Firsname :
Lastname :
Date :

Hierarchical representation

Image Processing Master 2nd Errasmus Mundus

François Merciol

2021-2022

(This page intentionally left blank)

Contents

List of Figures

List of Tables

List of Algorithms

1 Introduction

Earth observation has become essential for activities connected with the environment, whether to understand it or to preserve it.

One of the most efficient manners remains the use of remote sensing imagery. However, due to the increasing amount of data, we have to find both fastest processing and relevant results.

We propose in this document to present the hierarchical representation approach.

Among a great number of spatial-spectral approaches in the literature, morphological attribute profiles (APs) [6, 23] have been widely used to process and describe remote sensing images in the literature.

The reason relies on their powerful multilevel modeling of the image content and their efficient implementation via tree representation.

In fact, the core idea of APs is to explore hierarchical information from a sequence of filtered versions of an image.

These images are obtained by the application of different filter rules (called attributes) characterizing the size and shape of objects present in the image.

1.1 General presentation

It is the support of Image Processing module for the student who follow Erasmus Mundus Master 2nd year.

It contains independent chapters which address different aspect about hierarchical representation.

- Chapters
  - Hierarchical overview gives simple approach and based concepts of trees
  - Implementation presents an efficient way to build trees
  - Inclusion trees shows one family of trees
  - Partition trees shows an other one
  - Dimension shows images not limited to 2 dimensions
  - Attribute enhances information carry within trees
  - Filtering is a way to take benefits of attributes
  - Texture tools add-on associates extra non-hierarchical information from an image
  - Related work gives an example of use for classification

1.2 Goal

Why introduce this module in this training? The goal is to present a way to manage and parse remote sensing images. What we are going to develop is an efficient way of representing information. We will particularly insist on:

- Hierarchical representation of remote sensing images
- Image visualization with tree filtering
- Features extraction capabilities from node attributes
- Training (qgis, gdal, Triskele) for help understanding and memorization

1.3 Needs of hierarchical representation

The needs relate to different aspects such as:

- Earth observation. That is the main issue. The number and the size of data increase so dramatically than we need a scalable approach.
- Modeling. Mathematical morphology is a theory and technique for the analysis and processing of geometrical structures, based on set theory, lattice theory, topology, and random functions. Mathematical morphology is most commonly applied to digital images. The hierarchical representation is a part of Mathematical morphology.
- Abstraction. Thanks to the modeling we are able to define operators. The corresponding modifications help us analyze images.
- Efficient method. Data mining takes benefit of tree by the fact access to final values depend only on the deep of the tree. Each node can be consider as an alternative. The number of level is around $log(N)$ (where N is the number of pixels). That mean, only $log(N)$ answers drive you to the result.
- Tools. As we will see further, the tools we design are efficient for large images. The building process is almost linear complexity and the data mining in logarithm complexity.

1.4 Software validation

To enforce the lesson we will use some tools in Linux environment.

It is important to validate the connection with the server. We will check the data and software access as well.

- SSH keys. Each of you have key with two parts. The public part is sent to the administrator and give access to the server. The private part is

used to custom the tools you will use for connection.

To create a key please read this documentation : https://cluster-irisa.univ-ubs.fr/wiki/acces-ssh.

When you got it, you must upload the public part in the server with this documentation https://cluster-irisa.univ-ubs.fr/wiki/sesame

SSH could be invoked in terminal with a command line.

```
ssh -J proxyssh@cluster-irisa.univ-ubs.fr your-login@dmis
```

Fig. 1 – ssh connection

— DMIS connection. $dmis.univ-ubs.fr$ is the name of the cluster we will use for training.

When connected, this message must appear.

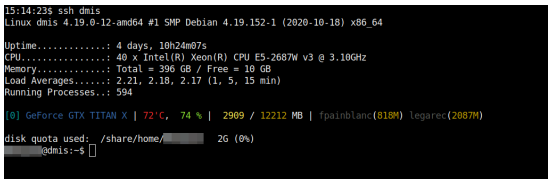


Fig. 2 – Screen connection

— Command shell.

In a terminal you can try the command "cd /share/projects/erasmus1/triskele/" and "ls -la".

The result may look like.

Fig. 3 – ssh ls

We will use command line `gdal_calc.py` for simple modifications. We create a new gray-scale image combined with 3 bands Arles image. The result is in your home directory is.

```
you@dmis:/share/projects/erasmus1/triskele$ gdal_calc.py -R arles.tif --R_band=1 \
-G arles.tif --G_band=2 -B arles.tif --B_band=3 --outfile=${HOME}/test.tif \
'--calc=R*0.2989 + G*0.5870 + B*0.1140'
0 .. 10 .. 20 .. 30 .. 40 .. 50 .. 60 .. 70 .. 80 .. 90 .. 100 - Done

you@dmis:/share/projects/erasmus1/triskele$ ls -l ~/test.tif
-rw-r--r-- 1 you usagers 23600822 nov.   7 15:35 /share/home/you/test.tif
```

Fig. 4 – gdal

— File transfer. Download the image at home. You can use any kind of tool (filezilla, putty, . . .). You can also use a command in terminal as "rsync dmis:test.tif .".

— Local visualization. We will use QGIS to show remote sensing image.

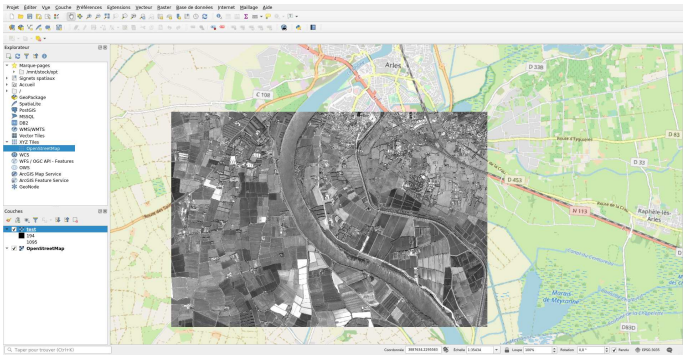


Fig. 5 – QGIS visualization

You can add openStreeMap layer by drag and drop item from "XYZ Tiles".

You can find an image:

— name IMG_SPOT7_P_201811080600105_SEN_SPOT7_20190318_14414914egoc4m9k1ge_1_R1C1.JP2

— width $38,593$

— height $43,773$

— pixels $1,689,331,389$

— bands 1

— values `UInt16`

Because this image is not free (licence is attached to it), you cannot uploaded it without explicit agreement.

We will do training with a small part of $2,000 \times 2,000$ pixels at the position $32,300$, $7,500$. So perform the commande:

```
$ channelGenerator IMG_SPOT7_P_201811080600105_SEN_\
  SPOT7_20190318_14414914egoc4m9k1ge_1_R1C1.JP2 \
  -c 0-* -x 32300 -y 7500 -w 2000 -h 2000 crop.tif
```

Fig. 6 – crop image

2 Trees

An image is usually represented by a matrix of pixels. This method shows pixels distribution, but does not consider the relation between different parts of the image.

Hierarchical image representation using component tree is an efficient method to represent an image in different scales and consider its vertical structure.

It has many applications such as image filtering, segmentation and boundary detection. The Table 1 shows some of them.

2.1 Hierarchical view

It is always difficult to introduction a large domain with many concepts without give incomplete definitions that will be precised further. So in this section we will give:

- Pragmatic approach as an averview of the domain
- Connected components the based concept for tree structure
- Connectivity necessary to define relation between pixels
- Max-tree overview as an example of tree
- Area attribute overview as an example of many others attributes
- Pruning operation as an operator performed in a tree.

2.2 Pragmatic approach to hierarchical representation (Isoline)

In this subsection we give some clues to understand a hierarchical representation. For that we say few words about:

- DSM (Digital Surface Model)
- Isoline (contour line)
- Slices of hill (tree filtering)

Illustrations in figure 7 come from https://gisgeography.com/dem-dsm-dtm-differences/and just give short explanations about LiDAR (LIght Detection And Ranging).

In a LiDAR system, pulses of light travel to the ground.

When the pulse of light bounces off its target and returns to the sensor, it gives the range (a variable distance) to the Earth.

The results obtained are distances. They can be used to define :

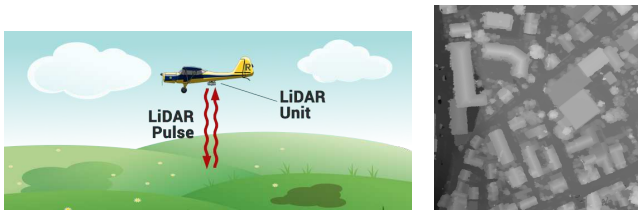
- DEM – Digital Elevation Models
- DSM – Digital Surface Models
- DTM – Digital Terrain Models and even
- TIN – Triangular Irregular Networks

With the Figure 7 (b) it is easy to imagine elevation of buildings.

Suppose we draw isolines as Figure 8 (a). We will obtain the graphical representation slope of each building. These isolines will be the nodes of our tree structures.

These nodes refer to pixels organized according values (elevation ou gray intensity).

These nodes are linked from children to parent until the ground is reached (called root).

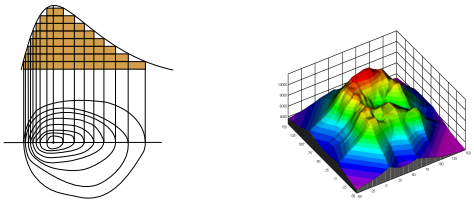


(a) LiDAR acquisition (b) DSM image

Fig. 7 – Use of LiDAR

In the Figure 8 (b) you can imagine nodes with differente colors (from purple to red). The base nodes cover all the image. they are stacked. Some are disconected on the top. That gives you the concept of tree.

Strangely when computer scientist describe a tree (data structure) the root is on the top of the figure and the leaves at the bottom. We will use the same convention for the rest of the document.



(a) Isoline (b) 3d model

Fig. 8 – Hierachical approach

Now you have the main idea. We will start to give more formal definition.

We consider an image compose by N pixels.

2.3 Connected components

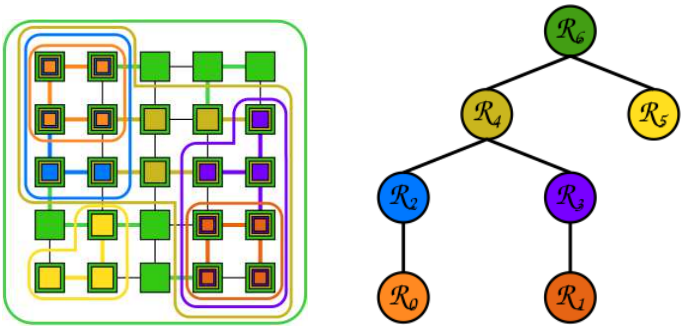
The general idea is to find connected components [4] in the image in different levels based on similarities between adjacent pixels as it is shown in Figure 9 (a).

