into account. Among them, the most significant are the number of load/store operation, the amount of $merge$ required, and code size.

EPDT_22c considers all the meaningful pixels of the $2 \times 1 \times 1$ block-based mask, adding also the pixels ignored by EPDT_19c. Including missing pixels and generating the corresponding DTree allows to reduce the total number of load/store operations required by the algorithm from $\sim1\%$ to $\sim3\%$, depending on the considered dataset. Indeed, using the information provided by additional pixels, it is possible to identify equivalence classes earlier during the first scan, without having to perform expensive merge operations later.

Nevertheless, expanding the size of the mask has a cost: the generated tree code lines increase from $\sim4k$ for EPDT_19c to $\sim7k$ for EPDT_22c, negatively impacting the instruction cache. However, the benefits of reducing merges and in general load/store operations outclass the disadvantages of the increased cache miss rate.

On the other hand, when comparing EPDT_19c and EPDT_26c we face a totally different scenario: the reduction in load/store operations is much more significant ($\sim10\% \div \sim19\%$ depending on the dataset), but at the same time the tree size increases up to more than $41k$ lines of code. In this case the cache miss rate prevails, causing an overall performance degradation. The described effects are magnified by any further increase in the mask size. For this reason, we limit our analysis to masks up to 26 conditions.

Comparing EPDT_22c with LEB, conclusions similar to those above can be drawn. LEB is based on a hand-crafted decision tree that guarantees the minimum number of accesses to the input image, while neglecting what concerns merges and accesses to the output image. Taking the OASIS dataset as a reference (Table III), LEB requires about $1.4x$ the load/store operations of EPDT_22c, on average. On the other hand, the small tree employed by LEB completely fits instruction cache: EPDT_22c and LEB have comparable performance on OASIS dataset. As the number and the complexity of connected components grow (Mitochondria and Hilbert), the gap in the number of load/store operations and the improvement of EPDT_22c over LEB become more evident.

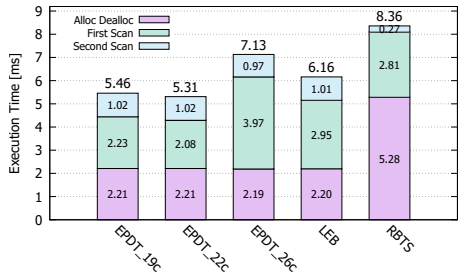
Differentiating time spent in actual algorithm execution from time used for memory allocation (Fig. 6), it is possible to draw further interesting conclusions. EPDT_19c, EPDT_22c, EPDT_26c, and LEB make use of the same data structures, taking the same time to allocate and deallocate memory (negligible oscillations are due to measurement accuracy). In contrast, RBTS requires additional data structures to keep track

TABLE II
AVERAGE RESULTS IN MS. LOWER IS BETTER.

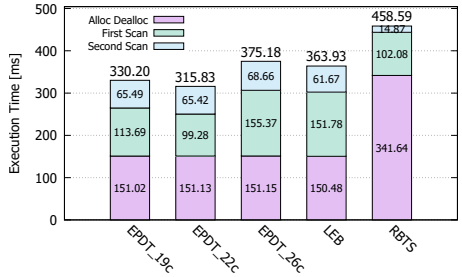
	Hilbert	**Mitochondria**	**OASIS**
EPDT_19c_RemSP	4.77	276.72	30.12
EPDT_19c_TTA	5.07	281.07	30.54
EPDT_19c_UF	4.78	277.66	30.38
EPDT_19c_UFPC	4.81	277.63	30.57
EPDT_22c_RemSP	**4.68**	276.48	29.07
EPDT_22c_TTA	4.84	276.66	**29.00**
EPDT_22c_UF	4.73	276.79	29.16
EPDT_22c_UFPC	4.69	**271.62**	29.36
EPDT_26c_RemSP	7.96	398.39	51.22
EPDT_26c_TTA	8.15	398.30	51.45
EPDT_26c_UF	8.02	400.45	51.61
EPDT_26c_UFPC	7.93	399.57	51.62
LEB_TTA	5.84	313.63	29.03
RBTS_TTA	8.24	430.46	45.50

TABLE III
AVERAGE NUMBER OF LOAD/STORE OPERATIONS ON THE *OASIS* DATASET. QUANTITIES ARE GIVEN IN MILLIONS.

