Chapter 2

Theoretical Background

In this chapter, it is presented a brief theoretical background about graph theory, basic image processing concepts and mathematical morphology necessary to explain the tools, algorithms and applications that will be developed in the subsequent chapters.

2.1 Graph Theory Basic Definitions

Definition 2.1 ***Undirected Graph:*** *An undirected graph can be seen as a set of nodes, V, and a set of edges, E, represented by non-ordered pairs of nodes. Usually a graph G is denoted by $G(V, E)$.*

Definition 2.2 ***Path:*** *A path between nodes i and j is a sequence of edges connecting these two nodes.*

Definition 2.3 ***Simple Path:*** *A path with no repeated nodes is called a simple path.*

Definition 2.4 ***Cycle:*** *A cycle is defined as being a path that starts and ends at the same node.*

Definition 2.5 ***Connected Component:*** *A connected component of an undirected graph is a maximal set of nodes in which any two nodes are connected to each other by paths.*

Definition 2.6 ***Connected Graph:*** *A graph is said connected if there is a path from any node to any other node in the graph.*

Definition 2.7 ***Tree:*** *A tree is an undirected graph G that is connected and has no cycles.*

An undirected graph G with $V = \{0, 1, 2, 3, 4, 5\}$ and $E = \{(1, 2), (1, 3), (2, 3), (3, 4), (3, 5)\}$ is illustrated in Figure 2.1. The graph is not connected, since node 0 is not connected to anyone, and the path joining nodes 1, 2, and 3 form a cycle. A tree is illustrated in Figure 2.2.

Definition 2.8 ***Rooted Tree:*** *A rooted tree, $\mathcal{T}$, is a tree in which one of its nodes has been designated the root, therefore its edges have a natural orientation, i.e. it is a directed graph. It has a partial ordering of its nodes. We say that $i \leq j$, if and only if the path from the root to i passes through j.*

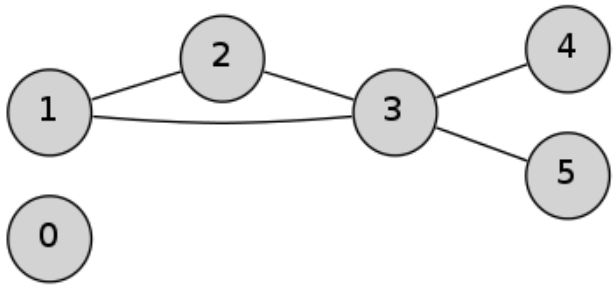


Figure 2.1: Illustration of a graph G with $V = \{0, 1, 2, 3, 4, 5\}$ and $E = \{(1, 2), (1, 3), (2, 3), (3, 4), (3, 5)\}$.

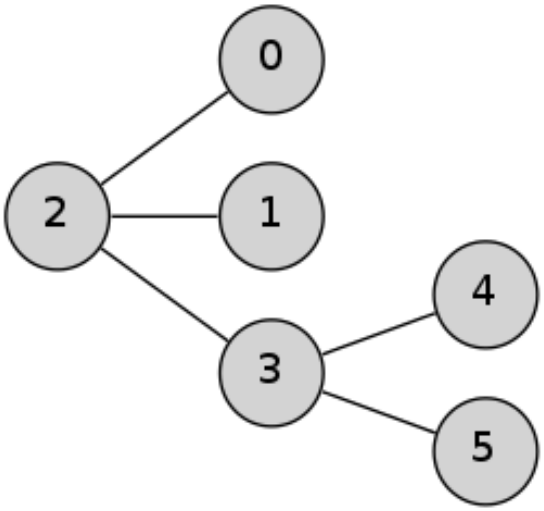


Figure 2.2: Illustration of a tree.

Definition 2.9 ***Parent:*** *The parent of a node is the node connected to it on the path to the root.*

Definition 2.10 ***Child:*** *A child of a node i is a node of which i is the parent.*

Definition 2.11 ***Root:*** *It is the only node in the rooted tree that does not have a parent.*

Definition 2.12 ***Leaf:*** *It is a node in the tree that does not have children.*

Definition 2.13 ***Bifurcation:*** *It is a node with more than one child.*

Definition 2.14 ***Internal:*** *It is any node that is neither a leaf or the root of the tree.*

Definition 2.15 ***Sibling:*** *Two nodes are said to be siblings if they have the same parent.*

Definition 2.16 ***Descendant:*** *A descendant node i of a node j is a node such that j is in the path from i to the root of the tree.*

Definition 2.17 ***Ancestor:*** *An ancestor node of i is any node in the path from i to the root of the tree.*

Definition 2.18 ***Branch:*** *A branch is the set of nodes that join a leaf to the root node.*

A rooted tree is illustrated in Figure 2.3. The arrows point towards the parents. Node 0 is the root. Nodes 6 and 7 are leaves, nodes 3 and 4 are siblings. Node 2 is a ramification. The sequence $0 - 1 - 2 - 3 - 5 - 6$ constitutes a branch of the tree.

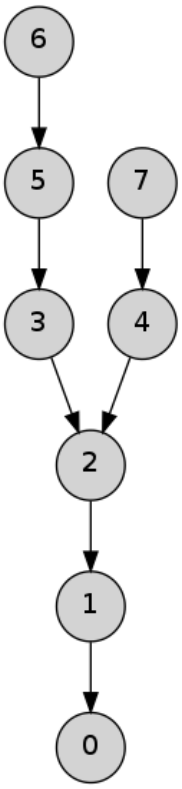


Figure 2.3: Illustration of a rooted tree. The arrows point towards the parents.

2.2 Image Processing Definitions

Definition 2.19 ***Gray-scale image:*** *It is a function* $I(z) : E \to k$, $E \in \mathcal{N}^2$ *and* $k \in \mathcal{Z}$. p *is an ordered pair* (z_{lin}, z_{col}) *that represents a pixel of the image, and* $I(z)$ *is its luminosity intensity, usually* $I(z) \in [0, L-1]$, $L > 1$. *If* $L = 2$, *the image is said to be a binary image, the pixels with intensity equal* 1 *compose the foreground and the pixels with intensity equal* 0 *compose the background of the binary image.*

Definition 2.20 ***Negative of an image:***

$$Neg(I)(z) = (L-1) - I(z), \quad \forall z \in E \tag{2.1}$$

Definition 2.21 ***Image Thresholding:*** *The image thresholding is a procedure that given a threshold* h, *it transforms an image into a binary image. There are two cases the upper and the lower thresholding. They are expressed, respectively by the following equations:*

$$\mathcal{X}_h^{\geq}(z) = \begin{cases} 1 & \text{if } I(z) \geq h \\ 0 & \text{otherwise} \end{cases}, \tag{2.2}$$

$$\mathcal{X}_h^{\leq}(z) = \begin{cases} 1 & \text{if } I(z) \leq h \\ 0 & \text{otherwise} \end{cases}. \tag{2.3}$$

Definition 2.22 ***Neighborhood of a pixel:*** *The neighborhood of a pixel* z *corresponds to a set of coordinates* $C(z) \subset E$ *defined in relation to* z *in the image domain* E. *The most common pixel neighborhoods used are the neighborhoods* $C4$ *and* $C8$ *defined below:*

$$C4(z) = \{z + (-1, 0), z + (0, -1), z + (0, 1), z + (1, 0)\} \cap E \tag{2.4}$$

(a) $C4$ (b) $C8$

Figure 2.4: Illustration of the neighborhoods (a) $C4$ and (b) $C8$ of a pixel. The blue pixels are the neighbors of the red pixels.

$$C8(z) = C4(z) \cup \{z + (-1,-1), z + (-1,1), z + (1,-1), z + (1,1)\} \cap E \tag{2.5}$$

The neighborhoods $C4$ and $C8$ of a pixel are illustrated in Figure 2.4.

Definition 2.23 ***Connectivity:*** *An image can be seen as a graph, where the pixels correspond to the nodes. Two pixels are connected, i.e. there is an edge joining them, if they are neighbors, according to the neighborhood used, that share a common property that defines a component. The property may be color, brightness, range of brightness values, or anything else of interest.*

Definition 2.24 ***Adjacency:*** *Two pixels z_1 and z_2 are adjacent, if they are connected.*

Definition 2.25 ***Upper threshold set:*** *The upper threshold set is the set of connected components $\{C_{h,1}, C_{h,2}, ..., C_{h,n_{CC}}\}$ that compose the foreground of the binary image resulting of the upper threshold $\mathcal{X}_h^{\geq}(I)$. The similarity criterion to define the connectivity is that the pixels have the same gray-level.*

An important property of level sets is that for each connected component $C_{h+1,m}$ resulting of the upper threshold $\mathcal{X}_{h+1}^{\geq}(I)$ there is a connected component $C_{h,n}$ resulting of the upper threshold $\mathcal{X}_h^{\geq}(I)$, such that $C_{h+1,m} \subseteq C_{h,n}$.

Definition 2.26 ***Flat zone:*** *Flat zones are connected components in which the similarity criteria used to define connectivity is that the pixels have the same gray-level.*

Definition 2.27 ***Regional Maximum:*** *A regional maximum of an image I is a flat zone M, if $I(z) > I(q)$, $\forall z \in M$ and q is any neighboring pixel of M.*

Definition 2.28 ***Partition*** *A partition $\mathcal{P}$ of a set X is a set of nonempty subsets of X such that every element x in X is in exactly one of these subsets.*

$$\mathcal{P}(X) = \{X_0, X_1, ...X_{n-1}\}, \quad X_i \cap X_j = \emptyset, \quad for \quad i \neq j \tag{2.6}$$

$$X = X_0 \cup X_1 ... \cup X_{n-1} \tag{2.7}$$

Definition 2.29 ***Gray-scale Image Ordering:*** *A gray-scale image f is said to be contained in a gray-scale image I of the same dimensions, $f \leq I$, if and only if:*

$$f[z] \leq I[z], \quad \forall z. \tag{2.8}$$

Definition 2.30 ***Anti-extensive Filter:*** *An anti-extensive filter ψ is a filter such that:*

$$\psi(I) \leq I, \quad \forall I. \tag{2.9}$$

Definition 2.31 ***Extensive Filter:*** *An extensive filter ψ is a filter such that:*

$$I \leq \psi(I), \quad \forall I. \tag{2.10}$$

Definition 2.32 ***Connected Filter:*** *is a filter in which the partition composed of the input image flat zones is always finer than the partition of the filtered image flat zones. The mathematical definition of a connected filter ψ is given by:*

$$\mathcal{P}_I \subseteq \mathcal{P}_{\psi(I)}, \quad \forall I. \tag{2.11}$$

Connected filters do not create new contours in the image and they do not modify the position of the existing contours.

Definition 2.33 ***Image Region:*** *A region Q is a set of pixels, where each pixel has at least one neighbor pixel that belongs to the region.*

Definition 2.34 ***Boundary of a Region:*** *The boundary ∂Q_i of a region Q_i is the set of pixels that are neighbors to at least one element of the region, but do not belong to the region.*

Definition 2.35 ***Extremal Region:*** *A extremal region $Q \subset E$ is a region such that either for all $z \in Q$, $q \in \partial Q$: $I(z) > I(q)$ (maximum intensity region) or for all $z \in Q$, $q \in \partial Q$: $I(z) < I(q)$ (minimum intensity region).*

Definition 2.36 ***Maximally Stable Extremal Region (MSER):*** *Let Q_1,..,Q_{i-1},Q_i be a sequence of nested extremal regions, $Q_i \subset Q_{i+1}$. Extremal region Q_j is maximally stable if and only if j is a minimum of the following expression:*

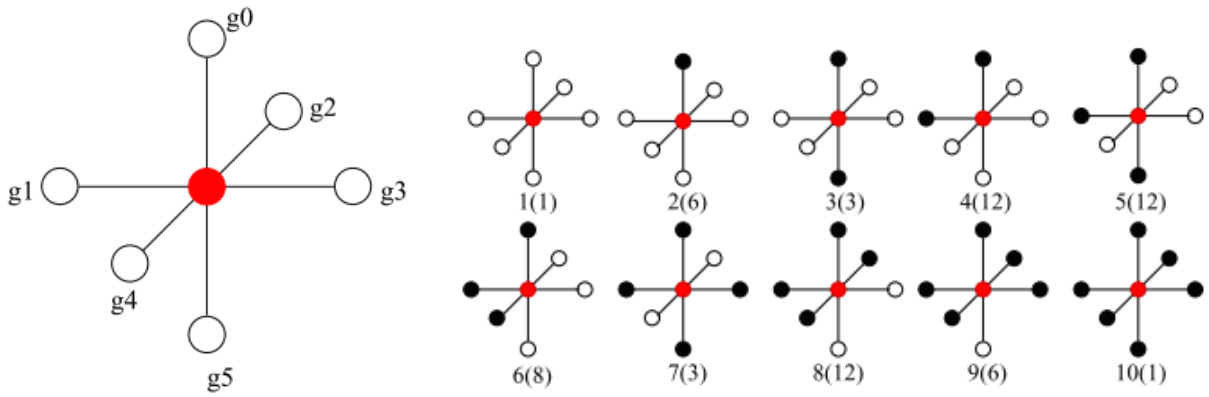
$$q(j) = \frac{\mid Q_{j+\Delta} \setminus Q_{j-\Delta} \mid}{\mid Q_j \mid}. \tag{2.12}$$

MSER regions corresponding to a sequence of upper thresholds and lower thresholds are denominated $MSER+$ and $MSER-$, respectively. According to Matas et al. (2002), MSER regions are invariant to affine intensity transformations, they preserve the adjacency when submitted to continuous transformations, and they perform multi-scale detection, since the detected regions may be either large or small.