These components can be arranged in a tree structure as depicted in Figure 9 (b). In this structure the leaves of the tree are the smallest components and

Application	Main strategy	Tree structure	Literature works
Hyperspectral image segmentation	Tree pruning Node merging based on spatial-spectral features	Binary partition tree	Valero et al., Vangazones et al., Tochon et al. (2010-2014)
PolSAR image filtering and segmentation	Tree pruning Node merging based on coherence matrix	Binary partition tree	Salembier et al., Alonso-Gonzalez et al. (2010-2014)
Lidar building modeling, segmentation	Tree pruning	Binary space partition tree	[27, 18]
Panchromatic image classification (first paper)	Attribute profiles	Min tree, max tree	[15]
Optical image classification	Attribute profiles on partition trees	Alpha-tree, omega-tree	[3]
High resolution optical image classification	Feature attribute profiles (extension of attribute profiles)	Min tree, max tree	[21]
Hyperspectral image classification	Extended attribute profiles (marginal approach = one tree per spectral band)	Min-tree, max-tree, tree of shapes	[16, 11, 5, 22], etc.
Hyperspectral image classification	Vector attribute profiles (vector strategy = one tree for the whole image)	Min-tree, max-tree	[1]
Highly-textured optical image classification	Attribute profiles on derived textures	Min-tree, max-tree	[25]
SAR image classification	Feature attribute profiles	Min-tree, max-tree	[30]
Land cover classification uing 3D Lidar data	Attribute profiles	Min-tree, max-tree	[7, 13, 12]
Rubble detection from VHR panchromatic images	Differential attribute profiles (DAP)	Min-tree, max-tree	[17]
Change detection from VHR images	Attribute profiles	Min-tree, max-tree	[9]
Anomaly detection from hyperspectral images	Attribute profiles	Min-tree, max-tree	[28]
Building extraction from Lidar data	Attribute profiles	Min-tree, max-tree	[14]
Image retrieval	Pattern Spectra	Min-tree, max-tree	[2]

Tab. 1 – Hierarchical representation uses

each component in lower level is a subset of a component in higher level. Finally the root of the tree corresponds to the entire image.



(a) In image (b) In tree

Fig. 9 – Connected components (same color for similar pixels in a component)

The number of component is between 1 (all pixels of a flat image connected to the root) to $N-1$ (each time we intruduce a pixel it modify the propery of a node and need to create a new one).

The number of connected components in practice are 20% to 30% of N.

Hierachical representation is compact structure. The properties of the use is:

— Speed up the browsing by small set of components

— Tree depth reduced bring information access to logarithmic complexity.

2.4 Connectivity

Two neighbors pixels are C4-connected if they are adjacent horizontally or vertically. Two neighbor pixels are C8-connected if they are adjacent horizontally, vertically or diagonally. After this step we have a list of indices of all connected pixels pairs.

But the connectivity can be defined more precisely. The Table 2 gives the base items to describ a connectivity. For instance, we can obtain C8 by addition of C4, C6P and C6N.

C4 gives neighbors vertically and horizontally. C8 gives all neighbors in two dimension. C6P gives 6 neighbors with positive diagonal (x+1, y+1). C6N gives 6 neighbors with negative diagonal (x+1, y-1).

Connectivity start with CT mean we consider time. The neighbors are taken in previous and next frame image.

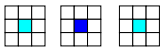
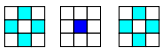
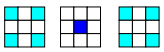
The connectivity impact:

— Shape smoothing. A lone pixel touching a large shape diagonally can be absorbed by it.

— Chaining effect risk. Two shapes can be crossed by a diagonal contact. If you are considering a chessboard, white cells can be merged into one connected component and black into another.

— Computation size impact. As we will see the building process of tree consider all neighbors. C8 compared to C4 double the number of neighbors and so the memory size used to create tree.

— Computation time impact. For the same reason, if we consider the double of items during the building process we double the time of building..

connectivity name	neighbours (⟶ time ⟶)
C4	
C6P	
C6N	
C8 = C4|C6P|C6N	
CT	
CTP	
CTN	

Tab. 2 – Axes of connectivity components

2.5 Max-tree overview

An image is usually represented by a matrix of pixels. This method shows pixels distribution, but does not consider the relation between different parts of the image.

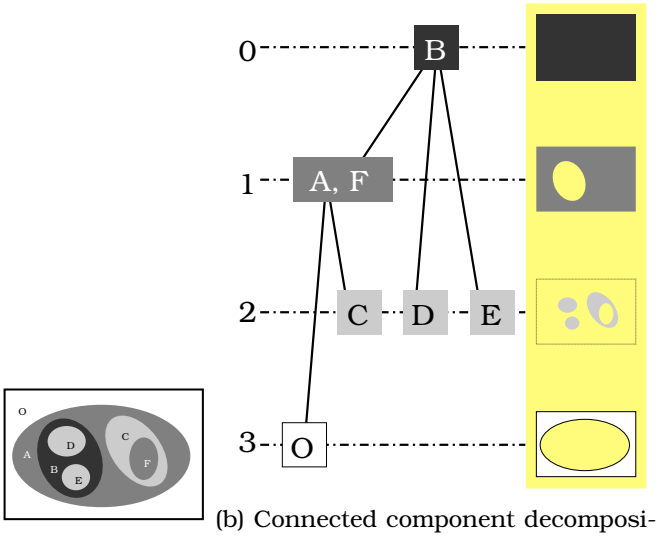
We will details this structure in section 4.3. Let us give the main propoerty at this moment:

— Max values in leaves. As explain, the root is on the top and leaves on the bottom. The leaves have the maximum values (for instance 255 for a byte, or maximum intensity if you prefer).

— Children can't be lower (hole in c-node, af-node merge). We couldn't have black component under light-gray component.

— Parents embed (cover) substructure. That mean all pixels referred by a children component must be included by its parent. That also mean the size of children always smaller than its parent..

This Figure only mention the connected components. We must be improved it with pixels. In that case pixels will be under the bottom line. All connected component must link these pixels. All pixels must be linked by one and only one component.



(a) Reference (b) Connected component decomposition

Fig. 10 – Max-tree

2.6 Attributes

Again, we will give here a short introduction of the attribute concept. We will explain about different kinds of node attributes with more details in chapter 7.

Attribute is often a scalar number that is computed for each tree node. It can be based on node shape (area, moment of inertia, aspect ratio ...) or based on node content (average, variance ...) or can be simply the node level in the tree structure.

Attributes are:

— Properties of nodes. This data can be inherited from its parent or synthesized from its children.

— Enhance node. Beyond the time saving due to the precomputation, it increases the information available at the node level.

— Dynamic production. In the tools we use, attributes can be chosen on the fly when needed.

2.6.1 Pruning

If we make trees, it is to perform operations on this structure. One of the most common is pruning the tree by removing the less significant branches. This could allow noise or minor elements to be removed in order to focus on the most significant elements.

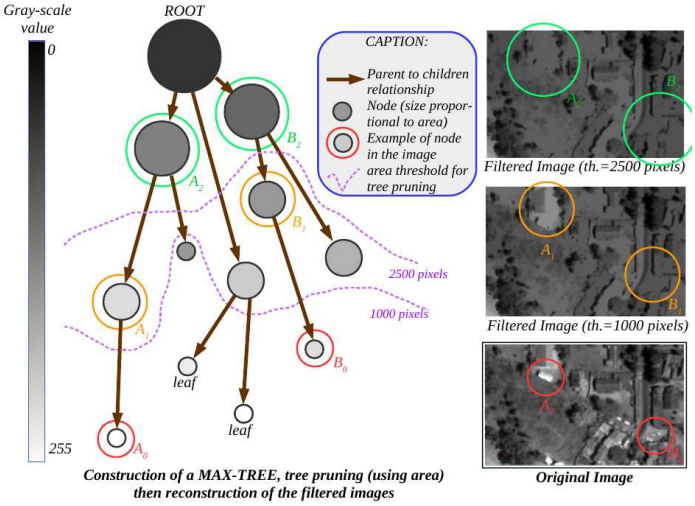
In the figure 11, the left part shows a max-tree. The smallest branches contain the brightest parts. Dark (like bright) pixels are all referenced by components.

The pruning operation is to replace all pixels referenced by the removed branch with the gray level of the parent node.

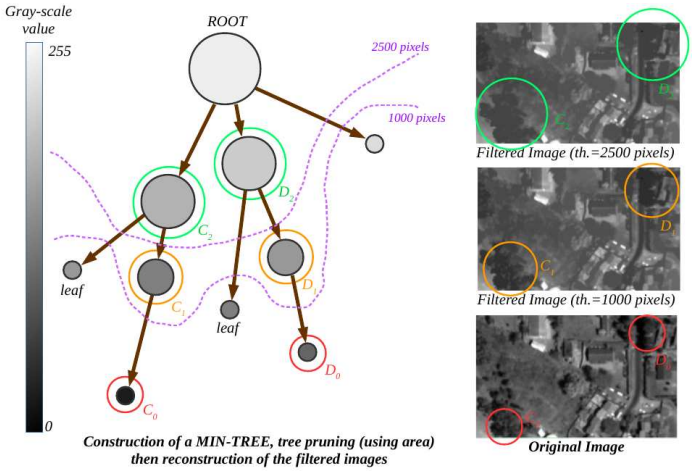
Removing the smallest elements is not necessarily equivalent as removing the brightest elements. If there is a shiny element left, it is large in size or it contains (topological neighbor) many brighter pixels.

The right part represents the opposite situation with a min-tree where the smallest branches containing the black parts.

When pruning, it is mainly the smallest dark parts that will disappear at first.



(a) Max-tree



(b) Min-tree

Fig. 11 – Pruning using area

2.6.2 Training

We are now going to perform the first remote sensing image processing.

```
$ /share/projects/erasmus1/triskele/channelGenerator
  /share/projects/erasmus1/triskele/crop.tif --time \
  test.tif -b 0 -t max -a area \
  --thresholds 10,100,1000
```

Considering the original image is 425 times greater, how many time it will take if do do the same process on the original one.

Please don't try it alltogether on the same sever "dmis".

It is possible to list the available arguments using the "--help".

```
$ /share/projects/erasmus1/triskele/channelGenerator --help
```

— Selection (crop, band). The first command is to reduce the scope of processing.

It must be considered that the image is a cube of data in 3 dimensions X, Y and the number of colors. You need at least select one band to be process ("-b") or to be copy ("-c").

```
$ /share/projects/erasmus1/triskele/channelGenerator \
  /share/projects/erasmus1/triskele/arles.tif \
  test.tif -w 1000 -h 1000 -c 0
```

— Tree type (max). The tree type is selected with "-t".

— Attribute cut (area). The attribute is selected with "-a".