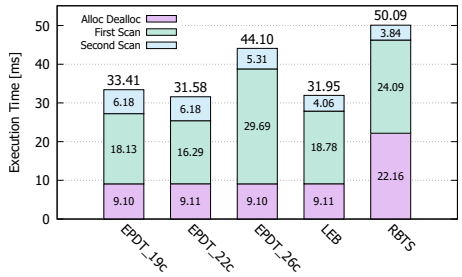
Algorithm	**Binary Image**	**Labels Image**	**Equivalences Vector**	**Total**
LEB	11.461	27.182	9.851	48.494
EPDT_19c	14.917	17.760	1.169	33.846
EPDT_22c	14.057	17.753	1.145	32.955
EPDT_26c	13.695	13.145	0.728	27.568



(a) Hilbert



(b) Mitochondria



(c) OASIS

Fig. 6. Average run-time tests with steps in ms (lower is better).

of run information, thus spending about twice the time in memory allocation. On the other hand, the additional information stored by RBTS allows to significantly reduce the number of merge operations in the first scan, and to early identify final labels, avoiding to replace provisional ones during the second scan. Obviously, datasets with few connected components and few patterns that cause merge occurrence (OASIS) practically nullify the advantages of runs.

However, the benefits introduced by runs are not sufficient to compensate for the allocation cost overhead. For this reason, RBTS is the worst in all considered settings. In very specific context, where it is realistic to perform memory allocation only once, like an embedded system which capture fixed size images, RBTS would be one of the best performing solution.

VI. Conclusion

With this paper we introduced a generic heuristic approach for generating decision trees starting from an OR-decision table. The resulting approach, Entropy Partitioning Decision Tree, EPDT in short, has been proven to generate near-optimal decision trees when applied to connected components labeling. Through the proposed heuristic it was possible to extend the block-based approach also to 3D volumes, significantly improving the state-of-the-art on the field. Source-code of the EPDT-based algorithms is publicly available in [48].

References

[1] A. Rosenfeld and J. L. Pfaltz, "Sequential Operations in Digital Picture Processing," *Journal of the ACM*, vol. 13, no. 4, pp. 471–494, Oct. 1966.

[2] A. Dubois and F. Charpillet, "Tracking Mobile Objects with Several Kinects using HMMs and Component Labelling," in *Workshop Assistance and Service Robotics in a human environment, International Conference on Intelligent Robots and Systems*, 2012, pp. 7–13.

[3] C. Zhan, X. Duan, S. Xu, Z. Song, and M. Luo, "An Improved Moving Object Detection Algorithm Based on Frame Difference and Edge Detection," in *Fourth International Conference on Image and Graphics (ICIG 2007)*. IEEE, 2007, pp. 519–523.

[4] A. Abramov, T. Kulvicius, F. Wörgötter, and B. Dellen, "Real-Time Image Segmentation on a GPU," in *Facing the multicore-challenge*. Springer, 2010, pp. 131–142.

[5] F. Pollastri, F. Bolelli, R. Paredes, and C. Grana, "Improving Skin Lesion Segmentation with Generative Adversarial Networks," in *2018 IEEE 31st International Symposium on Computer-Based Medical Systems (CBMS)*. IEEE, 2018, pp. 442–443.

[6] A. Körbes, G. B. Vitor, R. de Alencar Lotufo, and J. V. Ferreira, "Advances on Watershed Processing on GPU Architecture," in *International Symposium on Mathematical Morphology and Its Applications to Signal and Image Processing*. Springer, 2011, pp. 260–271.