Definition 2.37 ***Structural Similarity index (SSIM) :*** *The SSIM index proposed by (Wang et al., 2004) measures the structural similarity between images. It is defined by the following expression:*

$$SSIM(x, y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)}, \tag{2.13}$$

where x and y are images of the same dimensions μ_x, μ_y, σ_x^2, σ_y^2, and σ_{xy} represent the means, the variances and the covariance of images x and y. C_1 and C_2 are constants and their default values are 0.01 *and* 0.03*, respectively.*



(a) (b)

Figure 2.5: (a) LBP with 6 neighbors. (b) LBP patterns.

Definition 2.38 ***3D Local Binary Pattern*** *Local Binary Patterns (LBPs)(He and Wang, 1990) are texture descriptors that describe the texture surrounding a pixel.*

LBPs were initially designed to be used in 2D images, but many extensions to 3D were proposed. An obvious extension of LBPs to 3D using the 6-neighborhood of a voxel c (Figure 2.5(a)) is given by the following equation:

$$LBP(c) = \sum_{i=0}^{5} [f(g_i) > f(c)]2^i, \tag{2.14}$$

where f is the function that returns the gray-level intensity of that voxel in the image. This extension results in a code with $2^6 = 64$ pattern possibilities. Montagne et al. (2013) proposed a 3D LBP extension which is rotation invariant and has only 10 groups. The main idea consists in analyzing the 6-neighborhood of a voxel c (Figure 2.5(a)) and counting the number of voxels g_i that have a higher gray-level intensity than c. According to this count and the distribution of those voxels the patterns are clustered (Figure 2.5(b), Table 2.1).

Table 2.1: 3D LBP groups.

Group	$\sum(f(g_i) > f(c))$	**Pattern**
1	0	
2	1	
3	2	opposite voxels
4	2	bend voxels
5	3	voxels on the same plane
6	3	voxels on different planes
7	4	voxels on the same plane
8	4	voxels on different planes
9	5	
10	6	

2.3 Mathematical Morphology

Definition 2.39 ***Gray-scale Erosion****: Denoting an image by $I(z)$ and the gray-scale struc-*

turing element by $b(z)$. The gray-scale erosion of I by b is given by:

$$(I \ominus b)(z) = \inf_{y \in b}[I(z+y) - b(y)], \tag{2.15}$$

where inf *is the infimum operator.*

Definition 2.40 ***Gray-scale Dilation****: The gray-scale dilation of I by b is given by:*

$$(I \oplus b)(z) = \sup_{y \in b}[I(y) - b(z-y)], \tag{2.16}$$

where sup *is the supremum operator.*

Definition 2.41 ***Conditional Dilation****: The gray-scale conditional dilation of I by D conditioned by g is given by:*

$$(I \oplus_g D)(z) = (I \oplus D) \wedge g, \tag{2.17}$$

where $\wedge$ is the point-wise minimum operator.

Definition 2.42 ***Morphological Reconstruction****: The gray-scale morphological reconstruction of g from the marker I is given by:*

$$(g \triangle_D I)(z) = (I \oplus_g D)^{\infty}, \tag{2.18}$$

where ∞ represents repeating the conditional dilation until stability.

Definition 2.43 ***Watershed transform from minima****: The watershed transform from minima is a segmentation technique, in which it sees the image as a topographical relief. Its basic idea consists of placing water sources of different colors in each regional minimum in the relief. The relief is flooded using these sources, and barriers are built when different water sources meet (Figure 2.6). Each segmentation region defined by a regional minimum is called catchment basin (CB).*

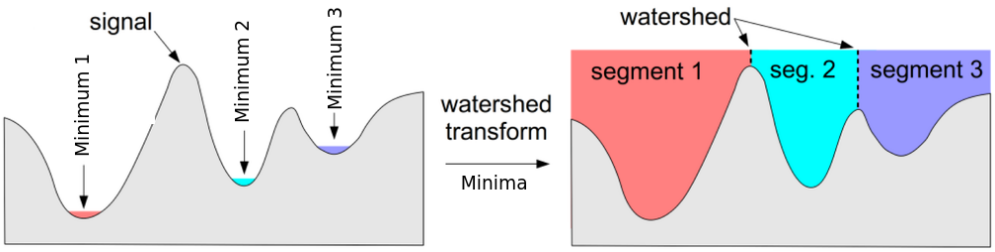


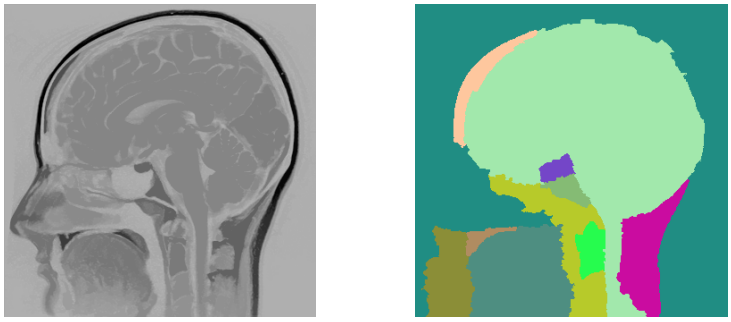
Figure 2.6: Watershed from minima operation.

Usually, the watershed transform is applied either to the negative of the image, the gradient of the image or the distance transform of the image. The major problem of the watershed from minima is that images usually have thousands of minima, leading to an over-segmentation problem. This problem is illustrated in Figure 2.7. There, we apply the watershed from minima to the negative of a sagittal slice of a brain MR image (MRI) (Figure 2.7(b)), and the segmentation result (Figure 2.7(c)) completely fails to segment any relevant structure.

(a) Brain MRI slice (b) Negative (c) Watershed from minima

Figure 2.7: Illustration of the watershed from minima applied directly to the negative of a brain MRI slice.

In order to avoid the over-segmentation problem, the images are usually pre-processed using a flooding filter that reduces the number of minima in the image. The most common flooding filter used is the hmax (Salembier et al., 1998), which is a contrast filter. The pre-flooding followed by the watershed from minima methodology is illustrated in Figure 2.8. In this example, the negative of the original image was pre-flooded by 30% of its maximum gray-level intensity. We see that the segmentation result is visually better, since we can identify the brain in the segmentation result.



(a) Pre-flooded negative image (b) Watershed from minima

Figure 2.8: Illustration of the watershed from minima applied to the negative of a brain MRI slice pre-flooded by 30% of its maximum gray-level intensity.

Definition 2.44 ***Watershed transform from markers****: The watershed from markers has a similar operation as the watershed from minima, the difference is that the process starts from imposed markers in the image, that do not necessarily have to be minima.*

This way the number of regions in the segmentation output is given by the number of imposed markers. The watershed from markers operation is illustrated in Figure 2.9. The excellence of the segmentation results are dependent on the quality of the markers chosen.

The watershed from markers applied to the brain MR image depicted in Figure 2.7 is illustrated in Figure 2.10. Two markers are used in this case, one in the brain and one in the scalp.

The watershed transform can be seen as an optimization problem. (Audigier and Lotufo, 2005; Audigier et al., 2005) showed that the watershed transform may have multiple solutions, and its output may depend on algorithm implementation details, such as the order the image

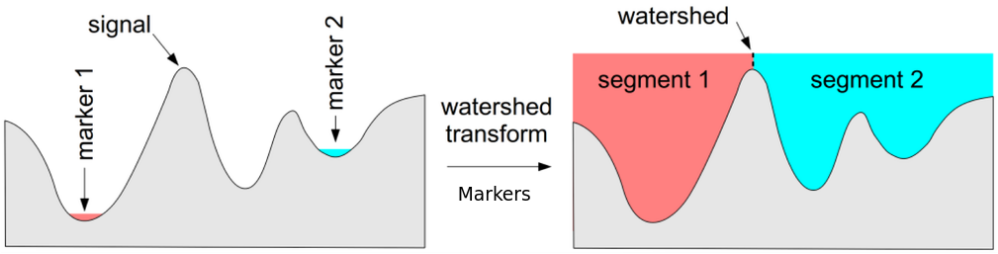


Figure 2.9: Watershed from markers operation.

(a) Markers (b) Watershed from markers

Figure 2.10: Illustration of the watershed from markers.

pixels or pixels neighbors are processed. For instance, the watershed result applied to an image can be different of the watershed result applied to the same image rotated by 90° , which is an undesirable feature. Figure 2.11 illustrates a simple tie-zone example, where we obtain two different watershed results depending on the image scanning order.

The tie-zone influence can be practically negligible in some cases (1 pixel bias, Figure 2.11), but in other cases it can have a considerable influence on the segmentation result.

Definition 2.45 ***Tie-zone watershed*** *The tie-zone watershed assigns a tie-zone label to the regions that have the same cost to more than one CB, therefore the tie-zones regions may be addressed in a post-processing step.*

(a) Image (b) Output 1 (c) Output 2

Figure 2.11: Illustration of the tie-zone problem. (a) Numeric 1D image. (b)-(c) Possible segmentation results. A and B are the segmentation labels.

The tie-zone watershed result using the markers in Figure 2.10(a) is depicted in Figure 2.12. The black region corresponds to the tie-zone, i.e. a region that could be either assigned as background or foreground using the usual watershed implementations. The area of the brain region in this example is of 43294 pixels, while the tie-zone has 2084 pixels, therefore the brain segmentation region size may vary up to 6.3% in this case.

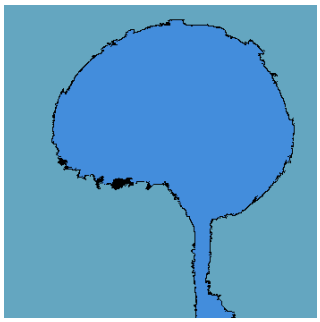


Figure 2.12: Illustration of the tie-zone watershed from markers. The tie-zone is represented in black.

2.4 Chapter Conclusions

This chapter presented the minimal theoretical background necessary for the development of subsequent chapters. More detailed and formal definitions can be found in (Dougherty and Lotufo, 2003) and (Gonzalez and Woods, 2006).

Chapter 3

Max-tree

In this chapter, we define the max-tree, its signature analysis, the procedure of building a max-tree of a max-tree that leads to the space of shapes (Xu et al., 2012), and extinction values (Vachier, 1995). These concepts have already been defined in the literature, so the first portion of this chapter corresponds to a review. The second portion of this chapter proposes and presents a mathematical definition of extinction filters using increasing attributes and a methodology to compute extinction filters using non-increasing attributes. These new definitions are the main contribution of this chapter. Extinction filters can be efficiently computed on the max-tree and are the basis of the applications developed in the next chapters.

3.1 Max-tree

Max-tree is a data structure that represents a gray scale image as a tree based on the hierarchical property of threshold decomposition. It was proposed by (Salembier et al., 1998) as an efficient structure to implement anti-extensive, and extensive by duality, connected filters. The max-tree processing pipeline has three steps: max-tree construction, tree filtering and image reconstruction. This pipeline is depicted in Figure 3.1. The max-tree filtering and image reconstruction processing times are usually negligible compared to the construction time, therefore the max-tree is even more efficient when performing a succession of filtering steps, such as the ones used to construct extinction profiles (Ghamisi et al., 2016b).

3.2 Max-tree Representation

Suppose that the gray-levels of an image I are bounded between I_{min} and I_{max}. The upper threshold sets $\{C_{h,1}, C_{h,2}, ..., C_{h,n_{CC}}\}$ resulting of every upper threshold $\mathcal{X}_h^{\geq}(I)$ for h varying between I_{min} and I_{max} can be computed, and we know that threshold sets have the following property:

$$\mathcal{X}_{h+1}(I) \subseteq \mathcal{X}_h(I), \tag{3.1}$$

and therefore:

$$\forall C_{h+1,m}, \quad \exists C_{h,n} : C_{h+1,m} \subseteq C_{h,n}. \tag{3.2}$$

This hierarchy property allows a tree representation. The max-tree, $\mathcal{T}_{MT}$, is a rooted tree in which each node stores a set of attributes. It is completely defined by its set of nodes

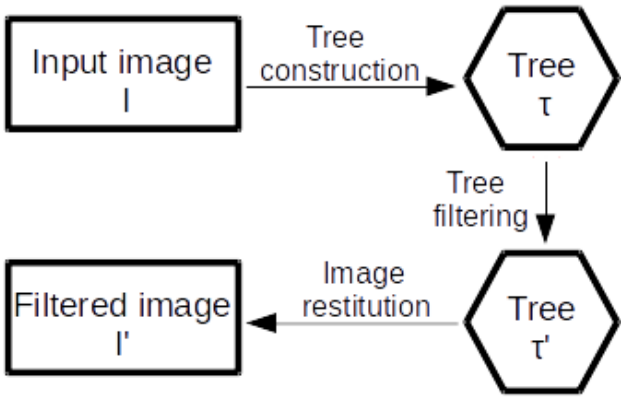


Figure 3.1: Max-tree filtering fluxogram.