— Threshold. The thresholds are dfine with "--thresholds" followed by list of values. You need to indicate tree type, attribute type and thresholds to obtain an output.

```
$ /share/projects/erasmus1/triskele/channelGenerator
  /share/projects/erasmus1/triskele/arles.tif \
  test.tif -w 1000 -h 1000 -b 0 -t max -a area \
  --thresholds 500,1000,5000
```

```
$ /share/projects/erasmus1/triskele/channelGenerator
  /share/projects/erasmus1/triskele/arles.tif \
  test.tif -w 1000 -h 1000 -b 0 -t min -a area \
  --thresholds 500,1000,5000
```

Then it is necessary to download the file "test.tif" on your PC and use QGIS for visualization.

3 Implementation

This section provides the keys to understanding the implementation of the algorithms used to build trees.

We will successively detail:

3.1 Implementation

— Complexity. It is an instrument to compare the efficiency of algorithms.

— Sort process. Which is a simple and instructive example of implementing complexity.

— Union-find. Which is the most efficient 70s algorithm for building a tree.

— Triskele. The tool we have developed in our research team to build trees.

3.2 Complexity overview

Complexity is the way to compare algorithms. This does not indicate precisely the execution time but rather an order of magnitude.

In fact the execution time is secondary, it is always possible to reduce this time by half by taking a more efficient processor or a higher execution frequency.

Here are the important elements for complexity:

— $0(f(n))$. Complexity is characterized by a function. In practice we will say that an algorithm is comparable to the complexity of a characteristic function. The factors are completely ignored, because once again, it suffices to change the technical characteristics of a processor to remedy it.

— Find loop. As a first approximation, it can look for loops in the algorithm. This gives information which may be imprecise but nevertheless significant. This is what we will do in the case of a sorting algorithm.

— Size of image (square grow). The execution time obviously depends on the size of the data processed. What concerns us is the evolution of time according to the evolution of the size of the input set.

— Algorithms . The known algorithm families are:

  — $log(n)$ *dichotomous search*

  — n *search*

  — $n \cdot log(n)$ *often fast*

  — n^2 *slow*

  — n^3

  — $poly(n)$ ...

In case of $1ms$ per instruction we get the following runtime times:

	10^2	100^2	$1\ 000^2$	$10\ 000^2$
$log(n)$	$.002''$	$.004''$	$.006''$	$.008''$
n	$.1''$	$10''$	$16'$	$27h$
$n \cdot log(n)$	$.2''$	$40''$	$1h40$	$9d$
n^2	$10''$	$1d$	$31y$	$317x10^3y$
n^3	$16'$	$31y$	$317x10^6y$	$31x10^{12}y$

Tab. 3 – Common complexities trends

Under these conditions, you understand that it is crucial to want to reduce the complexity of an algorithm regardless of the size of the images that we have to process.

Note, if a sorting algorithm is $O(n)$, it means that when we examine an item (or twice), we are able to determine exactly where it will be placed regardless of the items that come with it. It is divination.

Current best algorithms is $0(n \cdot log(n))$.

3.3 Sort process

Here we present a sorting algorithm.

Quicksort features:

— Most popular. Because it can be applied to any circumstance.

— $O(n \cdot log(n))$. Its complexity is among the best that can be done.

— Define pivot. The first step is to choose an element at random (at one end).

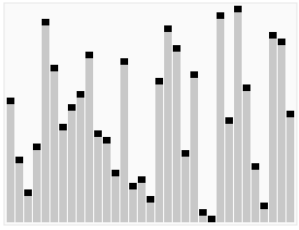
— Split. Then separate the whole into two subsets: the lower and the upper. To do this we consider the limits and each time either we exchange with the max and we increment the lower limit, or in exchange with the min and decrease the upper limit.

— Swap. When the sets are well splited, we swap the pivot with the middle.

— Recursive on subset. You just have to perform the same treatment with each of the two subsets. As at each step the whole is divided into two, the logarithm coefficient appears.

Figure 12 (a) shows an animation of quicksort.

You can count quicksort and counting sort at the same time with the animation Figure 12 (b). The magic thing will be explained in the next section.



(a) Quicksort

(b) Counting sort

Fig. 12 – Efficient sorting algorithms

Sometimes computing can be magic.

Counting sort can be described as:

— 3 pass. The first create a histogram by counting the property to sort. The second is an accumulation of histograms to determine the position of each slice of elements. The third is to correctly place the element where it should be.

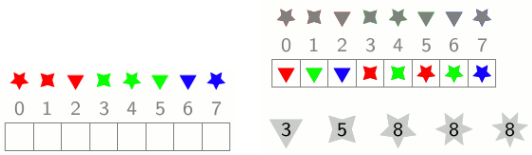
— Unsuitable with floating point values. The hypothesis is to assume that we can make a histogram. However, this is not possible if the set of values is infinite.

— Values cardinality close to the size of set. This technique is only possible in practice if the range of values is comparable to the number of items to be sorted.

The favorable case is when the range of values is only a few thousand, even though the number of items to be sorted is several billion.

— Keep previous sort. It should be noted that even if one carries out consecutive sorts according to several criteria, the preceding partial orders are preserved.

For example here the polynomials have been sorted by color (red, green, blue) before being sorted according to the number of branches.



(a) Initial stage (b) Final stage

Fig. 13 – Counting sort example

A dynamic animation can be found on https://triskele-wiki.kaz.bzh/algo/sort/start.

3.4 Tarjan's Union-find algorithm (1970)

Here we will describe briefly the algorithm used for creating a tree, in order to understand the main steps involved, and the reasons why it has a very low computational complexity.

We should mention that our algorithm is inspired by the concept of Union-find described by Tarjan [29].

First we explain the construction of max-tree. Then we will describe the difference for the construction of other types of trees (min-tree, tree-of-shape, α-tree, ω-tree).

The different steps to build a max-tree are as the following:

— Make an adjacency list of all connections between two pixels that are connected together in C4 or C8 connectivity.

→ This step complexity is $O(N)$, where N is the number of pixels of the image.

— Find minimum gray scale value for each pixels pair in the adjacency list. Now we have a list of triples. Each triple consists of indices of two neighbor pixels and their minimum gray value that we call adjacency value. This list actually helps to find the leaves of the tree and is used in next step to merge different pixels together. When we look on the minimum gray level, the value of parent node is always less than its children that is the desired property of max-tree.

→ This step complexity is $O(N)$

— Sort the adjacency list according to adjacency value. Since in a max-tree we first merge the maximum pixels, we sort adjacency list in descending order to have the largest gray scales at the beginning of the list.

We use an efficient sorting algorithm that is called "counting sort". When there are limited number of distinct values to be sorted, the complexity of counting sort is $O(N)$ that is less than other sorting algorithms. This is the case of pixels values where we have 256 distinct values for a 8-bit gray scale image.

→ This step complexity is $O(N)$

— Start from the beginning of the sorted adjacency list and merge each pair of pixels together. This will make the first level of the tree with new adjacency list now between tree nodes. Then the process of merging is repeated until the root of the tree is reached.

The complexity of this algorithm is generally proportional to the number of tree levels which is actually the number of gray levels of the image. But we used an optimization in algorithm that reduces the complexity to a constant value.

→ This step complexity is constant.

Now if we sum up all the different previous steps, we can conclude that for the construction of a max-tree

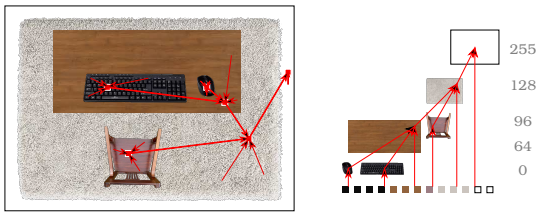
the overall complexity is $O(N)$.

Algorithm 1 *UnionFind* algorithm

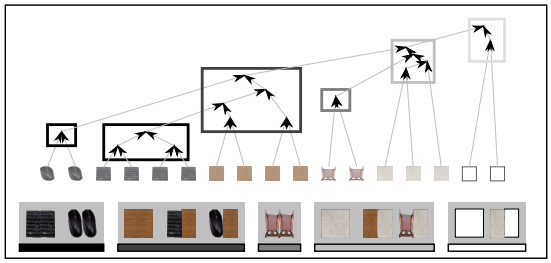
```
function findRoot(zpar, x)
    if zpar(x) = x then
        return x
    else
        zpar(x) ← findRoot(zpar, zpar(x))
        return zpar(x)

function unionFind(R)
    for all p do
        zpar(x) ← undef
    for i ← N − 1 to 0 do
        p ← R[i], parent(p) ← p, zpar(x) ← p
        for all n ∈ N(p) do
            if then zpar(n) ≠ undef
                r ← findRoot(zpar(x), n)
                if r ≠ p then
                    parent(r) ← p, zpar(r) ← p
        return parent
```

3.5 Trikele (min-tree case)

Let's do a first application of the previous algorithm on an example of a toy in Figure 14.



(a) Image components



(b) Building tree steps

Fig. 14 – Image with symbolic components

We consider on the representation composed of homogeneous pixel object. This is incorrect. But as an approximation.

We will say that the values of the pixels of the ground are the brightest, those of the carpet less, those of the chair and the table even less. The pixels on the keyboard and mouse are the darkest.

We define a pair (neighbors) of pixels. We sort them more and more by considering the maximum of the value of the pair.

we take each pair in sequence and build the hierarchical structure. Each time we change the set of pairs of commit shapes on the image (mouse and keyboard, table, chair, ...).

3.6 Triskele

Now let's go into more detail with Triskele.

First of all, we use the fundamental properties of hierarchical data representation. Consider a gray scale image (left side of Figure 15a). This image is a matrix of pixels. We can build different types of trees (min-tree, max-tree, ...) to manage the image. We choose to build a max-tree (right side of figure 15a).

This tree is a hierarchical representation. All pixels have parent. These parents are called nodes. All nodes (except the root) also have a parent node. This defines an inclusion tree. Because we choose a max-tree, the nodes are organized from light gray (low level of the tree) to dark gray (high level). All nodes can have properties. We represent them with colored bullets in Figure 15a.

Triskele uses a compact representation of this data. The tree is defined only with an array of parents (see Figure **??**). All pixels and nodes are associated with a unique index. The indexes under "pixel cardinality" refer to the pixels from the first row to the last row of the image. Indexes with higher "cardinality of pixels", refer to tree nodes. Thanks to this building process, all nodes are sorted according to their level. So the light-gray nodes are on the right and the darker-gray ones are on the left.

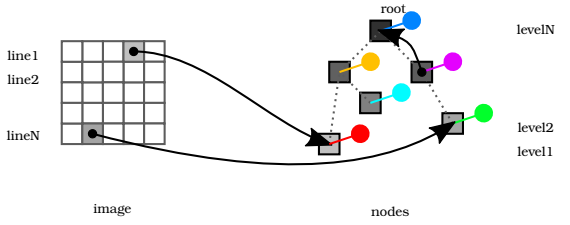
Attributes are only defined for nodes. The attributes can be simple scalar (surface, perimeter, gray value) or multivalued (centroid, bounding box), and they are also stored in another array. The position of an attribute in an array gives the position of the associated node. Here are our efficient calculations of new attributes.