[7] A. Eklund, P. Dufort, M. Villani, and S. LaConte, "BROCCOLI: Software for fast fMRI analysis on many-core CPUs and GPUs," *Frontiers in neuroinformatics*, vol. 8, p. 24, 2014.

[8] H. V. Pham, B. Bhaduri, K. Tangella, C. Best-Popescu, and G. Popescu, "Real time blood testing using quantitative phase imaging," *PloS one*, vol. 8, no. 2, p. e55676, 2013.

[9] F. Pollastri, F. Bolelli, R. Paredes, and C. Grana, "Augmenting data with GANs to segment melanomaskin lesions," in *Multimedia Tools and Applications*. Springer, 2019, p. 15575–15592.

[10] L. Canalini, F. Pollastri, F. Bolelli, M. Cancilla, S. Allegretti, and C. Grana, "Skin Lesion Segmentation Ensemble with Diverse Training Strategies," in *Computer Analysis of Images and Patterns*. Springer, September 2019, pp. 89–101.

[11] T. Lelore and F. Bouchara, "FAIR: A Fast Algorithm for Document Image Restoration," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 8, pp. 2039–2048, 2013.

[12] F. Bolelli, "Indexing of Historical Document Images: Ad Hoc Dewarping Technique for Handwritten Text," in *Italian Research Conference on Digital Libraries*. Springer, 2017, pp. 45–55.

[13] F. Bolelli, G. Borghi, and C. Grana, "Historical Handwritten Text Images Word Spotting through Sliding Window HOG Features," in *Image Analysis and Processing - ICIAP 2017*. Springer, 2017, pp. 729–738.

[14] ——, "XDOCS: an Application to Index Historical Documents," in *Digital Libraries and Multimedia Archives*. Springer, 2018, pp. 151–162.

[15] T. Berka, "The Generalized Feed-forward Loop Motif: Definition, Detection and Statistical Significance," *Procedia Computer Science*, vol. 11, pp. 75–87, 2012.

[16] M. J. Dinneen, M. Khosravani, and A. Probert, "Using OpenCL for Implementing Simple Parallel Graph Algorithms," in *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*, 2011.

[17] T. Falk, D. Mai, R. Bensch, Ö. Çiçek, A. Abdulkadir, Y. Marrakchi, A. Böhm, J. Deubner, Z. Jäckel, K. Seiwald *et al.*, "U-Net: deep learning for cell counting, detection, and morphometry," *Nature Methods*, vol. 16, no. 1, p. 67, 2019.

[18] I. H. Laradji, N. Rostamzadeh, P. O. Pinheiro, D. Vazquez, and M. Schmidt, "Where are the Blobs: Counting by Localization with Point Supervision," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 547–562.

[19] G. Máttyus, W. Luo, and R. Urtasun, "DeepRoadMapper: Extracting Road Topology from Aerial Images," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 3438–3446.

[20] W. Chen and J. Hays, "SketchyGAN: Towards Diverse and Realistic Sketch to Image Synthesis," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 9416–9425.

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

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

[23] L. He, Y. Chao, and K. Suzuki, "A Linear-Time Two-Scan Labeling Algorithm," in *International Conference on Image Processing*, vol. 5, 2007, pp. 241–244.

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

[25] F. Bolelli, L. Baraldi, M. Cancilla, and C. Grana, "Connected Components Labeling on DRAGs," in *International Conference on Pattern Recognition*, 2018, pp. 121–126.

[26] F. Bolelli, S. Allegretti, L. Baraldi, and C. Grana, "Spaghetti Labeling: Directed Acyclic Graphs for Block-Based Connected Components Labeling," *IEEE Transactions on Image Processing*, vol. 29, pp. 1999–2012, 2019.