$\mathcal{N}(\mathcal{T}_{MT})$:

$$\mathcal{N}(\mathcal{T}_{MT}) = \{0, 1, .., m' - 1\}, \tag{3.3}$$

and parents $\mathcal{P}(\mathcal{T}_{MT})$:

$$\mathcal{P}(\mathcal{T}_{MT}) = \{p_i | i \in (\mathcal{N} \setminus 0)\}, \tag{3.4}$$

where by convention 0 is the root. $\widehat{C_{h,n}}$ is defined by:

$$\widehat{C_{h,n}} = \{z \in C_{h,n} | I(z) = h\}. \tag{3.5}$$

It represents a set of flat zones. For each non-empty $\widehat{C_{h,n}}$, there is a max-tree node i with $h_i = h$ and $\widehat{C}_i = \widehat{C_{h,n}}$. The pixels represented by $\widehat{C}_i$ and their level h_i are the minimum set of attributes that a node i has to store in order to be able to recover the image from the max-tree. Note that storing $\widehat{C}_i$ is much more memory efficient than storing C_i, there is no redundancy in the pixels stored. The connected component C_i can be recovered by:

$$C_i = \widehat{C}_i \cup \bigcup_{\forall j \in D(i)} \widehat{C_j}, \tag{3.6}$$

where $D(i)$ is the set containing all the descendants of i. Node i is a parent of node j if C_i is the smallest connected component that contains C_j.

The number of connected components, $nlevels$, that a max-tree node i represents is given by:

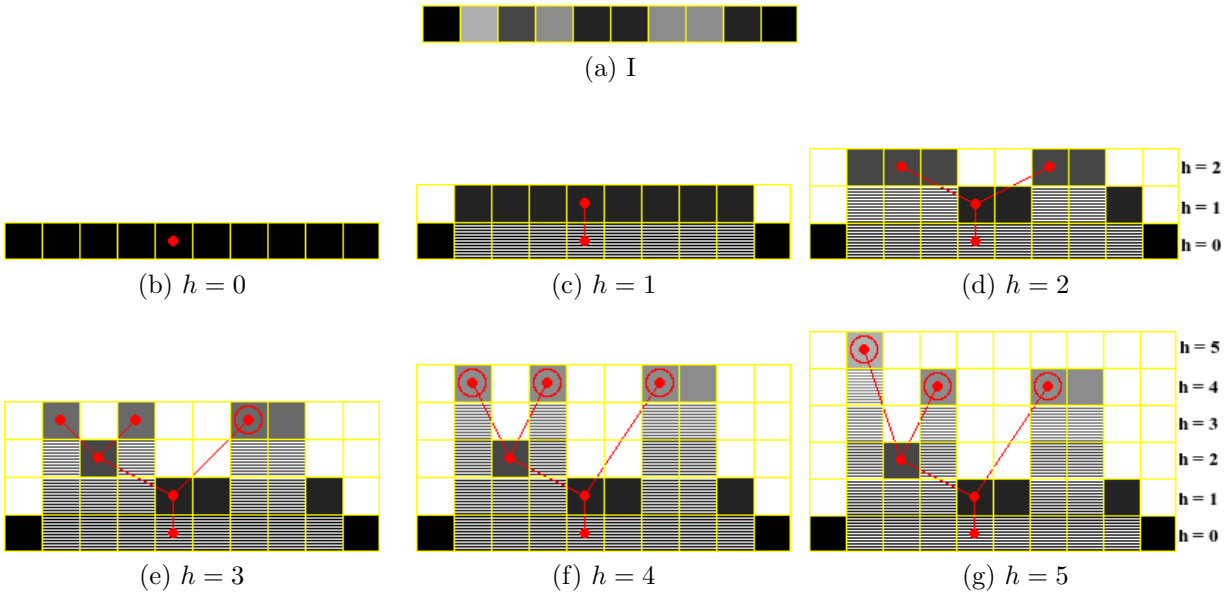
$$nlevels(i) = h_i - h_{p_i}. \tag{3.7}$$

If $nlevels(i) > 1$, this node is a **composite node**, i.e. a node that represents a connected component that remained unchanged for a sequence of threshold values. This notion of composite node is useful to implement the max-tree based algorithms.

The restitution of the image corresponding to a given max-tree, is an inverse mapping of the pixels stored in the nodes to the image coordinates system, and is expressed by the following equation:

$$I(z) = \{h_i | z \in \widehat{C}_i\}. \tag{3.8}$$

The construction of the max-tree corresponding to the 1D image $I = [0, 5, 2, 4, 1, 1, 4, 4, 1, 0]$ is illustrated in Figure 3.2. The pixels stored in each node is obtained by the set



(a) I

(b) $h = 0$ (c) $h = 1$ (d) $h = 2$

(e) $h = 3$ (f) $h = 4$ (g) $h = 5$

Figure 3.2: Max-tree construction of the 1D image $I = [0, 5, 2, 4, 1, 1, 4, 4, 1, 0]$ (a). Bottom-up construction (b)-(g). Resulting max-tree (g).

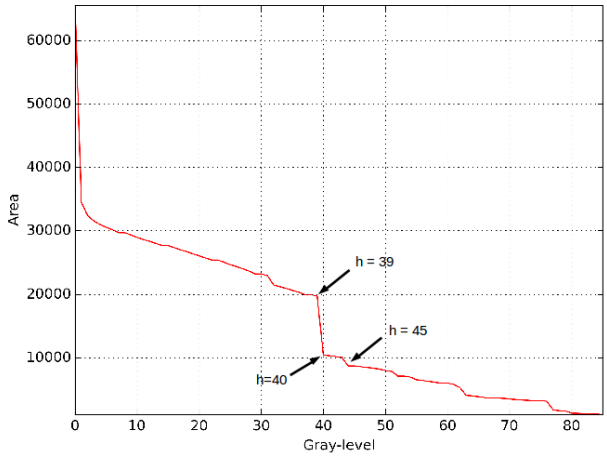
difference between the pixels obtained at level h minus the pixels at level $h + 1$, and the empty nodes are removed from the final representation. This illustration is meant just to explain the max-tree structure and construction, it does not correspond to the most efficient form to build it, for that, there are algorithms that run in quasi-linear time (Salembier et al., 1998; Carlinet and Géraud, 2014). The composite nodes are represented by double circles. The filled pixels in each connected component represent the pixels each max-tree node stores ($\widehat{C}_i$), and its union with the dashed pixels in the same connected component (C_i) is the result of applying Equation 3.6 to this node.

3.3 Max-tree Signature Analysis

The max-tree signature consists in analyzing the variation of an attribute of any pair of nodes i and j connected by a max-tree path. It conveys information concerning the variation in shape or size of a connected component. The attribute signature uses the linking information between connected components at sequential gray-levels in the image to help the decision making process.

The attribute signature may be used to choose a threshold value to segment an object in an image. One sagittal slice of a brain MR is depicted in Figure 3.3(a). The red dot in the image corresponds to a regional maxima, i.e. a max-tree leaf. The area is the number of pixels that compose a connected component. The area signature starting at this leaf and ending at the root of the tree are shown in Figure 3.3(b). The area signature presents some discontinuities. The greatest discontinuity happens in the transition between $h = 39$ to $h = 40$. At $h = 39$, the area of the connected component is of 19710, in the following gray-level, $h = 40$, the area drops to 10458, which represents the disconnection between the brain (with a portion of the esophagus still attached to it) and the skull. The next discontinuity splits the brain from the esophagus. The reconstruction of the connected components at levels 39, 40 and 45 are depicted in Figure 3.3(c)-(d).

(a) Sagittal slice



(b) Area signature

(c) Reconstruction at $h = 39$

(d) Reconstruction at $h = 40$

(e) Reconstruction at $h = 45$

Figure 3.3: (a) Sagittal brain MR slice. (b) Area signature. (c) Reconstruction of the connected component at level 39, (d) 40, and (e) 45.

3.4 Extinction Values

Extinction values are a measure of persistence of extrema (minima or maxima) proposed by (Vachier, 1995). The measure of persistence is related to an attribute, which when initially defined by Vachier had to be increasing. In the max-tree framework, an increasing attribute is an attribute that for any node i of the tree the attribute value of p_i is greater or equal to the attribute of i. Extinction values of the height attribute are also known as dynamics (Grimaud, 1992). Extinction values can be formally defined. Consider M a regional maximum of an image I, and $\Psi = (\psi_\lambda)_\lambda$ is a family of decreasing connected anti-extensive transformations. The extinction value corresponding to M with respect to Ψ and denoted by $\varepsilon_\Psi(M)$ is the maximal λ value, such that M still is a regional maxima of $\psi_\lambda(I)$. This definition can be expressed through the following equation:

$$\varepsilon_\Psi(M) = \sup\{\lambda \geq 0 | \forall \mu \leq \lambda, M \subset Max(\psi_\mu(I))\} \tag{3.9}$$

Extinction values of minima can be defined similarly. The height minima and maxima extinction values of a 1D signal are illustrated in Figure 3.4. It is important to emphasize that extinction values are not only related to the amplitude of the peak, but they also depend of the adjacent extrema. For instance, the signal shown in Figure 3.4 has more than one hundred regional maxima, but only six of them are relevant.

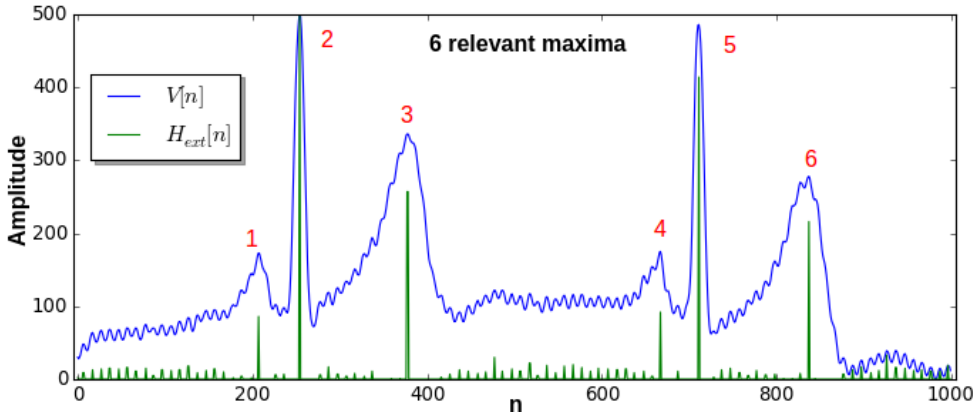
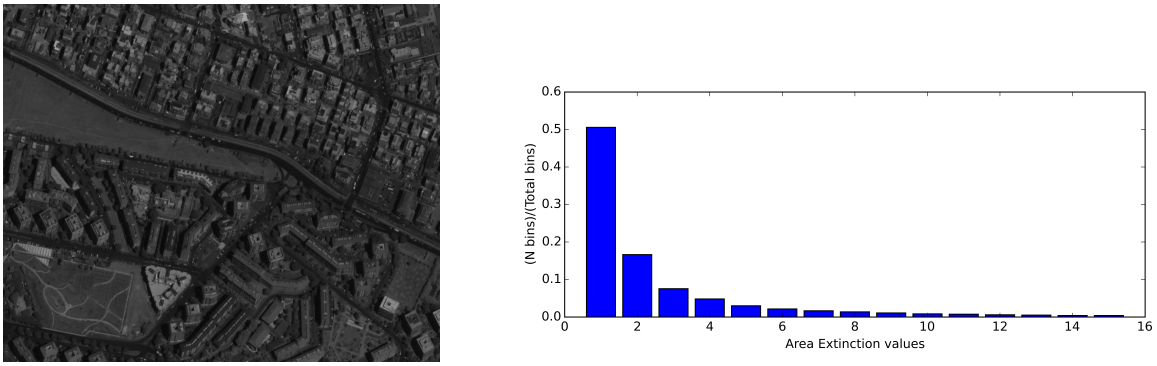


Figure 3.4: Extinction values of a 1D signal. This signal has many regional maxima, but only six of them are relevant.

Extinction values can be efficiently computed from the max-tree. There are algorithms with linear complexity to compute them (Bertrand, 2007; Silva and Lotufo, 2008).

3.5 Extinction Filter

Natural (real) images are contaminated by noise, therefore they contain many irrelevant extrema, *i.e.* extrema with low extinction values. For example, a satellite high resolution panchromatic image of an urban area of the city of Rome, Italy acquired by the QuickBird satellite is depicted in Figure 3.5(a). The image is 972×1188 pixels and has 67960 regional maxima. More than 50% of the maxima has an area extinction value of one (Figure 3.5(b)), therefore if we apply an area-open (Vincent, 1994) filter set to filter structures smaller or equal to one, more than 50% of the image maxima would be filtered.



(a) (b)

Figure 3.5: (a) Rome satellite image and (b) its area normalized extinction histogram.