Let's highlight the main properties of Triskele:

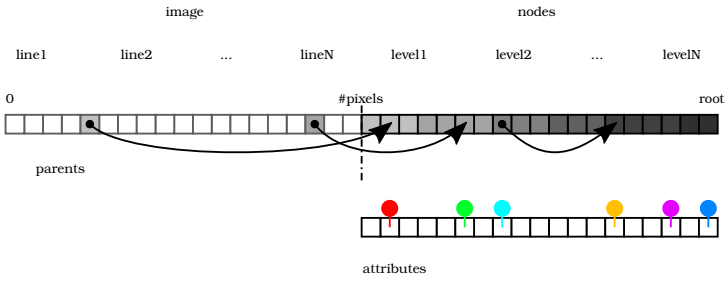
— Minimum memory. The cells of the table contain only the links necessary for chaining.

— Sorted weight. The connected components are sorted by weight.

— Scalable attributes. It is possible to dynamically add attributes by juxtaposing an additional table.



(a) Model of parents link



(b) Array implementation of parents link

Fig. 15 – Triskele parents link implementation

We come to describe a real case of an image of 16 pixels (4 by 4).

Figure 16 (a) represents all the data structure handled by the program. At the top left corner we find the already filled pixel matrix. The matrix just below (which we will call weight) will receive the value of the components. Just to their right are two matrices (will we will call parents) will receive the parents of the components and those of the pixels respectively. The last essential matrix is called count and will receive the number of children pointing to the component (determined according to its position in the matrix).

We then find secondary matrices.

Edges is used to receive the neighbor couple (a and b) with the weight. In the case of a min-tree, we will take the maximum value of them that we will sort in increasing order.

Position is only there to provide us with a quick way to find out the location of a and b and is not used by the algorithm.

Short-cut is an optimization to save time and record the local root during the build process.

Children will receive the indexes identifying the children.

There are two rounds. The first to link the pixels to their parents up to the root and we obtain the Figure 16 (b). The second is to do the reverse chaining from the root to the children and we obtain the Figure 16 (c).

For the first round, we start by sorting the edges (neighbors) in ascending order. Then for each edge we search for the corresponding position in the pixel matrix then we carry out the chaining (possible by creating a component if necessary). It is remarkable that each edge is evaluated once and only once. The processing was straightforward so the complexity is well linear: $O(n)$.

The second round begins where the previous one ends. We start by perform a cumulative sum of all the counters of children. We add zero in front of these values. This provides exactly the intervals of position the indexes to the children. From 0 to 2 (not include) red children. From 2 to 4 (not include) yellow children. From 4 to 10 (not include) green children, and so on. Note that index children less than 16 refer to one pixel and upper a component.

Algorithm 2 *children* algorithm

function BUILDCHILDREN ($nbComp$, $parents$, $childCount$, $children$)
  for $i \leftarrow 0$ to $nbComp$ **do**
    if $parents(i) = root$ **then**
      $next i$
    $children[childCount[parents[i]]++] \leftarrow i$

You can remember Triskele:

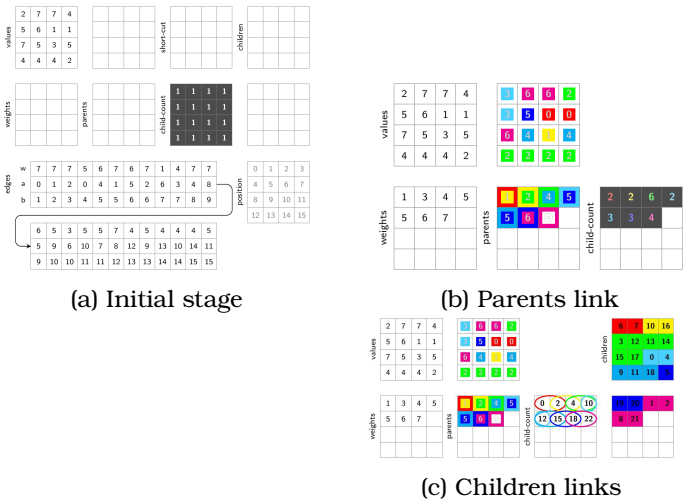
— Based on Union-Find. It give us a linear complexity.

— Custumized by neighbors distance. We just change the distance function between neighbors to change the tree structure.

— Generic engine for min, max, med, tos, α .... The previous property gives us the capability to enhance our tool.

— Visite one time each neighbors to build tree. That the evidence we get linear complexity in the first round.

— Visite one time each components to reverse links. That the evidence we get linear complexity in the second round.



(a) Initial stage

(b) Parents link

(c) Children links

Fig. 16 – Triskele building tree algorithm

A dynamic animation can be found on https://triskele-wiki.kaz.bzh/algo/build-tree/start.

3.7 Training

The "--timeFlag" option allows us to know the execution time of the program.

The "--coreCount" option set the number of core used to launch the program.

By changing the size input image we can see the linear progression of the execution time.

Try these commands.

```
$ /share/projects/erasmus1/triskele/channelGenerator \
/share/projects/erasmus1/triskele/arles.tif \
test.tif -b 0 -t max -a area --thresholds 1 \
--time -w 1000 -h 1000 --core 2

$ /share/projects/erasmus1/triskele/channelGenerator \
/share/projects/erasmus1/triskele/arles.tif \
test.tif -b 0 -t max -a area --thresholds 1 \
--time -w 1000 -h 2000 --core 4
```

— Display calculation time. Display the time information.

— Change crop size. Display the time information. Notice, if you double the width and height at the same time, you are squaring the number of pixels.

— Change the number of core. Display the time information. Because not the whole algorithm can be parallelized, double the cores does not halve the execution time.

4 Inclusion trees

4.1 Inclusion trees

In this section we introduce inclusion tree structures:

- Max-tree
- Min-tree
- Med-tree
- ToS

4.2 Inclusion/Partition

The connected components in each level of the tree do not necessarily cover all the image. Based on their coverage, two types of trees are defined that are called partitioning tree and inclusion tree. As depicted in Figure 17 in partitioning tree the entire image can be reconstructed from the tree nodes at each level. But in inclusion tree the children nodes are separated connected components that don't cover the original image completely.

The leaves of the inclusion tree consist of small connected components corresponding to image minima or maxima. These minima or maxima grow in higher levels by adding new pixels until the root of the tree which represents the whole image. Three types of inclusion tree in literature are Max-tree, Min-tree and Tree of Shapes.

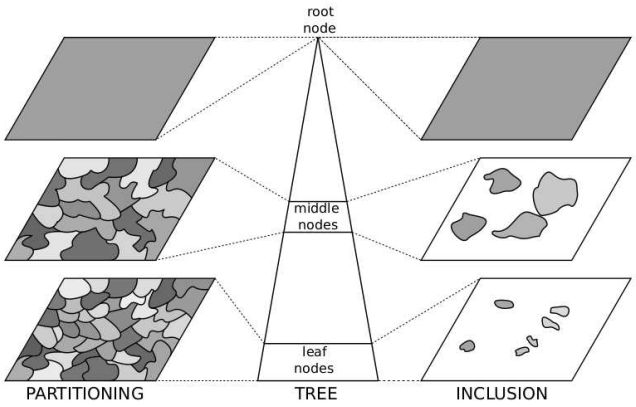


Fig. 17 – Inclusion partition trees

In contrast the nodes of the partitioning tree at each level form a partition of the image. They do not intersect and their union constructs the whole image. The components include similar pixels. The leaves of the tree is the finest partitions and they merge together to form coarser partitions in higher levels. Three types of inclusion tree in literature are alpha-tree, omega-tree and binary partition tree.

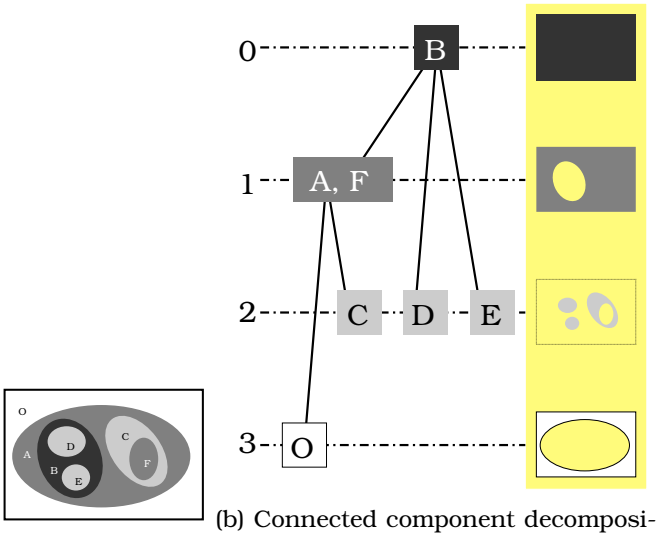
The main concepts are:

- The root gives the whole image
- Partition
  - Nodes cover all
  - Merged nodes
- Inclusion
  - Nodes associated with weight
  - Nodes include sub weights

4.3 Max-tree

Max-tree is an inclusion tree whose leaves are local maxima (bright pixels) in the image. For a binary image max-tree has only two levels consisting of a root which is connected to all bright connected components.

In case of a gray scale image it can be converted to a binary image by putting a threshold on its gray level. The root of the tree is a connected component that is the result of thresholding the image at lowest gray level. By increasing the threshold, smaller separated connected components appear in the image. Each one of these connected components builds a node of the tree that is connected to its superset connected component in previous level. This procedure continues until the highest gray level of the image.



(a) Reference (b) Connected component decomposition

Fig. 18 – Max-tree

Figure 18 (a) shows a toy example of gray scale image. We designed this image in order to represent different variations in gray scale images that allow us to compare different tree types. In this synthetic image upper-left corner is local maximum, upper-right corner is local minimum and lower corners show some intensity variations.

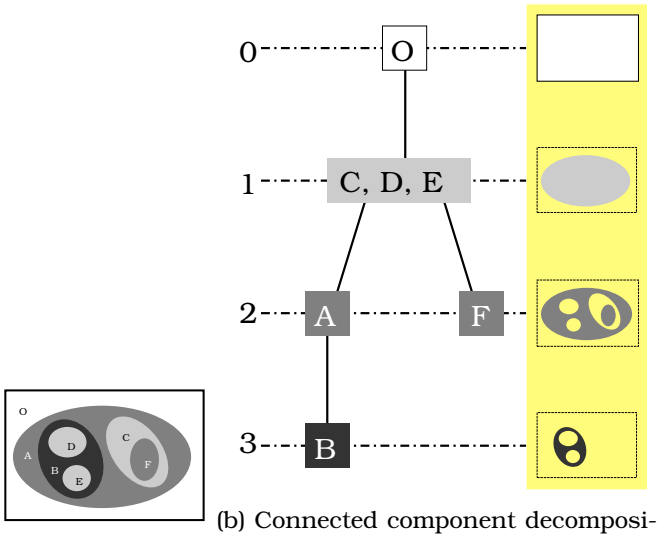
Figure 18 (b) shows the max-tree of the original image. Main stream on left branch of tree starts with combining the bright upper-left corner of the image. Then upper-right and lower-left corners connect to it. Also lower-right corner of the image is decomposed in the right branch of the tree and connects to the main stream.