[27] F. Spagnolo, F. Frustaci, S. Perri, and P. Corsonello, "An Efficient Connected Component Labeling Architecture for Embedded Systems," *Journal of Low Power Electronics and Applications*, vol. 8, no. 1, p. 7, 2018.

[28] S. Allegretti, F. Bolelli, M. Cancilla, and C. Grana, "Optimizing GPU-Based Connected Components Labeling Algorithms," in *2018 IEEE International Conference on Image Processing, Applications and Systems (IPAS)*. IEEE, 2018, pp. 175–180.

[29] C. Zhao, W. Gao, and F. Nie, "A Memory-Efficient Hardware Architecture for Connected Component Labeling in Embedded System," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 30, no. 9, pp. 3238–3252, 2020.

[30] S. Perri, F. Spagnolo, and P. Corsonello, "A Parallel Connected Component Labeling Architecture for Heterogeneous Systems-on-Chip," *Electronics*, vol. 9, no. 2, p. 292, 2020.

[31] S. Zavalishin, I. Safonov, Y. Bekhtin, and I. Kurilin, "Block Equivalence Algorithm for Labeling 2D and 3D Images on GPU," *Electronic Imaging*, vol. 2016, no. 2, pp. 1–7, 2016.

[32] S. Allegretti, F. Bolelli, M. Cancilla, and C. Grana, "A Block-Based Union-Find Algorithm to Label Connected Components on GPUs," in *Image Analysis and Processing - ICIAP 2019*. Springer, September 2019, pp. 271–281.

[33] Y. Komura, "GPU-based cluster-labeling algorithm without the use of conventional iteration: Application to the Swendsen–Wang multi-cluster spin flip algorithm," *Computer Physics Communications*, vol. 194, pp. 54–58, 2015.

[34] S. Allegretti, F. Bolelli, and C. Grana, "Optimized Block-Based Algorithms to Label Connected Components on GPUs," *IEEE Transactions on Parallel and Distributed Systems*, pp. 423–438, August 2019.

[35] F. Bolelli, M. Cancilla, L. Baraldi, and C. Grana, "Toward reliable experiments on the performance of Connected Components Labeling algorithms," *Journal of Real-Time Image Processing*, pp. 229–244, 2018.

[36] D. Zhang, H. Ma, and L. Pan, "A Gamma-signal-regulated Connected Components Labeling Algorithm," *Pattern Recognition*, vol. 91, pp. 281–290, 2019.

[37] T. Chabardès, P. Dokládal, and M. Bilodeau, "A labeling algorithm based on a forest of decision trees," *Journal of Real-Time Image Processing*, pp. 1527–1545, 2019.

[38] T. Chabardes, P. Dokládal, M. Faessel, and M. Bilodeau, "A parallel, O(N) algorithm for unbiased, thin watershed," in *2016 IEEE International Conference on Image Processing*. IEEE, 2016, pp. 2569–2573.

[39] J. Chen, K. Nonaka, H. Sankoh, R. Watanabe, H. Sabirin, and S. Naito, "Efficient Parallel Connected Component Labeling with a Coarse-to-fine Strategy," *IEEE Access*, vol. 6, pp. 55 731–55 740, 2018.

[40] A. Hennequin, L. Lacassagne, L. Cabaret, and Q. Meunier, "A new Direct Connected Component Labeling and Analysis Algorithms for GPUs," in *2018 Conference on Design and Architectures for Signal and Image Processing (DASIP)*. IEEE, 2018, pp. 76–81.

[41] D. Ma, S. Liu, and Q. Liao, "Run-Based Connected Components Labeling Using Double-Row Scan," in *International Conference on Image and Graphics*. Springer, 2017, pp. 264–274.

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

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

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

[45] C. Grana, L. Baraldi, and F. Bolelli, "Optimized Connected Components Labeling with Pixel Prediction," in *Advanced Concepts for Intelligent Vision Systems*. Springer, 2016, pp. 431–440.