Extinction filters (EFs) can be efficiently implemented using the max-tree structure (Souza et al., 2015b). The general description of the EF operation on the max-tree is the following:

1. Build the image max-tree if filtering maxima (anti-extensive) or min-tree if filtering minima (extensive).

2. Compute the leaves extinction values of the increasing attribute being analyzed.

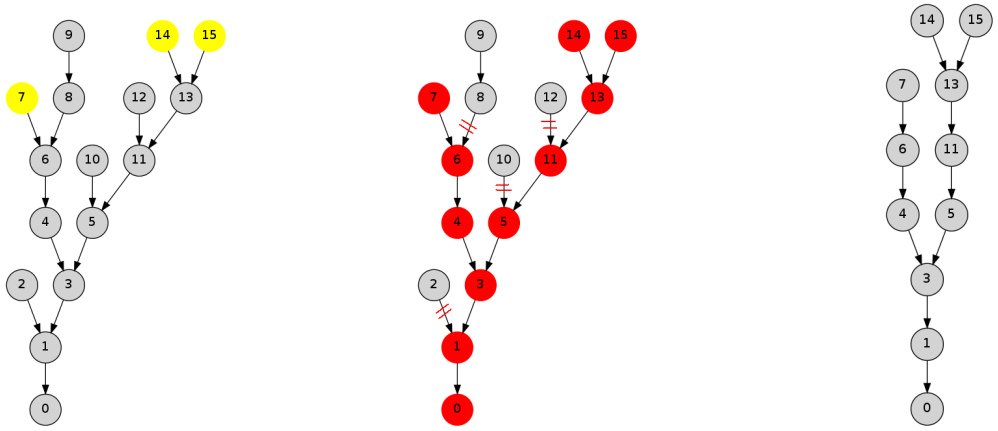


Figure 3.6: (a) Original max-tree, the yellow nodes are the three nodes with highest extinction values. (b) Nodes in the path from the three leaves with highest extinction values to the root are marked in red. (c) The result of the pruning of the nodes not marked in red.

3. Mark all nodes on the paths starting from the n max-tree leaves with highest extinction values to the root.

4. Prune the nodes that were not marked in the previous step.

5. Reconstruct the image from the filtered tree.

The EF procedure in the max-tree is illustrated in Figure 3.6. Suppose that $n = 3$ and nodes 7, 14 and 15 (the yellow nodes) of Figure 3.6(a) are the leaves with highest extinction value according to the attribute being analyzed. The nodes in the paths from these leaves to the the root are marked in red, Figure 3.6(b), the remaining nodes are pruned. The resulting tree is illustrated in Figure 3.6(c).

The formal definition of EF when filtering maxima is the following: consider that $Max(I) = \{M_1, M_2, ..., M_N\}$ denotes the set of regional maxima of the image I. M_i is an image the same size as I with zeros everywhere except in the positions of the pixels that compose M_i, where the gray-value is the value of the maximum. Each regional maxima M_i has an extinction value ϵ_i corresponding to the increasing attribute being analyzed. The EF of I that preserves the n maxima with highest extinction values, $\mathrm{EF}^n(I)$, is given by the morphological reconstruction by dilation (Soille, 2003) as follows:

$$\mathrm{EF}^n(I) = (I \triangle_D g), \tag{3.10}$$

where the mask image g is given by:

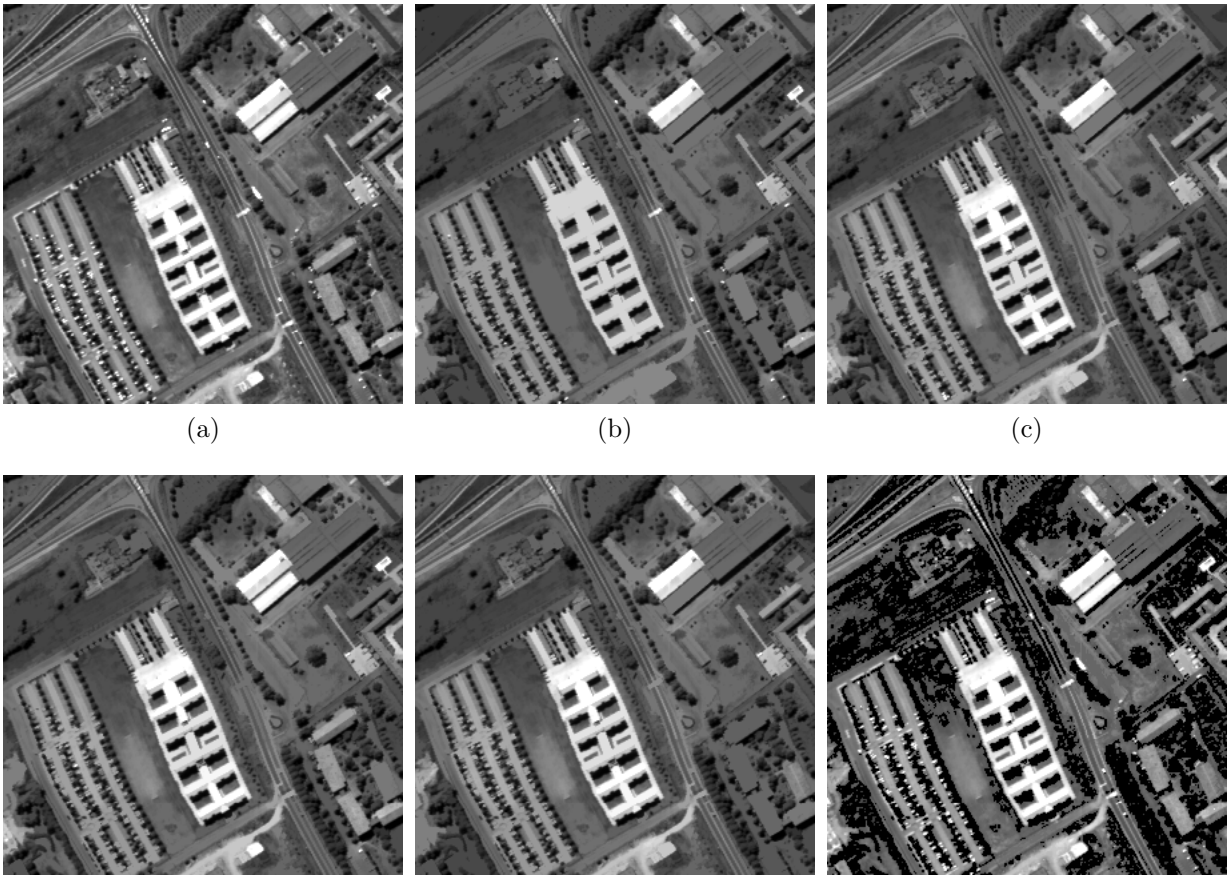
$$g = \max_{i=1}^{n}\{M'_i\}, \tag{3.11}$$

where max is the pixel-wise maximum operation. M'_1 is the maximum with the highest extinction value, M'_2 has the second highest extinction value, and so on.

EFs are connected filters, *i.e.* do not blur the image. Any filters that can be implemented in the max-tree is connected (Salembier et al., 1998). They are also idempotent filters, this comes straight from the definition, since they alter the image only the first time they are applied. EFs are extrema oriented and they have three parameters to set: the kind of extrema

it is going to filter (minima or maxima), the attribute being analyzed, and the number of extrema (n) to be preserved.

Figure 3.7(a) illustrates a component of an hyperspectral image acquired over the Pavia university using a reflective optics system imaging spectrometer. The results of applying height, area, volume, and bounding-box diagonal (all increasing attributes) EFs set with $n = 25$ are depicted in Figure 3.7(b)-(e). The original image has 3260 maxima.



(a) (b) (c)

(d) (e) (f)

Figure 3.7: (a) Original satellite image. (b)-(f) Height, area, volume, bounding-box diagonal and standard-deviation extinction filters with $n = 25$.

3.6 Space of Shapes

Xu et al. (2012) proposed to build max-trees of tree-based image representations, i.e. build a max-tree of a max-tree or a max-tree of a tree of shapes (Monasse and Guichard, 2000). In this methodology, instead of thresholding pixel intensities, the non-increasing attribute computed from the max-tree nodes are thresholded. This second tree construction takes the image to the space of shapes allowing the creation of a novel class of connected operators from the leveling family and more complex morphological analysis, such as the computation of extinction values for non-increasing attributes. This methodology was used for blood vessels segmentation, a generalization of constrained connectivity (Xu et al., 2013), and hierarchical segmentation (Xu et al., 2016). The space of shapes filtering fluxogram is depicted by the red path of Figure 3.8. An example of the second max-tree construction

using the non-increasing aspect ratio attribute on a synthetic image is presented in Figure 3.9.

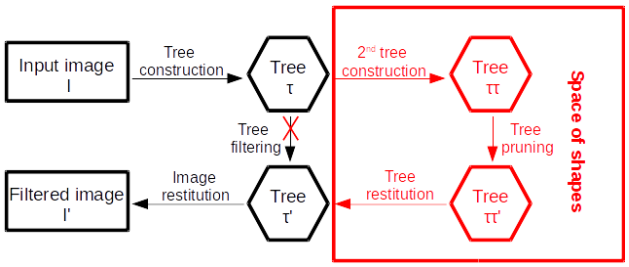


Figure 3.8: Max-tree and space of shapes filtering fluxogram.

After building the second max-tree, the attribute used becomes increasing in this space. Therefore, it is possible to compute extinction values and extinction filters for non-increasing attributes. The procedure for computing extinction filters for non-increasing attributes using the max-tree is the following:

1. Build the image max-tree if filtering maxima (anti-extensive) or min-tree if filtering minima (extensive).
2. Compute the second max-tree using the non-increasing attribute chosen.
3. On the second max-tree, compute extinction values for the non-increasing attribute.
4. On the second max-tree, mark all nodes on the paths starting from the n max-tree leaves with highest extinction values to the root.
5. On the second max-tree, filter the nodes that were not marked in the previous step.
6. Recover the initial max-tree (min-tree) from the second max-tree.
7. Reconstruct the image.

The extinction filter using the aspect ratio attribute applied to the image depicted in Figure 3.1(a) set with $n = 2$ is shown in Figure 3.10. The nodes to be preserved were highlighted in blue in Figure 3.1.

An illustration of the extinction filter applied to a satellite image (Figure 3.7(a)) using the non-increasing standard-deviation attribute, and $n = 25$ is depicted in Figure 3.7(f). The EF result using the non-increasing attribute has a different behavior than the EFs using increasing attributes due the fact it can remove internal nodes, *i.e.* it is not necessarily a pruning like EFs with increasing attributes.

3.7 Chapter Conclusions

In this chapter, we reviewed the max-tree, its signature analysis, and the space of shapes. Extinction filters were proposed and mathematically defined. Also, we explained how to use the space of shapes to extend extinction filters for non-increasing attributes. Extinction filters are further analyzed in the next chapters. EFs for both increasing and non-increasing

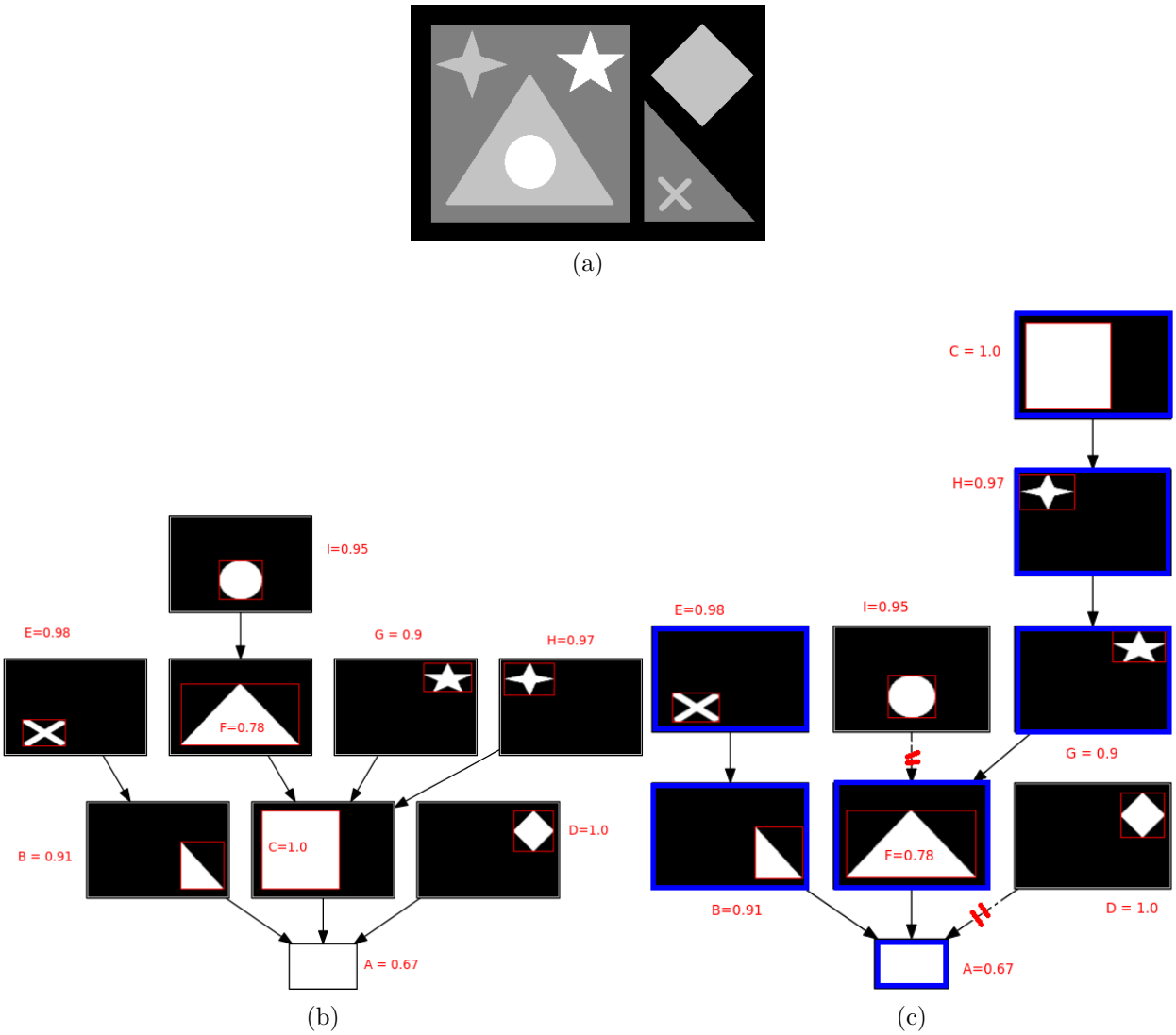


Figure 3.9: (a) Synthetic image (b) its max-tree and (c) second max-tree using aspect ratio as the attribute for the second tree construction.