As can be seen in Figure 18 (b) the bright components of the image appear in max-tree nodes. However, we cannot see the dark components of the image between max-tree nodes. Therefore if the objects in the image are bright, max-tree structure is useful to detect them.

— Max values in leaves
(white = max light on pixels)

4.4 Min-tree

Min-tree is another type of inclusion tree that its leaves are local minima (dark pixels) in the image. Min-tree is the dual form of max-tree. That means min-tree is the same as max-tree on the negative image.



(a) Reference (b) Connected component decomposition

Fig. 19 – Min-tree

Figure 19 (b) show the min-tree of the original image in Figure 19 (a). Here the main stream (left tree branch) starts with dark (minimum) pixels in upper-right corner of the image and then connects to bright pixels in upper-left corner of the image. Similar to max-tree the lower-right corner of the image is decomposed in the right branch of the min-tree and connects to the main stream.

We see in Figure 19 (b) that dark components of the image appear in min-tree nodes. Therefore, min-tree is a favorite structure to detect dark objects on a bright background. We also observe that none of the min-tree or max-tree can discriminate lower-left corner of the original image. In order to address this problem, median-tree is introduced that we explain in next section.

— Min values in leaves
(black = min light on pixels)

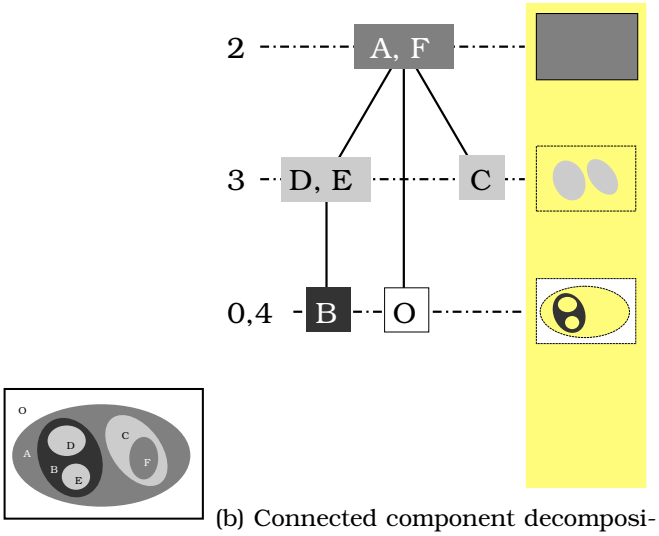
4.5 Med-tree

As it is explained in previous sections, max-tree and min-tree are not able to detect dark and bright parts of the image simultaneously. In order to solve this problem we introduce median-tree that can be considered as a combination of max-tree and min-tree.

In order to build median-tree the original image is first converted to an intermediate image. If I is the gray level of the original image, the median image is created by $I_m = |I - median(I)|$. All local extrema in original image will be converted to local maxima in the new image. Then a max-tree is built based on the new image.

Figure 20 (a) and (b) show the original image and the intermediate median image respectively. Median-tree is built by applying max-tree algorithm on the median image. Median-tree is able to detect both bright and dark components in the original image as depicted in 20 (c).

The building process of a median-tree needs to step:



(a) Reference (b) Connected component decomposition

Fig. 20 – Med-tree

4.6 Tree of Shape

Median-tree is an efficient representation to detect dark and bright components in the image. However, median-tree is essentially a max-tree and still is not able to detect the components with median intensity in the original image.

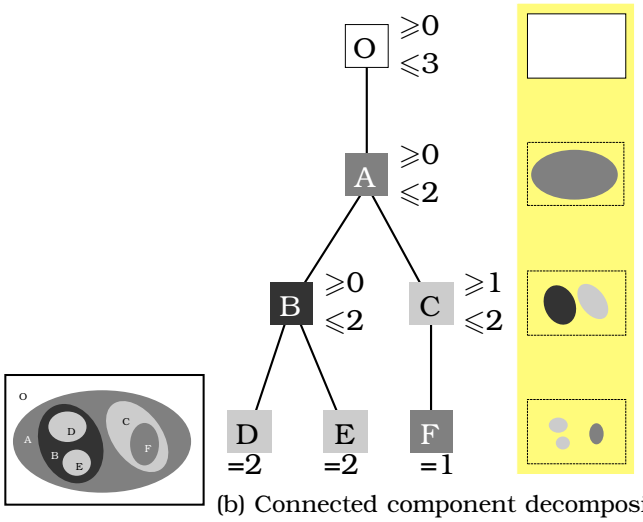
Tree of Shapes (ToS) is another inclusion tree that combines min-tree and max-tree. The leaves of ToS are local extrema that are not necessarily minimum or maximum intensities of the image. We mentioned that the nodes of max-tree and min-tree are the connected components that are obtained by putting thresholds on gray levels of the original image. The nodes of tree-of-shape are actually the cavities of the max-tree and min-tree nodes [10]. Nodes of ToS are complete shapes without any hole inside. Each pair

of ToS nodes are either separate or one of them is a subset of other one. The ToS is a self-dual structure which means that the negative image produces the same ToS.

Figure 21 shows a toy image with its Tree-of-Shape. We observe that the extrema components D, E and F that are neither maximum nor minimum of the image, appear as ToS leaves. Other nodes in ToS are complete and without holes. For example while the ellipse C with its hole F is a node in max-tree, it can be distinguished as a complete shape in ToS nodes.

— Find contour

— Based on priority queues



(a) Reference (b) Connected component decomposition

Fig. 21 – Tree of Shape

You already know the options. We are going to work on the choice of different type of tree with the same projection thresholds (pruning).

4.7 Training

— Tree type (max, min, med). See the impact of changing the tree type.

5 Partition trees

5.1 Partition trees

In this section, we will only see three types of partition trees:

— Binary Partition Tree

— α-tree

— ω-tree

5.2 Binary Partition Tree (BPT)

Binary Partition Tree (BPT)[31, 26] is a partitioning tree. The leaves of the tree is the initial predefined partition of the image that can be pixels, flat zones or any other fine partition. BPT is constructed by repeatedly merging two similar adjacent regions together. It needs the definition of region model and merging criterion. Region model is the intensity that we assign to each region and can be the average or median of all pixels intensities inside a region. Merging criterion is a dissimilarity measure between two regions that can be defined based on intensity or shape of the regions.

Figure 23 (c) shows a binary partition tree generated from the original image in Figure 23 (a).

Here the region model is the constant gray level and merging criteria is the size of regions. The leaves of the tree are all flat regions or connected components of the original image. In each level two adjacent regions with smallest area are merged together. Gray level of the parent is equal to gray level of the bigger child or the average gray level of two children in case of equal size.

Figure 23 (b) gives a segmentation take from the hierarchical structure.

BPT is the most flexible type of tree structure since different structures can be generated using different definitions of region model and merging criterion.

The main items are:

— Repeatedly merging two similar adjacent regions

— Region model (gray level)

— Merging criterion (intensity or shape of the regions)

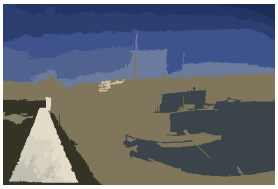
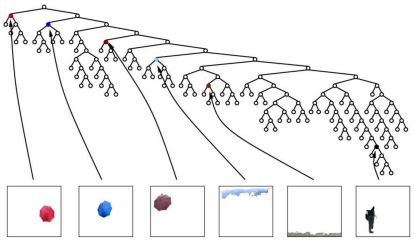


Fig. 22 – BPT visual example

(a) Reference

(b) Segmentation



(c) Tree structure

Fig. 23 – Binary partition tree

5.3 α-tree (alpha-tree)

α-Tree is another partitioning tree based on similarity of the adjacent pixels. α-connected component is defined as a region in the image that the distance between every two adjacent pixels is not more than α. Different values of α produce different levels of partition on the image.

Figure 26 (b) shows α-tree of the original image in Figure 26 (a). For this single channel image the distance between two pixels is simply the difference of their intensities. Therefore, using $\alpha = 0$ the image is partitioned to its flat zones that form the tree leaves. The partitions of the image grow with increasing values of α until the entire image is partitioned to a single component that is the tree root.

Each child component in the lower level is a subset of a parent component in higher level. The α-tree is the hierarchy of all α-components in different levels as shown in Figure **??**.

α-tree is suitable for image segmentation in different scales. However, since it only restricts the adjacent pixels, a long sequence of similar pixels may connect together to produce a very large component with high intensities variations.

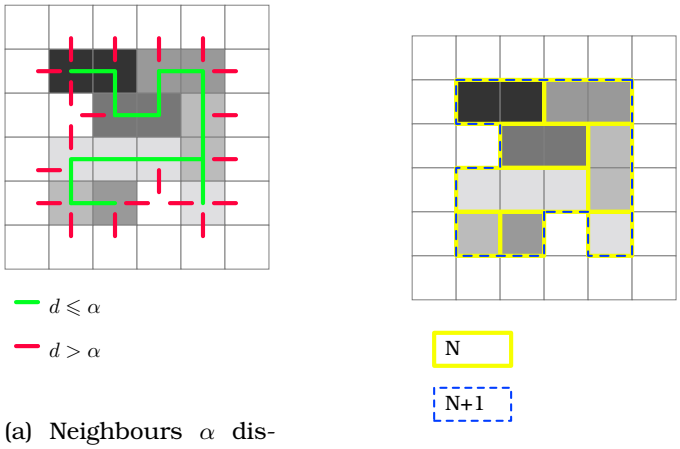
This problem that is called chaining effect is the result of transition zone in the image where adjacent pixels have almost similar intensities, but far pixels are very different. An example can be seen Figure 26 (b) at level $\alpha = 2$ when very different pixels are connected together to produce a large tree node.

To define a node, we can two ways (Figure **??** (a)):

— Path where $d \leqslant \alpha$ can connect all embeded pixels or

— Boundary where $d > \alpha$ that isolate the node from all others pixels.

The Figure 24 (b) shows node $\alpha\ N+1$ is fully composed by sub nodes $\alpha\ N$.



(a) Neighbours α distance

(b) Nodes inclusion

Fig. 24 – α-tree properties

It is not easy to represent tree with a 2 dimesional document. The Figure 25 gives:

— 3 different texture areas in sky a quasi flat zone, in sea surrounded by boundary made with objects and in a wall with granular texture.