[46] F. Bolelli, M. Cancilla, and C. Grana, "Two More Strategies to Speed Up Connected Components Labeling Algorithms," in *International Conference on Image Analysis and Processing*. Springer, 2017, pp. 48–58.

[47] S. Allegretti, F. Bolelli, M. Cancilla, F. Pollastri, L. Canalini, and C. Grana, "How does Connected Components Labeling with Decision Trees perform on GPUs?" in *18th International Conference on Computer Analysis of Images and Patterns*. Springer, 2019, pp. 39–51.

[48] YACCLAB Benchmark. Accessed on 2019-05-21. [Online]. Available: https://github.com/prittt/YACCLAB

[49] H. Samet and M. Tamminen, "An improved approach to connected component labeling of images," in *International Conference on Computer Vision And Pattern Recognition*, vol. 318, 1986, p. 312.

[50] B. A. Galler and M. J. Fisher, "An improved equivalence algorithm," *Commun. ACM*, vol. 7, no. 5, pp. 301–303, May 1964.

[51] L. He, Y. Chao, and K. Suzuki, "An efficient first-scan method for label-equivalence-based labeling algorithms," *Pattern Recognition Letters*, vol. 31, no. 1, pp. 28–35, 2010.

[52] Y. Jang, J. Mun, K. Oh, and J. Kim, "Block-Based Connected Component Labeling Algorithm with Block Prediction," in *2017 40th International Conference on Telecommunications and Signal Processing (TSP)*. IEEE, 2017, pp. 578–581.

[53] F. Bolelli, M. Cancilla, L. Baraldi, and C. Grana, "Connected Components Labeling on DRAGs: Implementation and Reproducibility Notes," in *Reproducible Research in Pattern Recognition*. Springer, 2019, pp. 89–93.

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

[55] R. Lumia, "A New Three-Dimensional Connected Components Algorithm," *Computer Vision, Graphics, and Image Processing*, vol. 23, no. 2, pp. 207 – 217, 1983.

[56] L. Thurfjell, E. Bengtsson, and B. Nordin, "A New Three-Dimensional Connected Components Labeling Algorithm with Simultaneous Object Feature Extraction Capability," *CVGIP: Graphical Models and Image Processing*, vol. 54, no. 4, pp. 357–364, 1992.

[57] L. He, Y. Chao, and K. Suzuki, "Two Efficient Label-Equivalence-Based Connected-Component Labeling Algorithms for 3-D Binary Images," *IEEE Transactions on Image Processing*, vol. 20, no. 8, pp. 2122–2134, 2011.

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

[59] L. Rokach and O. Maimon, "Top-Down Induction of Decision Trees Classifiers - A Survey," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 35, no. 4, pp. 476–487, 2005.

[60] J. R. Quinlan, "Induction of Decision Trees," *Machine learning*, vol. 1, no. 1, pp. 81–106, 1986.

[61] ——, *C4.5: Programs for Machine Learning*. Elsevier, 2014.

[62] C. Grana, F. Bolelli, L. Baraldi, and R. Vezzani, "YACCLAB - Yet Another Connected Components Labeling Benchmark," in *2016 23rd International Conference on Pattern Recognition (ICPR)*. IEEE, 2016, pp. 3109–3114.

[63] A. Lucchi, Y. Li, and P. Fua, "Learning for Structured Prediction Using Approximate Subgradient Descent with Working Sets," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2013, pp. 1987–1994.

[64] D. S. Marcus, A. F. Fotenos, J. G. Csernansky, J. C. Morris, and R. L. Buckner, "Open Access Series of Imaging Studies (OASIS): Longitudinal MRI Data in Nondemented and Demented Older Adults," *J. Cognitive Neurosci.*, vol. 22, no. 12, pp. 2677–2684, 2010.

[65] E. W. Dijkstra, *A discipline of programming*. Prentice-Hall Englewood Cliffs, N.J, 1976.