attributes are the basis of the Extinction Profiles methodology for classification of remote sensing data that will be presented in Chapter 6 and it is also used in the brain extraction approach presented in Chapter 8.

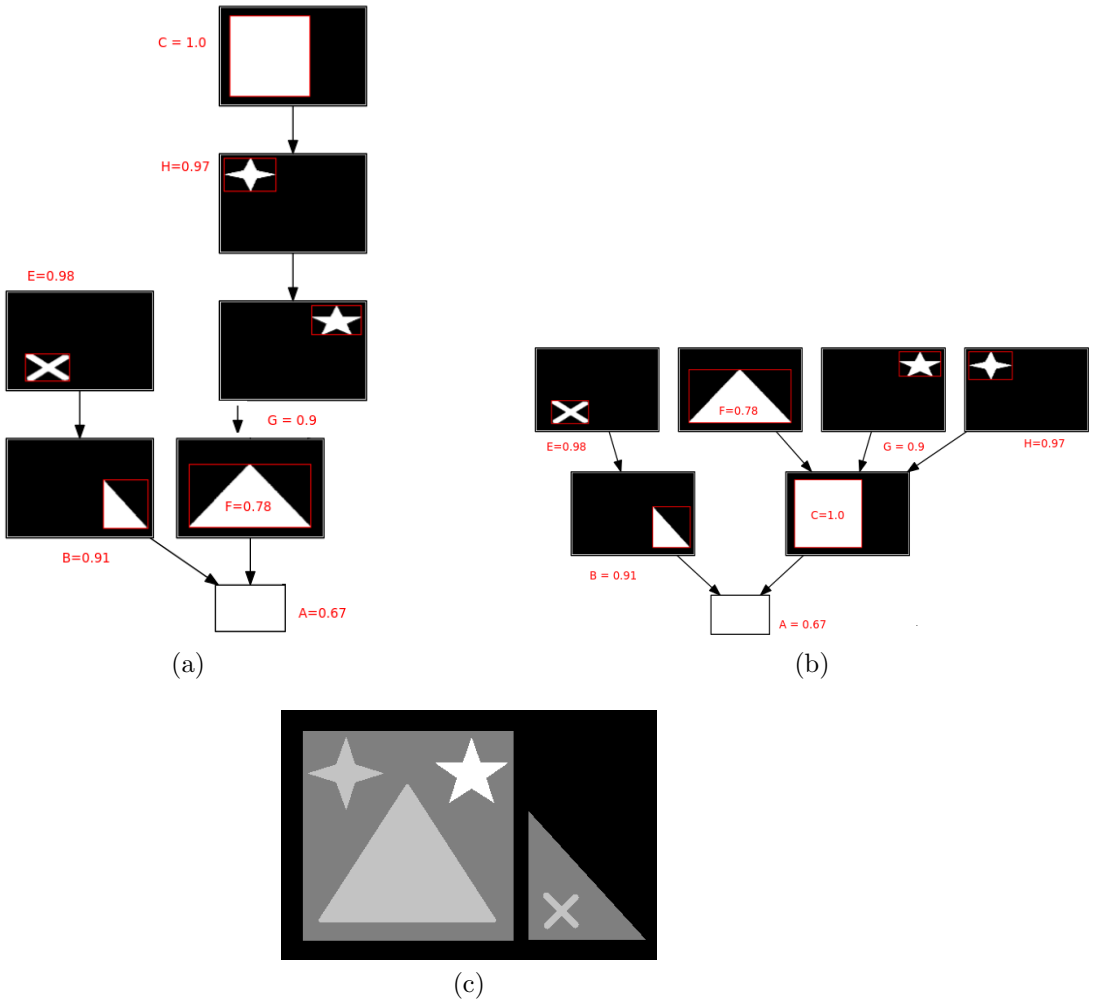


Figure 3.10: (a) Second max-tree after filtering step. (b) recovered initial max-tree after filtering step. (c) Resulting image after the filtering procedure.

Chapter 4

An Array-based Node-oriented Max-tree Representation

4.1 Introduction

The max-tree was proposed by Salembier et al. (1998) as a suitable and efficient structure to implement anti-extensive connected operators. It can be build in quasi-linear time. Its construction algorithms can be classified in three classes: immersion algorithms (Berger et al., 2007; Najman and Couprie, 2006), flooding algorithms (Salembier et al., 1998; Wilkinson, 2011) and merge-based algorithms (Wilkinson et al., 2008; Matas et al., 2008). The best algorithm depends on memory restrictions and the image quantization. A decision tree to choose the most appropriate max-tree construction algorithm in each case is presented in (Carlinet and Géraud, 2014).

Many max-tree algorithms represent the max-tree using a parent array the same size as the image it represents that stores the parent relationship (Wilkinson et al., 2008). Carlinet and Géraud (2014) argue that just having the parent array is an incomplete representation, since it is not sufficient to easily perform tree traversals. They prefer to represent the max-tree using a pair of arrays parent and S, each one with the same number of elements as the image. The S is an ordering array, such that when browsing its elements allows to traverse the tree from its root to its leaves. This representation can be seen as a pixel oriented max-tree representation, since the max-tree nodes are implicitly represented by the level-roots of the parent array. Huang et al. (2003) proposed another form to represent the max-tree using a linked list and a hash table. They store in each element of the linked list: the node ID, its father ID, a list of children IDs, and a list of pixels coordinates. Their structure can be seen as a node oriented max-tree, since it provides direct access to the max-tree nodes, but it is not array based.

In this chapter, we present an array-based node-oriented max-tree representation that can be obtained in linear time on the number of image pixels from the parent/S max-tree representation. Our representation uses less memory and is more suitable for parallel processing using OpenMP (Open Multi-Processing), which is an API that supports multi-platform shared memory multiprocessing programming in C++ and other languages (Dagum and Menon, 1998). Our experiments show that computing the height attribute on our proposed structure is on average 11.4 times faster than computing the height attribute on the parent/S max-tree representation. Our OpenMP parallel filtering implementation in a 4-core

Algorithm 1 Algorithm to filter the parent/S max-tree structure.

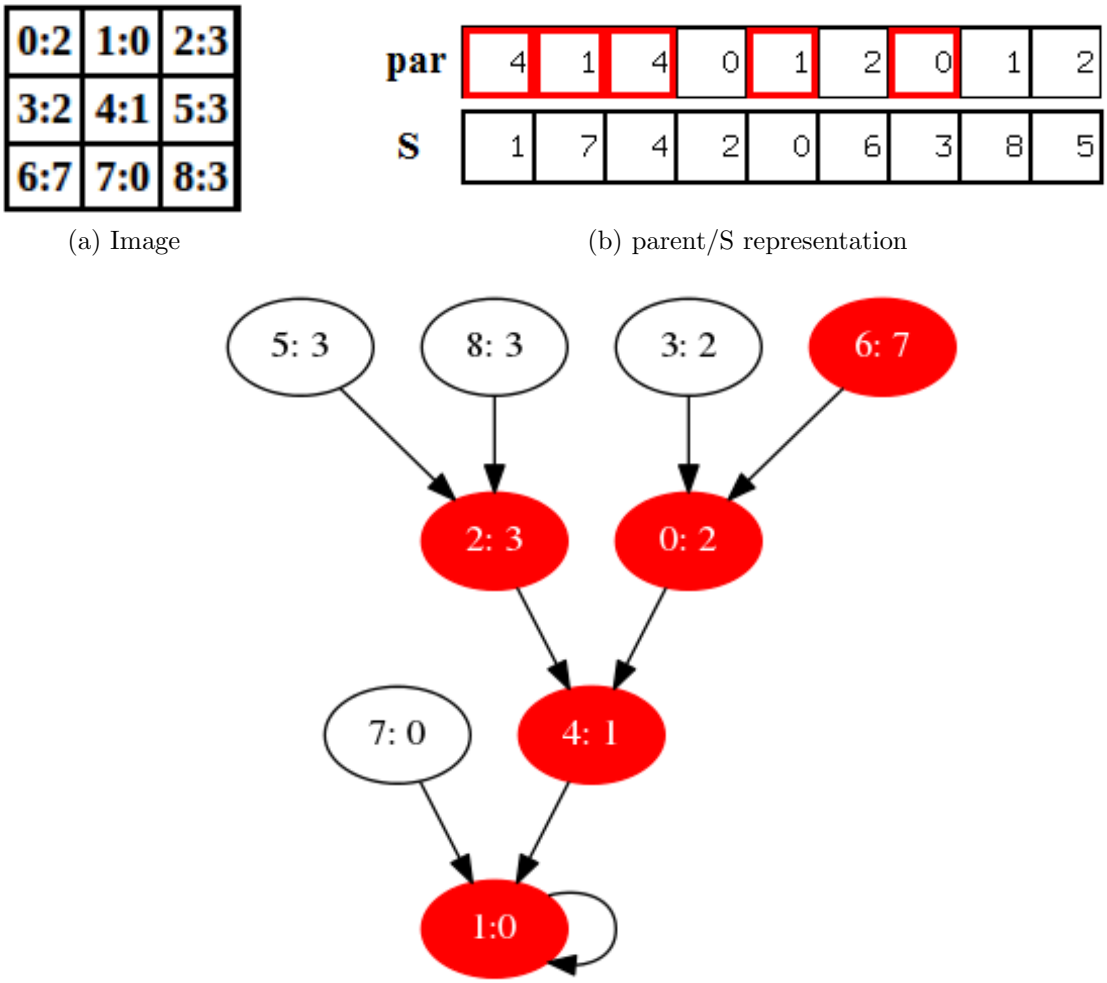
```
1: function DIRECT-FILTER(S,parent,f,attr)
2:     p_root ← S[1]
3:     if attr[p_root] < λ then
4:         out[p_root] ← 0
5:     else
6:         out[p_root] ← f[p_root]
7:     for p ∈ S forward, do
8:         q ← parent[p]
9:         if f[q] = f[p] then                          ▷ p not canonical
10:            out[p] ← out[q]
11:        else if attr[p] < λ then                     ▷ Criterion failed
12:            out[p] ← out[q]
13:        else                                         ▷ Criterion passed
14:            out[p] ← f[p]
15:    return out
```

CPU is faster than the filtering based on the parent/S structure. Also, when performing successive filtering steps our sequential implementation becomes faster than the parent/S filtering. The work presented here was published at the *2015 International Conference on Image Processing* (Souza et al., 2015a).

4.2 parent/S Max-tree Structure

The parent/S max-tree structure is composed of the parent array that encodes the parent relationship between pixels, and the S array which stores an ordering of the pixels that allows to traverse the tree from the root to its leaves. Each node of the max-tree is represented by a single pixel called the canonical element or level-root. A pixel p is a level-root if $parent[p] = p$ (p is the root of the max-tree) or $f[parent[p]] < f[p]$. Therefore, this structure also needs to store the image corresponding to the max-tree, in order to know which pixels are level-roots. This representation requires $2NI + NU$ bytes of memory space where N is the number of image pixels, I the size in bytes of an integer, since points stored in S and parent are actually positive offsets in a pixel buffer, and U is the size in bytes of the image pixels. We call this structure a pixel oriented max-tree representation, since the max-tree nodes, i.e level-roots, are implicitly represented in the structure. An example of the pixel oriented max-tree representation where the level-roots are marked in red is displayed in Figure 4.1.