— $\Omega_x^a = |P^{\alpha=0}(x)|, |P^{\alpha=1}(x)|, |P^{\alpha=2}(x)|, \ldots, |P^{\alpha_{max}}(x)|$ the formula to define curves for each area.

— Curves: $area/\alpha$ the axes of each curves that give the number of pixels of nodes according the α level.

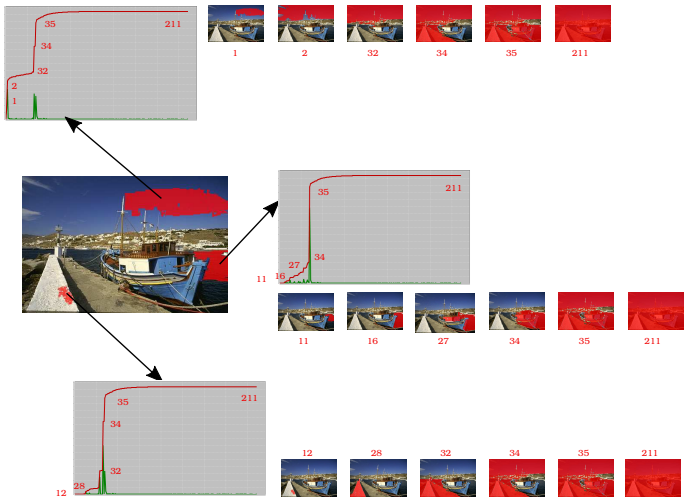


Fig. 25 – Size evolution curves

5.4 ω-tree (omega-tree)

ω-tree is a partitioning tree that tries to solve the chaining effect problem in the α-tree. While α restricts the difference only for adjacent pixels, ω restricts the maximum difference between lowest and highest gray level inside each connected component. In other words α is local constraint and ω is global constraint.

Nodes of the ω-tree are a subset of nodes of α-tree. It means that ω-tree is built by removing some of the α-tree nodes that don't satisfy the global constraint. ω-component is defined as the largest α-component that the maximum intensity difference between its pixels is equal to ω. Figure **??** shows the ω-tree for the same image in Figure **??**. If a parent node and a child node in α-tree have the same ω, the child node is removed from the tree to create ω-tree. Here, four nodes with $\alpha = 2, 5, 6, 7$ are removed from Figure **??** because all of them have the same $\omega = 10$ with tree root.

The ω-tree has less nodes and more levels than α-tree ($\omega_{root} \geq \alpha_{root}$). Since ω-components are a subset of α-components, the ω-tree cannot solve the problem of chaining effect completely.

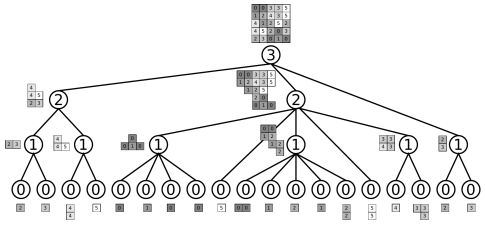
Both α-tree and ω-tree have self-dual property. It means that the constructed trees from original and negative images are the same; because similarities of the pixels don't change in negative image. The α-tree and ω-tree can be extended to multi-channel images by defining a similarity measure between vectors.

— Chaining effect of α-tree

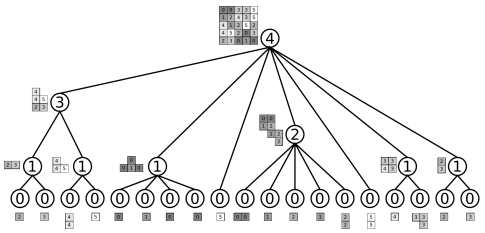
— $\Delta(min, max)$ limits for ω-tree

0	0	3	3	5
1	2	4	3	5
4	1	2	5	2
4	5	2	0	3
2	3	0	1	0

(a) Reference



(b) α-tree



(c) ω-tree

Fig. 26 – ω-tree derived from α-tree

6 Dimensions

6.1 Dimensions

In the previous sections we explain the tree structure base on 2D images. Thanks to the connectivity we are able to apply the concept on any others dimension. For instance a time series of 3D images from brain activity give us an 4D images. Today we have not managed these kind of studies. So we limit the explanation to 3D images.

— Spectral stack of bands

— Time stack of dates

— Issues artifacts side effect

6.2 Time

Consider a gray-scale image from landscape includes some fields on Monday and all others days of the week. That is a time series. If we stack them we obtain a sort of cube. A field takes volume and can be see as a building. Merge or split fields suggest building start after the bottom and stop before the stop.

To build tree from these data, we can adopt different strategy.

— Split bands

— Merge bands to get cube

— Mixed (time + spectral) Some times we get multi-spectral remote sensing images from a time series. They are stacked in the same file. The stack of bands can order by dates then spectral or spectral then dates.

  To correctly managed them we need 3 informations:

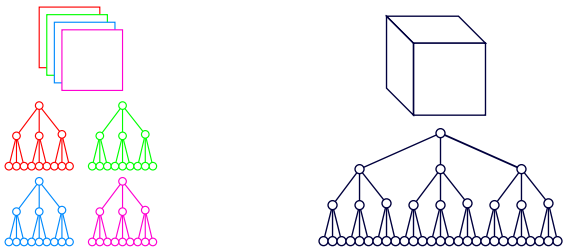
  — Number of dates

  — Number of spectral bands

  — Order date or spectral first

With a data cube we can decided to simply get a tree per band (date or spectral) or create a tree where each leaves referred a voxel (3D pixel).

In the of mixed bands (date and spectral) we prefer not build a unique tree, because the distance to link pixels are not the same if we consider two dates of the same spectral band (time distance) rather than pixels from two bands (see sub-section Spectral).



(a) 2d images (b) 3d image

Fig. 27 – Cube tree

6.3 Spectral

To build a hierarchical structure image from pixels value, we need to define a way to compare them. That's easy with gray-scale value (natural scalar order). It's not the case with multi-spectral image, where pixels owned one value per band (i.e. vextor values).

— Define vector ordering function There is no simple way to order vector values. We have t define a similarity function.

— lexicographic The first approach is to sort value according the rank in the vector.

— Define distance A more appropriate approach is to define metric according the study.

The most popular metric functions are named norm.

— The generic definition.

  L_p: $||x||_p = \sqrt[p]{\sum_i |x_i|^p}$

— L_0: $||x||_0 = \#(i|x_i \neq 0)$

— Manhattan distance (a diamon)

  L_1: $||x||_1 = \sum_i |x_i|$

— Euclidean distance (a circle)

  L_2: $||x||_2 = \sqrt[p]{\sum_i |x_i|^2}$

— Chebyshev distance (a square)

  L_∞: $||x||_\infty = \max_i(|x_i|)$

6.4 Issue

A similarity function to order vectors is not unique, we may have many "path" of ordering.

The concrete impact appears when we perform some "opening" and "closing" operators on an image.

The result on a gray-scale image is "predictable". But with multi-spectral images it depend on the "path".

Convert to gray-scale then opening/closing it not equivalent of opening/closing then convert to gray-scale.

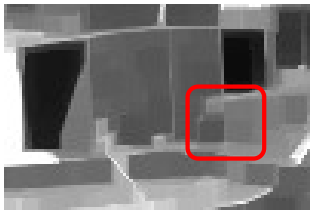
Somme artifacts may appears according the fact bands changes are not synchronized.

(a) Gray-scale

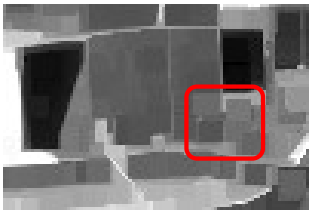
(b) Color

(c) 3 opens / 3 closes

(d) 3 opens / 3 closes



(e) Shape stability



(f) Color artifact

Fig. 28 – Divergence caused by bands interaction

7 Attributes

Attribute is often a scalar number that is computed for each tree node. It can be based on node shape (area, moment of inertia, aspect ratio ...) or based on node content (average, variance ...) or can be simply the node level in the tree structure.

An attribute is called "increasing" if its value for the parent node is always greater than the child node.

For example number of node pixels (area) is an increasing attribute but pixels variation is not like that.

7.1 Attribute

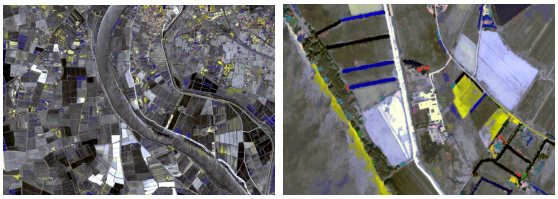
- Area
- Perimeter
- ZLength ($\Delta(BoundingBox.z)$)
- Compactness ($area/perimeter^2$)
- Complexity ($perimeter/area$)
- Simplicity ($area/perimeter$)
- Rectangularity ($area/BoundingBoxArea$)
- Weight (i.e. gray-scale level)
- SD ($StandardDeviation^2$ based on mean gray value of nodes)
- SDW ($StandardDeviation^2$ based on weight of nodes)
- MoI (Moment of Inertia based on centroid)
- Hidden attributes
  - Bounding box
  - Min
  - Max
  - Mean
  - Centroid

7.2 Area

It's the most common and efficient attribute. Easily understanding and computing, it refers the number of pixels attach to a region (a tree node).

The value of a pixel is at least 1 (may be more if we consider flat zone). The value of the root is always the size of the image.

The function describes the value from a pixel to the root is regularly growing function.



(a) Arles (b) Center

Fig. 29 – AP / area (10, 100, 5000)

The computation of these attributes is linear and even be paralyzed for all nodes with the same level.

Algorithm 3 Compute $area$

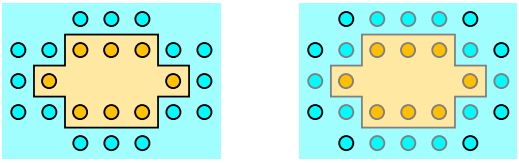
for $c \in nodes$ **do**
  $area_c \leftarrow 0$
for $c \in pixels$ **do**
  $p \leftarrow parent(c)$
  if $p \neq root$ **then**
    $area_p \leftarrow area_p + 1$
for $c \in nodes$ **do**
  $p \leftarrow parent(c)$
  if $p \neq root$ **then**
    $area_p \leftarrow area_p + area_c$

7.3 Perimeter

The perimeter is the set of pixels that define the border of a node. It depends on the connectivity. Figure 30 illustrates that the higher connectivity (C8) increases the number of pixels involved in perimeter. Note that, even if two areas share the same border, the perimeter depends on their convex (or concave) sides. The perimeter of an embedded node (orange) is always shorter than the enclosing node (blue).

If we study the function gives the perimeter from a pixel to the root, we will observe the function is not monotonous. It starts with the value equal to the connectivity (for one pixel) and ends with the double of the sum of width and height (for the root). This value can increase with the complexity of parent node, or decrease with the simplicity of parent node.