An algorithm to compute the max-tree direct filter that can be implemented in this structure with a complexity linear in the number of image pixels was proposed in (Carlinet and Géraud, 2014). The algorithm is depicted in Algorithm 1. Basically, on their algorithm, a loop is performed visiting each image pixel according to the ordering provided by S, this way each level-root is processed before its descendants and the scan occurs from the root to the leaves. If the pixel currently being processed is not a level-root, the output image receives the gray-level its parent received in the output image. If the pixel is a level-root the filter criterion is tested, if true its gray-level in the output image does not change, if false it also receives the gray-level its parent received in the output image. In this algorithm, the root is treated separately, outside the main loop, and since it is dependent on the scan order, it is not possible to implement it in parallel.

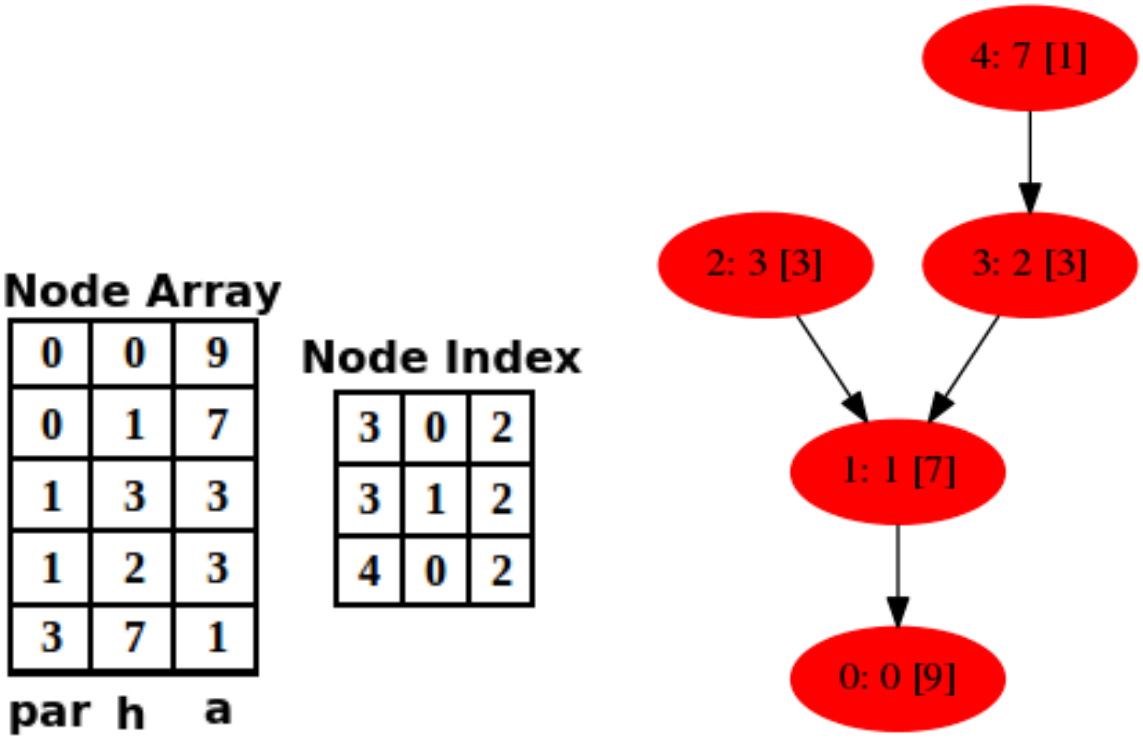


(a) Image

(b) parent/S representation

(c) parent/S representation

Figure 4.1: (a) Image f (pixel position: gray-level), (b) parent/S max-tree representation, and (c) the pixel oriented max-tree illustration (pixel ID: gray-level). The level-roots are highlighted in red.



(a) NA/NI (b) Node-oriented max-tree

Figure 4.2: (a) NA/NI max-tree representation , and (b) the node oriented max-tree illustration (node ID: h [area]).

4.3 Node Array/Node Index Max-tree Structure

The max-tree structure we propose is a node-oriented max-tree. It is composed of two arrays: node index (NI) and node array (NA). NI is an array with the same shape of the original image, and its elements point to the max-tree nodes they belong. NA is an array that is organized such that its sequential scan is a tree traversal from the root to its leaves. This organization avoids the requirement of the array S. It is an array $M \times nattr$, where M is the number of max-tree nodes and $nattr$ is the number of attributes it stores. Actually, each column of NA can be seen as a different array each one storing different attributes of the nodes. In order to be able to recover the image from this representation, NA has to store at least the parent relationship (par), and the gray-level (h) of each node. Our structure does not need to keep a copy of the image. It requires $(N + 2M)I$ bytes of memory, assuming all attributes are of type integer. If we consider that I is a 4 bytes integer and the image pixels size U requires 1 byte of memory, we have that our representation is more memory efficient, as long as the ratio N/M is greater than 1.6, which is true for most images, and our experiments in Section 4.4 confirm that. The illustration of the node oriented max-tree of the image shown in Figure 4.1(a) is depicted in Figure 4.2. In this example, NA also stores the area of each connected component (column a of NA).

NA and NI can be computed from the parent/S structure in linear time with respect to N. Each line of NA represents a node, and its attributes are stored in the columns. The algorithm for that is described in Algorithm 2. In this algorithm, we store in NA the parent relationship (par), the gray-level (h), and the area (a). First, we perform a loop in the pixels in reverse order of S to compute the areas, and to store the sorted level-roots (lines 7 through

Algorithm 2 Algorithm to compute NA and NI.

```
1: function COMPUTE-NA-NI(parent, S,f)
2:     ncols ← 3
3:     nlvroots ← 0
4:     sorted_lvroots ← []
5:     for i ← 1, N do                                          ▷ parallel loop
6:         NI[i] ← −1,   area[i] ← 1
7:     for p ∈ S backward, do
8:         area[parent[p]] ← area[parent[p]] + area[p]
9:         if f[parent[p]] ≠ f[p] or   parent[p] = p then
10:            sorted_lvroots.insert(p)
11:            nlvroots ← nlvroots + 1
12:    NA ← empty_array(nlvroots, ncols)
13:    k ← 0
14:    for p ∈ sorted_lvroots backward, do
15:        NI[p] ← k
16:        NA[k, 1] ← NI[parent[p]]                             ▷ parent (par)
17:        NA[k, 2] ← f[p]                                      ▷ gray-level (h)
18:        NA[k, 3] ← area[p]                                   ▷ area (a)
19:        k ← k + 1
20:    for i ← 1, N do                                          ▷ parallel loop
21:        if NI[i] = −1 then
22:            NI[i] ← NI[parent[i]]
23:    NA[1, 3] ← NA[1, 3]/2                                    ▷ root area was doubled
24:    return NA, NI
```

11). Then, another loop is performed in the level-roots, in order to fill NA and initialize NI (lines 14 through 19). Finally, a final loop is performed in order to fill the remaining positions in NI (lines 20 through 22). This last loop can be performed in parallel. The restitution of the image from the NA/NI structure is performed through a look-up table (Algorithm 3), where each pixel receives the gray-level of the node it belongs. This is a parallel loop.

Algorithm 3 Image restitution algorithm.

```
1: function GET-IMAGE(h, NI)
2:     N ← NI.size
3:     for i ← 1, N do                                          ▷ parallel loop
4:         f[i] ← h[NI[i]]
       return f
```

The direct filter algorithm for our proposed structure is described in Algorithm 4. The algorithm has four main stages: the first consists in updating the parent pointers (lines 8 through 15), the second consists in calculating the look-up table used to update NI (lines 16 through 21), the third stage updates NI (lines 22 and 23), and the fourth consists in the compression of NA by removing the lines corresponding to the filtered nodes (line 24). The loops in the algorithm are mainly linear in the number of nodes M, the only loop in N is the look-up table in lines 22 and 23 that update NI, which can profit from parallelism. The loops in M can not be done in parallel, because they make use of the ordering property of NA, which allows to scan the tree from its root to its leaves.

Algorithm 4 Direct filter algorithm using NA/NI.

```
1: function DIRECT-FILTER(NA, NI, to_keep)
2:     to_keep[1] ← True                                   ▷ cannot remove the root
3:     M ← NA.lines
4:     N ← NI.size
5:     for i ← 1, M do                                     ▷ parallel loop
6:         lut[i] ← i
7:         nearest_ancestor_kept ← 0
8:     for i ← 1, M do
9:         if (not to_keep[i]) then
10:            temp ← nearest_ancestor_kept[par[i]]
11:            nearest_ancestor_kept[i] ← temp
12:            lut[i] ←= lut[temp]
13:        else
14:            nearest_ancestor_kept[i] ← i
15:            par[i] ← nearest_ancestor_kept[par[i]]
16:    index_fix[1] ← (1 − to_keep[1])
17:    for i ← 2, M do
18:        index_fix[i] ← index_fix[i − 1] + (1 − to_keep[i])
19:        lut[i] ← lut[i] + index_fix[i]
20:    for i ← 1, M do
21:        par[i] ← lut[par[i]]
22:    for i ← 1, N do                                     ▷ parallel loop
23:        NI[i] ← lut[NI[i]]
24:    Remove NA lines corresponding to filtered nodes     ▷ parallel loop
25:    return NA, NI
```

4.4 parent/S versus NA/NI

In this section, we present an algorithm for computing the height attribute from the NA/NI structure, and we compare its processing time against the parent/S algorithm. Also, we compare filtering times, for a single area-open filter (Vincent, 1994), and an area-open followed by the hmax filter (Salembier and Oliveras, 1996) in each structure. All algorithms were implemented in C++, and wrapped in Python using SWIG (Beazley, 1996). The experiments were performed on a 4-core virtual machine running in the Intel Xeon X5675 server with clock of 3.06 GHz.

4.4.1 Height Attribute Computation

The height or dynamics (Grimaud, 1992) of a max-tree node is given by the maximum difference between the node gray-level and the gray-level of one of its descendants. The algorithms for computing the height attribute using the NA/NI structure and using the parent/S structure are displayed in Algorithm 5. The algorithms are basically the same, the main differences are that in our structure it is linear in the number of max-tree nodes (M) while in the parent/S it is linear in the number of image pixels (N), and the parent/S algorithm makes one more memory access than NA/NI in each iteration of the second for loop of the algorithm. The same reasoning we illustrated for the computation of the height attribute also applies for the computation of some other attributes, such as volume.

In order to compare the algorithms processing times, we used a dataset with 50 images chosen at random from the Internet with different sizes. We used connectivity $C8$ to build the max-trees, and we computed the height of the max-tree nodes for each image. Box

Algorithm 5 Algorithms for computing the height of the max-tree nodes in both the NA/NI and the parent/S structures.

```
1: function COMPUTE-HEIGHT-NA-NI(par, h)
2:     M ← length(par)
3:     for i ← 1, M do
4:         height[i] ← 0
5:     for i ← M, 1 do
6:         aux ← par[i]
7:         aux2 ← height[i] + h[i] − h[aux]
8:         height[aux] ← max(height[aux], aux2)
       return height
9: function COMPUTE-HEIGHT-PARENT-S(parent, S,f)
10:    N ← length(parent)
11:    for i ← 1, N do
12:        height[i] ← 0
13:    for i ← N, 1 do
14:        aux ← parent[S[i]]
15:        aux2 ← height[i] + f[i] − f[aux]
16:        height[aux] ← max(height[aux], aux2)
       return height
```

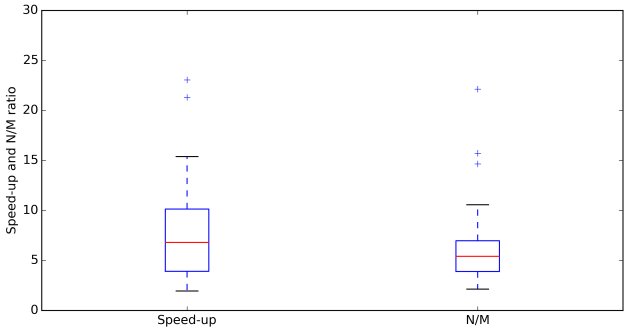


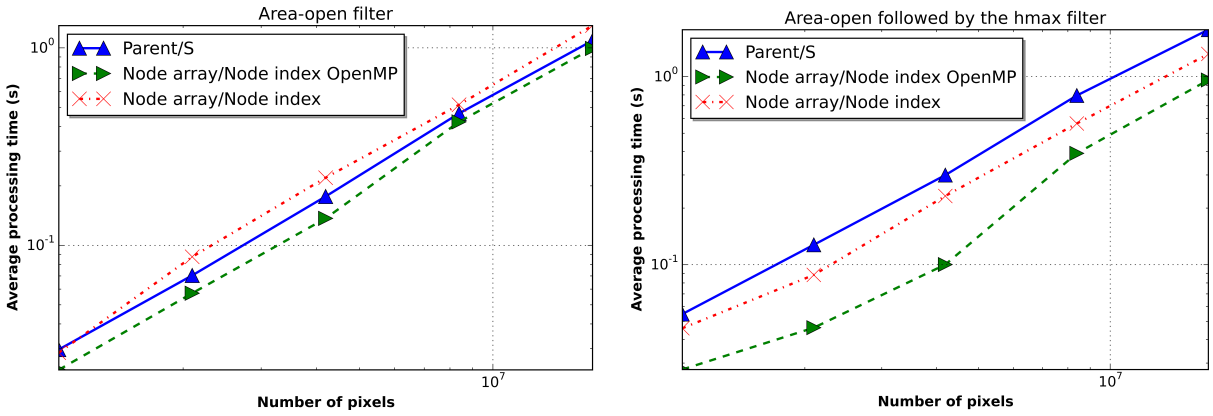
Figure 4.3: Box plots showing the speed-up summary for the height computation algorithms using the NA/NI structure in relation to the algorithm that uses the parent/S structure, and the ratio between N and M of the images used in the dataset.

plots illustrating the speed-up of our algorithm, and the ratio between number of pixels and number of max-tree nodes are depicted in Figure 4.3.

The results show that our algorithm is faster for all the images used in the dataset, since the minimum speed-up was of 1.9. We obtained an average speed-up of 8.9, with a standard deviation of 11.2, which can be explained by the fact that our algorithm is dependent on the number of max-tree nodes, which depends of the characteristic of the image. For all images, our max-tree representation is more memory efficient, since the smaller ratio between the number of pixels and the number of nodes was of 2.1.

4.4.2 Filtering Comparison

The filtering times were compared using the same dataset used in the previous experiment. The images were interpolated so that initially they had 1 mega-pixel (MP), then they were concatenated with themselves to generate each time an image twice as large as the previous one. It was used connectivity $C8$ to build the max-trees. We tested the filtering algorithms in two different cases: the first case is simply an area-open filter that removes



(a) (b)

Figure 4.4: Filtering time comparison. (a) Area-open filter, and (b) area-open followed by the hmax filter.

all nodes with area smaller than 80. The second case is the same area-open filter followed by a hmax filter that removes all nodes with height (contrast) smaller than 10. The main differences between the two cases is that our structure can compute the height attribute faster, and that we only have to recover the image at the end of the filtering steps, while the parent/S structure has to maintain the image updated during each filtering step. We tested our structure both for the sequential implementation and an implementation that uses OpenMP (Dagum and Menon, 1998) to accelerate the parallel loops of the algorithms we presented. The results of the experiments are summarized in Figure 4.4. The measurements do not consider the time to build the parent/S max-tree structure, which is common to both methods. In the first case, our sequential implementation was on average 1.14 times slower, and the OpenMP parallel implementation was 1.20 times faster than the filtering using the parent/S structure. In the second case, both our implementations were faster than the parent/S structure. The sequential was on average 1.34 times faster and the OpenMP parallel was 2.32 times faster. Our implementation using OpenMP was on average 1.38 and 1.74 times faster than the sequential implementation, for the one filtering and two filtering cases, respectively.

4.5 Chapter Conclusions

We presented an array-based node-oriented structure to represent the max-tree. Our structure is more memory efficient as long as the ratio between the number of pixels and the number of nodes is greater than 1.6, which was true in all the experiments we performed. Also, since we have direct access to the max-tree nodes, some attributes can be computed much faster in our structure, as it was seen in the height attribute computing experiment, where we obtained an average speed-up of 11.4. The experiments show that the OpenMP parallel implementation of our algorithm is faster than the algorithm proposed by (Carlinet and Géraud, 2014), and when performing a sequence of max-tree filtering steps our sequential implementation is also faster.

An open-source implementation of our algorithms is available as a max-tree toolbox entitled *iamxt*, which is explained in depth in Appendix A. As a future work, we intend to investigate further the advantages of our structure for filtering, feature extraction, and

Chapter 5

Comparison between Extinction Filters and Attribute Filters

5.1 Introduction

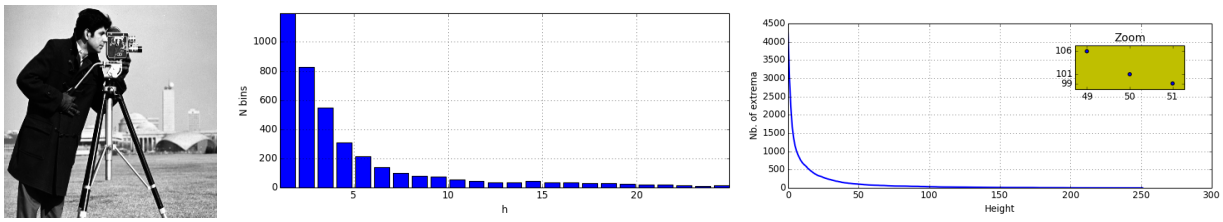
Attribute filters are the first set of connected filters that were proposed in the literature. If the attribute is increasing, they can be implemented through the max-tree pruning of all the nodes that do not meet the threshold used by the filter. The most common attribute filters are contrast filters, *hmax* (Salembier and Oliveras, 1996), size filters, *area-open* (Vincent, 1994), and a mix of contrast and size filtering, *vmax* (Vachier, 1995).

Extinction filters are based on the concept of extinction values, which are a powerful tool to measure the persistence of an increasing attribute, and are useful to discern relevant from irrelevant extrema, usually noise. The concept of extinction values was proposed in (Vachier, 1995), and it can be seen as an extension of the concept of dynamics (Grimaud, 1992). The most common attributes used to compute extinction values are height, area and volume, and they can be efficiently computed in the max-tree structure (Silva and Lotufo, 2008). Extinction filters are connected filters that preserve only the extrema with highest extinction values. Unlike attribute filters, they preserve the gray-level value of the extrema kept.

In this chapter, we compare attribute filters against extinction filters, both using a single increasing attribute. More specifically, the attributes used are height, area and volume. Extinction and attribute filters, set to preserve the same number of extrema, are analyzed and compared in depth in this work. The robustness of the filters are analyzed through repeatability tests comparing the Maximally Stable Extremal Region (MSER) (Matas et al., 2002) affine detector, which can be efficiently computed from the max-tree (Donoser and Bischof, 2006), and the MSER detector using attribute and extinction filters as a pre-processing step are shown. This is done also for the Hessian-Affine (Mikolajczyk and Schmid, 2002) detector. The results indicate that extinction filters are better than attribute filters with respect to simplification for recognition, since they preserve more the correspondences found by affine detectors. This chapter is a summary of the work published at the *2015 International Symposium in Mathematical Morphology* (Souza et al., 2015b). These results lead to the development of the methodology for classification of satellite images presented in the next chapter.

5.2 Setting Number of Extrema with Attribute Filters

Attribute filters, such as the *hmax*, *vmax*, and the *area-open* are usually used to simplify images in terms of number of flat zones, but they may also be used to set number of extrema (maxima or minima) to be preserved in an image. In order to do that, first the extinction values of the increasing attribute being analyzed have to be calculated. Then, the extinction histogram is computed. The relationship between number of maxima (minima) in the image and the parameter of the attribute filter is given by the curve attribute value versus number of maxima (minima) minus the cumulative distribution of the extinction histogram. This curve may have discontinuities, since typically in a real image there are extrema with the same extinction values, therefore when using attribute filters it is often not possible to set the exact number of extrema to be preserved. This case is illustrated in Figure 5.1. The height extinction histogram of the Cameraman image, Figure 5.1(a), is shown in Figure 5.1(b). The curve height versus number of maxima is shown in Figure 5.1(c). The zoom in the plot highlights the discontinuities in this curve. For instance, if the image is filtered with the *hmax* filter with $h = 50$ number of maxima in the resulting image will be 101, but if it is filtered with $h = 51$ number of maxima in the resulting image will be 99. It is possible to enhance the method to avoid extinction ties, for simplicity, we prefer not to include this enhancement, as the differences in the filter results would be negligible for the comparative experiments.



(a) (b) (c)

Figure 5.1: (a) Cameraman image. (b) Height extinction histogram, and (c) the curve that relates number of maxima in the image and the parameter h of the *hmax* filter.

A small portion of a vehicle license plate image is depicted in Figure 5.2(a). The results of applying the area EF and the *area-open* filter set to keep only the three most relevant minima are shown in Figure 5.2(b)-(c). The graphs corresponding to the original max-tree and the filtered max-trees are shown in Figure 5.3. The original image is 30×50 pixels. The original max-tree has 322 nodes, of which 54 are leaves. The max-tree filtered with the area EF has 174 nodes, and the max-tree filtered with the *area-open* has 72 nodes. This illustration makes clear the fact that attribute filters erode the heights (gray-level value) of the maxima, while the EF preserves it.

5.3 Experiments

In this section, the processing time, structural similarity and simplification performance of AF and EF are analyzed. We also analyze their robustness using them as a pre-processing step before extracting Hessian-Affine and MSER regions and performing repeatability tests. In order to do that, it was used the benchmark dataset proposed in (Mikolajczyk et al.,

(a) (b) (c)

Figure 5.2: (a) Original license plate image. (b) License plate filtered with the area EF, and (c) the *area-open* filter.

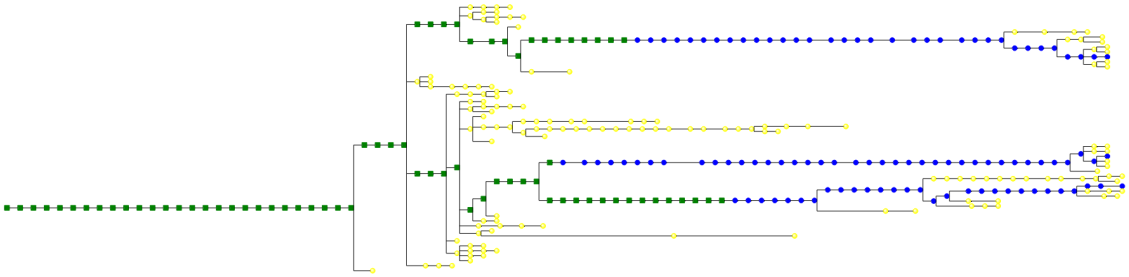
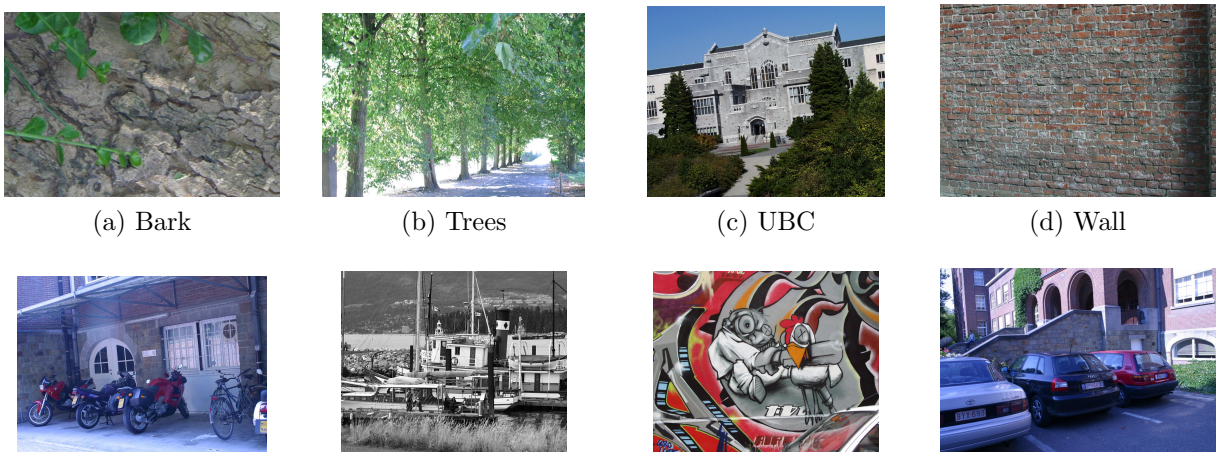


Figure 5.3: Graph corresponding to the original max-tree (all nodes), the max-tree after the area EF (green squares and blue circles), and the max-tree after the area-opening filter (green squares).

2005). The dataset is composed of structured and textured scenes divided in eight groups, where in each group a different type of image transformation was applied, such as scale change, blur, and compression. A sample image of each group of the dataset is depicted in Figure 5.4, and the information concerning their size, and deformation applied in each group of images are summarized in Table 5.1.



(a) Bark (b) Trees (c) UBC (d) Wall

(e) Bikes (f) Boat (g) Graffiti (h) Leuven

Figure 5.4: Sample images of each group of the dataset.

5.3.1 Processing Time

The processing time of AFs and EFs is dependent of image size, number of max-tree nodes, and number of tree leaves. In order to compare their processing times, we chose one sample image of each group in the dataset, and we cropped them so each image would be 512 pixels by 512 pixels. Each image was filtered using the *area-open* and the area EF using connectivity $C4$ and set to preserve different percentages of extrema. For each percentage and each image of every group the

Table 5.1: Summary of size, and deformation applied in each group of the dataset.