The attribute is a good example to see how to take into account the array data structure that implements the tree structure.



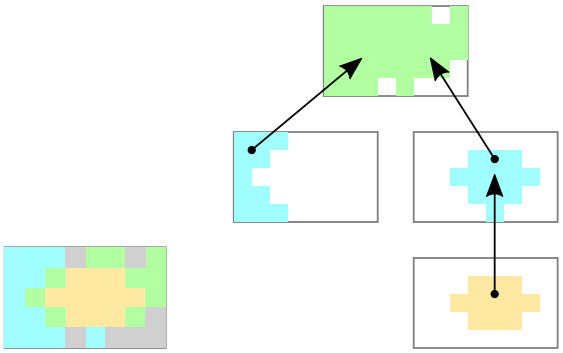
(a) C4 (b) C8

Fig. 30 – Connectivity impact on perimeter

A simple approach to finding the perimeter could be to create a parent map. In this case, we replace the gray level of each pixel with the index value of its parent. We must consider in this map all the pairs of neighbors (depending on the connectivity) to be sure not to miss any pixel border. On the other hand, we have to be sure not to consider a pixel twice to play this role. This simple approach requires an additional map. Its size is the same as the number of pixels in the original image. With huge images (Giga pixels), an index of parent needs at least 32 bits. The complexity of such algorithm is also inappropriate for handling huge images.

Our approach starts with morphological observations. Consider a simple image (a) with only 3 nodes

in Figure 31. We assume it is a gray-scale image, but colored here to highlight the nodes. We build an inclusion tree (e.g. max-tree). The orange pixels have the max value, the green ones have the min values and the blue ones in the middle. We also consider some no-data pixels.



(a) Image (b) Tree

Fig. 31 – Perimeter data structure

The hierarchical representation is depicted in (b) in the same figure. Figure 32 represents the pixel borders with C4 connectivity. Pay attention on the fact when a pixel becomes a perimeter pixel in a node, and when it loses this property in a parent node. We now develop these different cases, represented by 6 numbered pixels from the figure:

① is orange border with blue node but no longer in parent nodes.

② is orange border with blue node and blue border with green node, but no longer in parent nodes.

③ is border everywhere with no-data.

④ is blue border with green node but no longer in parent nodes.

⑤ is border with frame image or no-data everywhere since it appear in blue node.

⑥ is border with frame image or no-data everywhere since it appears in green node.

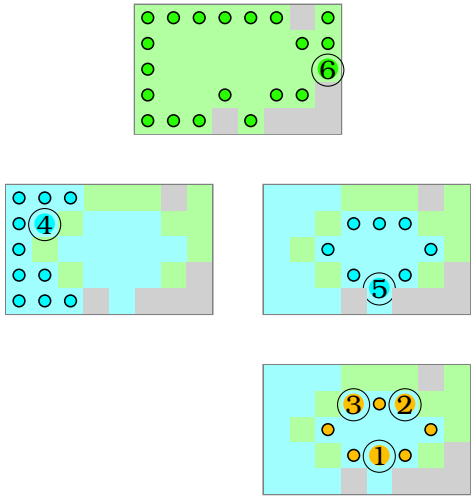
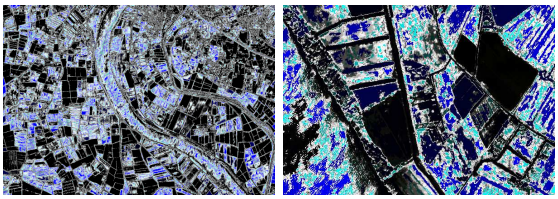


Fig. 32 – Pixel borders with C4 connectivity

Each pixel joins the tree at a base node. At this moment, it could be surrounded by the same value in flat area and never be considered as a border. Or it may be close to another value. If the value belongs to an under-node (near the leaves), it should simply wait to be joined by this neighbor and never be a border. If the value belongs to a upper-node (close to the root), it will play the role of border until it joins this node. So, the general approach is for each pixel, we consider the higher ancestor of all its neighbors. The algorithm 4 is quasi-linear. Time per pixel depends on efficient way to find ancestor.



(a) Arles (b) Center

Fig. 33 – Perimeter / area (10, 100, 5000)

Algorithm 4 Compute *perimeter*

```
for c ∈ nodes do
    perimeter(c) ← 0
for pi ∈ data − pixels do
    if isFrameImage(pi) then
        increasePerimeter(pp, root)
    else
        pp ← parent(pi)
        max ← pp
        npSet ← ∅
        for n ∈ neighbors − C4(pi) do
            if isNoData(n) then
                max ← root
            else
                npSet ← npSet ∪ parent(n)
        if max = root then
            increasePerimeter(pp, root)
        else
            sort(npSet)
            for np ∈ npSet do
                max ← ancestor(max, np)
            increasePerimeter(pp, max)
```

7.4 ZLength

The elevation axe (Z) is perpendicular of the XY axes. It correspond to the stack of bands. It could be used in a data cube to represent time for instance. The fast way to compute it is to compute the difference between the max and the min z value of all pixels.

Because the length is a difference between extra of node, it always growing from at least 1 to the number of bands. If the pixels is contained in a flat zone in all bands, the value will be the number of bands all along the branch, from the pixel to the root.

The final process of elevation is linear, but based on bounding box attribute.

Algorithm 5 Compute *zLength*

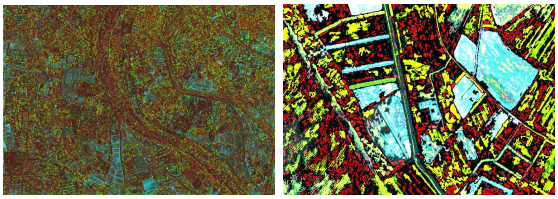
for $c \in nodes$ **do**
  $bb \leftarrow boundingBox_c$
  $zlength_c \leftarrow bb.max_z - bb.min_z$

7.5 Compactness

The compactness is obtained from area and perimeter. That mean the value function from pixel to root is not monotonous. The best compact exemple is a disq (minimum perimeter for a maximum pixels inside).

Don't confuse compactness and complexity. For instance a circle is compact and simple. A grid is compact and complex.

The final process of compactness is linear, but based on area and perimeter attributes. The compactness function is not monotonous.



(a) Arles (b) Center

Fig. 34 – Compactness / area (10, 100, 5000)

— Need perimeter values

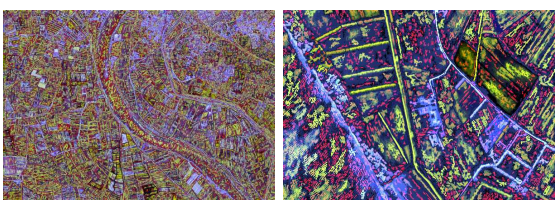
Algorithm 6 Compute *compactness*

for $c \in nodes$ **do**
  $compactness_c \leftarrow areas_c / {perimeters_c}^2$

7.6 Complexity

The complexity depends on the perimeter. More you have neighbors pixel, more you are complex (like start ou grid).

The final process of complexity is linear, but based on area and perimeter attributes. The complexity function is not monotonous.



(a) Arles (b) Center

Fig. 35 – Complexity / area (10, 100, 5000)

— Need perimeter values

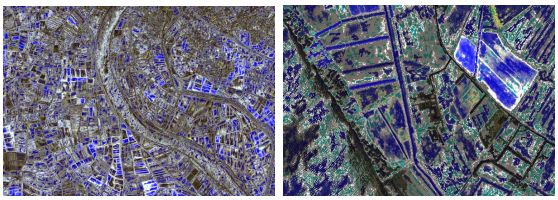
Algorithm 7 Compute *complexity*

for $c \in nodes$ **do**
  $complexity_c \leftarrow perimeters_c / areas_c$

7.7 Simplicity

Excactly inverse of complexity, it also depends on the perimeter. More you have neighbors pixel, less you are complex (like disq).

The final process of simplicity is linear, but based on area and perimeter attributes. The simplicity function is not monotonous.



(a) Arles (b) Center

Fig. 36 – Simplicity / area (10, 100, 5000)

— Need perimeter values

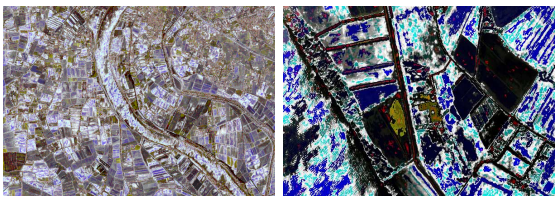
Algorithm 8 Compute *simplicity*

for $c \in nodes$ **do**
  $simplicity_c \leftarrow areas_c / perimeters_c$

7.8 Rectangularity

Rectangularity is not a "fair" attribute because it depends on the map direction (west-east or north-south). If we perform 45 degrees rotation, the result will change.

The rectangularity depends on area and bounding box. The function is not monotonous but obtain with linear computation.



(a) Arles (b) Center

Fig. 37 – Rectangularity / area (10, 100, 5000)

— Need bounding box values

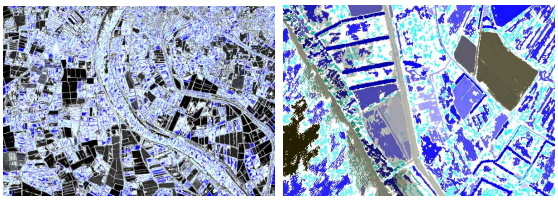
Algorithm 9 Compute *rectangularity*

for $c \in nodes$ **do**
  $bb \leftarrow boundingBox_c$
  $bbArea \leftarrow 1$
  for $a \in \{x, y, z\}$ **do**
    $bbArea \leftarrow bbArea \cdot (bb_a.max - bb_a.min)$
  $rectangularity_c \leftarrow areas_c / bbArea$

7.9 Weight

The weight is directly derive from the node value during the builting process.

It need no extra time to compute it. The type of function depends on the metric used to built the tree. It could be monotonous increase or decrease function (min or max). It could be non-monotonous (tree of shape).



(a) Arles (b) Center

Fig. 38 – Max / area (10, 100, 5000)

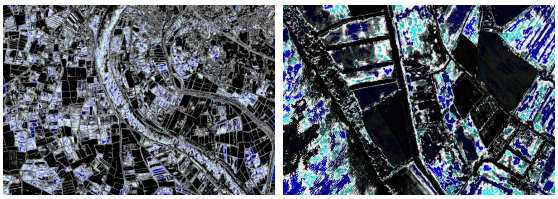
Algorithm 10 Compute *weight*

Done within compute tree

7.10 SD

The Standard Deviation attribute could be ambiguous. One can consider the gray-scale (weight) node value changes in branch. Another can consider the area node changes in a branch. We consider the weight.

The SD depends on mean attribute. The function is not monotonous but obtain with linear computation.