Group	Transformation	Size
Bark	scale change + rotation	765×512
Bike	increasing blur	1000×700
Boat	scale change + rotation	850×680
Graffiti	viewpoint angle	800×640
Leuven	decreasing light	900×600
Tree	increasing blur	1000×700
UBC	JPEG compression	800×640
Wall	viewpoint angle	1000×700

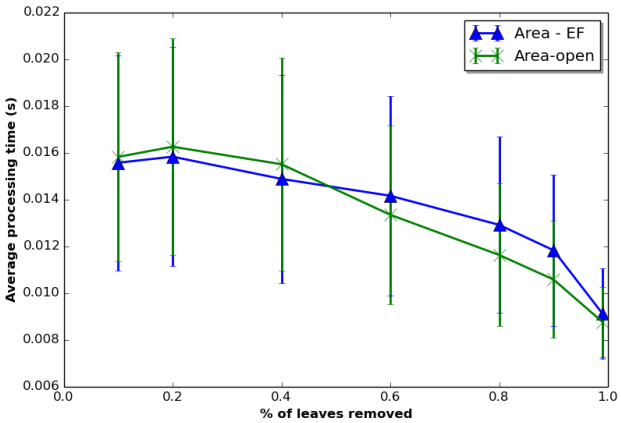
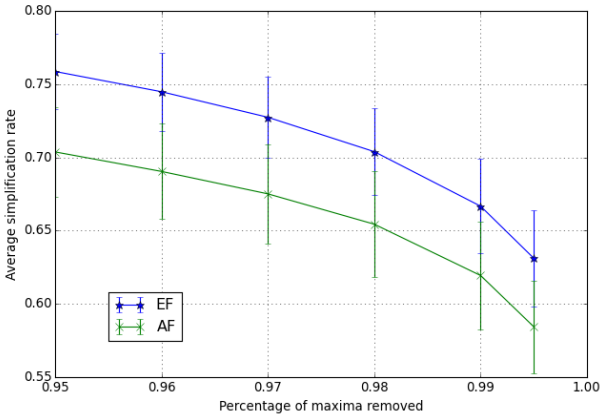


Figure 5.5: Average processing time and standard deviation of the *area-open* and the volume EF.

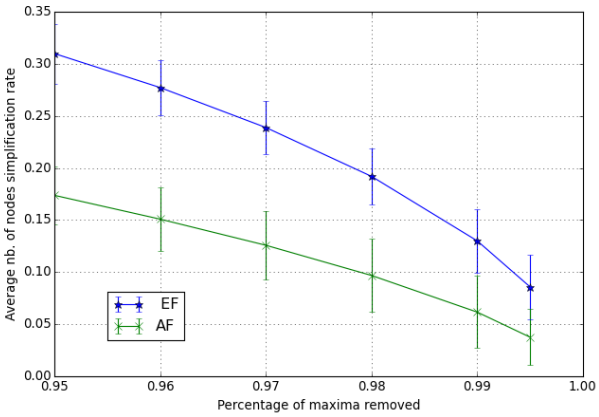
filtering was repeated 20 times. The average processing time and its standard deviation is depicted in Figure 5.5. The processing time computed encompasses just the filtering of the max-tree, since the max-tree construction, and image restitution times are the same for both filters. The time was measured in a *Intel Core i7* processor with 8GB of memory and a clock frequency of $3.4GHz$. The plot shows that both attribute filters and extinction filters have similar processing times, when set to preserve the same number of extrema, and that the processing time varies little with the number of leaves preserved. The main differences in the EF and the AF algorithms is that the EF needs to sort the leaves extinction values from the highest value to the lowest, and the AF needs only to compute the cumulative extinction histogram, which can be done linearly, while the sorting complexity depends on the algorithm, but usually requires more time. Other processing time differences can be explained by the data structure we use to represent the max-tree (Souza et al., 2015a).

5.3.2 Simplification Performance and Structural Similarity

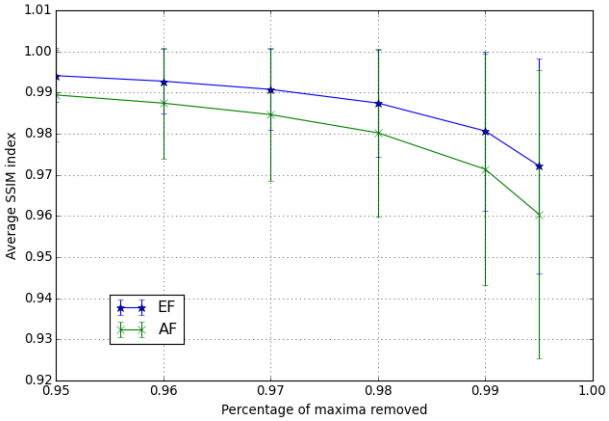
In order to evaluate the simplification performance, we computed the flat zones simplification rate given by the number of flat zones in the filtered image divided by the number of flat zones in the original image, and the number of max-tree nodes simplification rate, which is the same metric, but using the number of max-tree nodes instead of the number of flat zones. The images similarity was evaluated through the Structural Similarity (SSIM) index (Wang et al., 2004) between the original image and the filtered image. One sample image of each group in the dataset was used to compute these metrics. The mean values and their standard deviations are illustrated in Figure 5.6. As expected, AF have a greater simplification power than EF, since they erode the height of the extrema they keep (see Figure 5.3). On average it reduced almost 5% more the number of flat



(a) Flat zones



(b) Nodes



(c) SSIM index

Figure 5.6: (a) Average flat zones, (b) nodes simplification rates, and (c) SSIM index for different percentages of maxima removed.

zones on the image, and twice the number of max-tree nodes when compared to EF. On the other hand, EF obtained a SSIM index 0.7% higher on average than AF.

5.3.3 Robustness Evaluation through Repeatability Tests

In this subsection, it is evaluated the influence of AF and EF as a pre-processing step before the extraction of MSER and Hessian-Affine regions. We chose specifically these two detectors, in view of the fact that they achieved best results in the survey reported by (Mikolajczyk et al., 2005), and because of their distinct behaviors, MSER is a region detector, while Hessian-Affine is a point detector.

The protocol used to evaluate the results is the one proposed in (Mikolajczyk et al., 2005). The MSER and Hessian-Affine regions where computed using the same parameters and the binary files provided by (Mikolajczyk et al., 2005). In this chapter, the experiments reported used the *area-open* and the area EF as pre-processing filters, but the the same conclusions also hold when using the height and volume attributes. To compute the MSER regions, first the image was filtered using either the EF or the AF. Then, the MSER+ regions were computed. After that, the negative of the original image was filtered and the MSER- regions were computed. For the Hessian-Affine detector, the original image was filtered, then the negative of the filtered image was also filtered before extracting the Hessian-Affine regions.

The percentage of extrema kept was tested as being 10%, 5%, and 3% of the number of extrema of the original images. The number of regions and correspondences found for the group *Bikes*

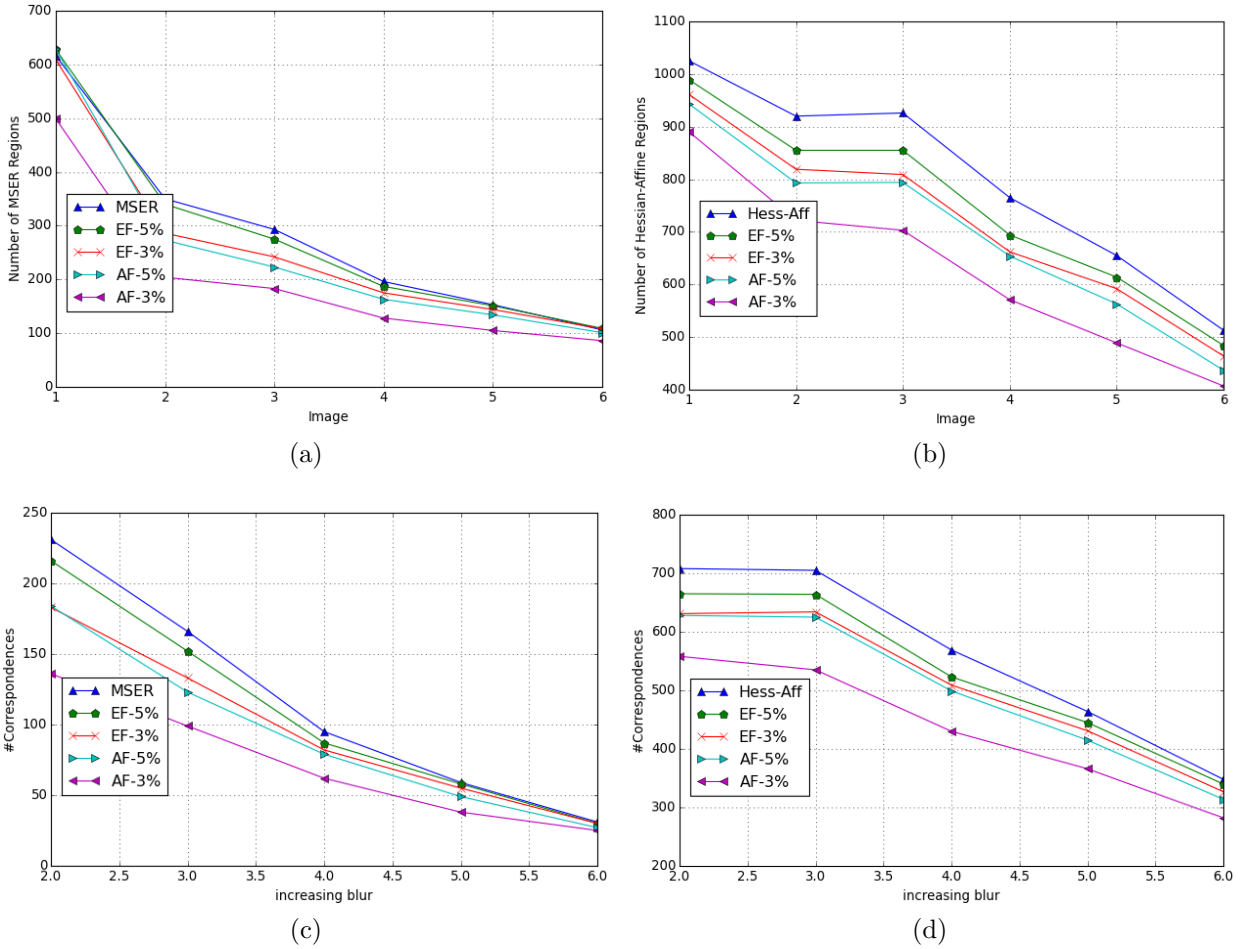


Figure 5.7: (a)-(b) Number of regions and (c)-(d) number of correspondences found for the group *Bikes* of the dataset for the MSER (left) and Hessian-Affine (right) detectors.

is depicted in Figure 5.7 [1]. The plots shown indicate the overall behavior of the filters in the whole dataset. EF preserve a significantly higher number of regions and correspondences, when compared to AF. The number of regions and correspondences found using EF and AF as pre-processing compared to the detectors applied directly to the unfiltered images is summarized in Table 5.2. Filtering 90% of the extrema using EF preserved on average practically 99% of the correspondences and simplified the number of max-tree nodes by a factor of 2.3, and filtering 95% preserved more than 94% of the correspondences and simplified the max-tree nodes by a factor of 3.2 when compared to the unfiltered results. Also, the EF set to preserve 3% of the extrema obtained on average a higher number of regions detected and matches, compared to the AF set to preserve 5% of the extrema.

5.4 Chapter Conclusions

This chapter presented a comparison between attribute filters and extinction filters using the height, area and volume attributes, when they are set to preserve the same or at least the closest value possible of extrema in the image. The methodology to set the number of image extrema using AF was introduced. It was seen that AF erode the height of the extrema kept. Both filters have similar processing times. AF filters have greater simplification power both in terms of

[1]The graphics for all the groups of the dataset are available at *http://adessowiki.fee.unicamp.br/adesso/wiki/code/MainPage/view/*

Table 5.2: Summary of the number of regions and correspondences found using EF and AF as pre-processing. The values are percentages computed in relation to the detectors results applied directly to the unfiltered images.

Filter - % extrema kept	**MSER**		**Hessian-Affine**	
	Δ#Regions	**Δ#Corresp.**	**Δ#Regions**	**Δ#Corresp**
EF - 10%	-0.65%	+0.54%	-2.23%	-2.81%
AF - 10%	-3.13%	-1.50%	-8.22%	-8.72%
EF - 5%	-4.57%	-3.61%	-5.45%	-7.17%
AF - 5%	-12.78%	-13.93%	-17.37%	-18.25%
EF - 3%	-12.91%	-15.11%	-10.18%	-12.94%
AF - 3%	-36.64%	-41.56%	-25.83%	-27.45%

number of flat zones and max-tree nodes, while EF achieve higher structural similarity scores. The repeatability tests performed showed that EF used as a pre-processing preserved a considerably higher number of regions and number of correspondences found by the affine region detectors than using AF as a pre-processing. Although AF has a higher power for simplifying the number of max-tree nodes, it also filters more relevant structures in the image. Therefore, we can conclude there is a trade-off. EF simplify a little less the max-tree structure than AF, but it is better with respect to simplification for recognition, since it preserves better the correspondences found by affine detectors. This comparison showing that EFs are better than AFs lead to the Extinction Profile methodology presented in the next chapter.