(a) Arles (b) Center

Fig. 39 – SD / area (10, 100, 5000)

$$\sigma = \sqrt{V} = \sqrt{\frac{1}{N}\sum_{i=0}^{n}(x_i - \bar{x})^2}$$

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=0}^{n} {x_i}^2 - \bar{x}^2}$$

— Need mean values

As an efficient implementation we only consider the square value of SD. The explain is we usually use this attribute as input for Random Forest. The main issue for the learning process is to find Inflection point of the curves. So we save time by skip compute square root.

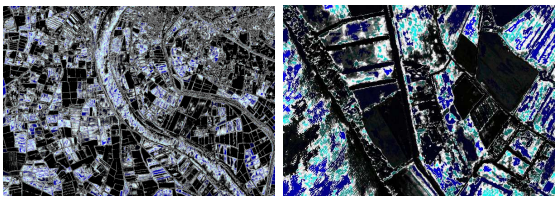
Algorithm 11 Compute sd^2

for $c \in nodes$ **do**
  $sd_c \leftarrow 0$
for $c \in pixels$ **do**
  $p \leftarrow parent(c)$
  if $p \neq root$ **then**
    $sd_p \leftarrow sd_p + (values_c)^2$
for $c \in nodes$ **do**
  $tmp \leftarrow (mean_c)^2$
  $sd_c \leftarrow sd_c/areas_c - tmp$
  $p \leftarrow parent(c)$
  if $p \neq root$ **then**
    $sd_p \leftarrow sd_p + tmp$

7.11 SDW

It's an efficient way to obtain a curve with same shape as SD, if the final issue is to be used as input of Random Forest. Sometimes the weight node values (gray-scale) are similar as average values in a node (med-tree). In that case, take the weight is an efficient substitution value.

The function is not monotonous but obtain with linear computation.



(a) Arles (b) Center

Fig. 40 – SDW / area (10, 100, 5000)

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=0}^{n} {x_i}^2 - weight_c^2}$$

Algorithm 12 Compute sdw^2

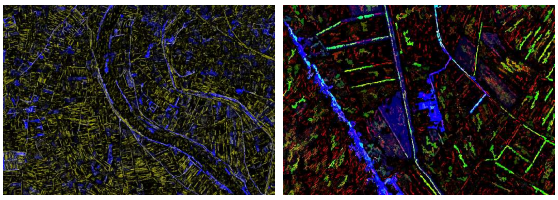
for $c \in nodes$ **do**
  $sd_c \leftarrow 0$
for $c \in pixels$ **do**
  $p \leftarrow parent(c)$
  if $p \neq root$ **then**
    $sd_p \leftarrow sd_p + (values_c)^2$
for $c \in nodes$ **do**
  $tmp \leftarrow (weight_c)^2$
  $sd_c \leftarrow sd_c/areas_c - tmp$
  $p \leftarrow parent(c)$
  if $p \neq root$ **then**
    $sd_p \leftarrow sd_p + tmp$

7.12 MoI

The moment-of-inertia attribute could be ambiguous. One can consider the gray-scale node value compare to its children nodes. Another can consider

the centroide node compare to its children nodes. We consider the centroide.

The SD depends on centroide, itself depends on bounding box attribute. The final process of MoI is linear. The function is not monotonous.



(a) Arles (b) Center

Fig. 41 – MOI / area (10, 100, 5000)

$$moi = \frac{\sum_{i=0}^{n} d(i,center)^2 \cdot area_i}{area^2}$$

— Need average coordonate values

Algorithm 13 Compute moi

for $c \in nodes$ **do**
  $moi_c \leftarrow 0$
for $c \in pixels$ **do**
  $p \leftarrow parent(c)$
  if $p \neq root$ **then**
    for $a \in \{x, y, z\}$ **do**
      $moi_p \leftarrow moi_p + (coord_{c,a} - coord_{p,a})^2$
for $c \in nodes$ **do**
  $moi_c \leftarrow moi_c/(area_c)^2$
  $d \leftarrow 0$
  for $a \in \{x, y, z\}$ **do**
    $d \leftarrow d + (coord_{c,a} - coord_{p,a})^2$
  $p \leftarrow parent(c)$
  if $p \neq root$ **then**
    $moi_p \leftarrow moi_p + d \cdot area_c$

7.13 Training

— Attribute cut (all)

— Thresholds

8 Filtering

8.1 Filtering

Feature is a vector that is computed based on node attributes in a tree structure. It is a descriptor for the underlying image and can be used for image classification / segmentation / retrieval. In this section we introduce attribute profile that is a well known family of tree features.

Mathematically, Attribute Profile (AP) of an image is defined as the stack of images that are produced by successively applying a set of morphological attribute filters. Attribute filters are applied on connected components of the image instead of its pixels considering a specific attribute of the components such as size and shape. Therefore, each pixel of the original image will be represented in the attribute profile by a vector (stack of pixels) that can be used later for classification or segmentation.

Practically, AP can be computed efficiently using the tree representation of an image (See Figure 42 in page 28). A tree is constructed from original image and the attribute is computed for its nodes. Tree is pruned with several thresholds on its node attributes and the filtered images are reconstructed from pruned trees. The stack of filtered images will produce the desired attribute profile. For example the max-tree in Figure 11 is pruned by two area thresholds. As the result the two filtered images along with the original image form the attribute profile with three layers. Therefore, each pixel is represented by a three-dimensional vector.

- Monotonic attributes or not
- From attributes (type + threshold)
- To attributes (type)
- DAP

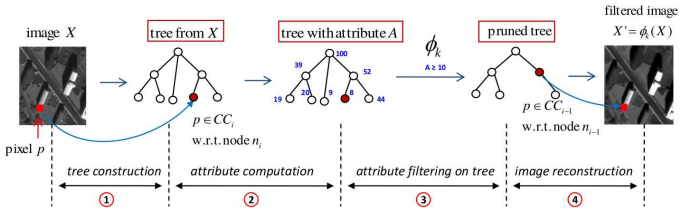


Fig. 42 – General framework of filtering an image by applying attribute filter on its tree representation [24]

8.2 Monotonic

In mathematics, a monotonic function is a function between ordered sets that preserves or reverses the given order. We cassified attribut according the monotonic common properties of functions from all pixels to the root.

- Examples
  - Area: ↗ monotonous increasing
  - Rectangularity: ⇝ not monotonous
  - Weight
    - Max-tree: ↗ monotonous increasing
    - Min-tree: ↘ monotonous decreasing
    - Tree-of-Shape: ⇝ not monotonous

Some strategies can be applyed to prune.

- Rules
  - Direct first occurrence
  - Max get max of all sub-nodes
  - Min get min of all sub-nodes
  - Subtractive

8.3 Attribute profile

Up to now, all things we done when filtering process is to extract the value (gray-scale) of a node. The gray-scale value is an attribute.

We can imagine attract any other kind of attributes. This generic approach is call features attraction.

We firstofall, details the Attributes Profile, then extended the concept to Features Profile.

- Output weight value
- Attribute selection
- Thresholds

```
channelGenerator arlesGS.tif o.tif -b 0 -f AP -t Med -a Area --thresholds 10,100,5000 --time
```

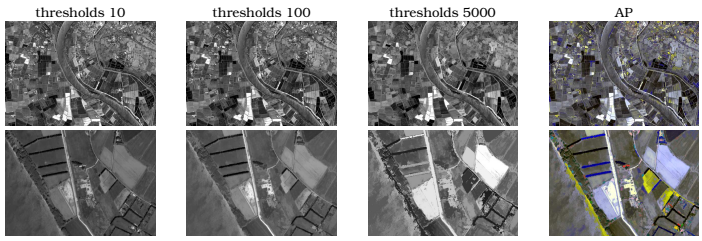


Fig. 43 – Attribute Profile

Recently, a novel extension of APs called feature profiles (FPs) [19] has been proposed by replacing pixel gray-levels with the attributes themselves when forming the output profiles.

FPs have been proved to be more efficient than the standard APs and showed that APs are indeed a part of the general FPs, in case that the gray-level is used as attribute to form the output profiles.

A recent paper has investigated the performance of APs and FPs generated from different kinds of tree structures (min-tree, max-tree, tree of shapes and partition trees) [20].

Another one has studied the filtering rules [8] during AP construction.

Nevertheless, tree formation and pruning are not the only factors that affect AP and FP performance.

The choice of attributes is even more important since they decide how the tree should be pruned as well as which output profiles are formed.

8.4 Feature profile

— Output attribute value

— Feature selection

```
channelGenerator arlesGS.tif o.tif -b 0 -f Area -t Med -a Area --thresholds 10,100,5000 --time
```

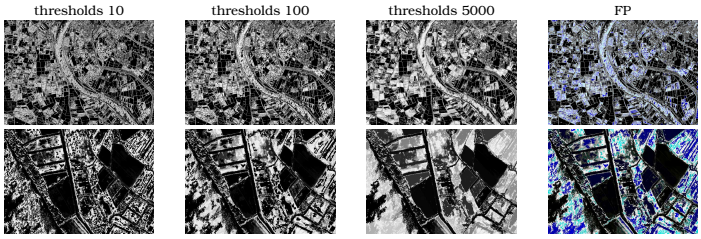
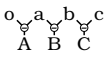
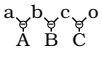
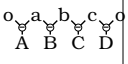
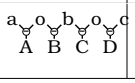


Fig. 44 – Feature Profile

8.5 Differential Attribute Profiles (DAP)

— Position

— Weight

Name	Representation	Compute the difference between:
apOrigNoPos	**0** 	- all successive filtered images
apOrigPosBegin	**1** 	- all successive filtered images - the original image and the first filtered image (lowest threshold)
apOrigPosEnd	**2** 	- all successive filtered images - the original image and the last filtered image (highest threshold)
apOrigPosBoth	**3** 	- all successive filtered images - the original image and the first filtered image (lowest threshold) - the original image and the last filtered image (highest threshold)
apOrigPosEverywhere	**4** 	- all the filtered image and the original image

Tab. 4 – DAP original band position

```
channelGenerator arlesGS.tif o.tif -b 0 -f AP -t Med -a Area --thresholds 10,100,5000 --dapPos
↪ apOrigPosBegin --auto --time
```

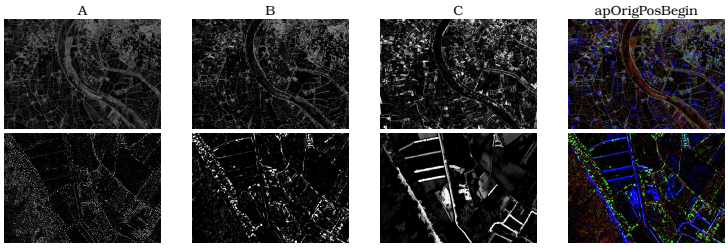


Fig. 45 – DAP example

8.6 Training

— DAP (FP, AP, thresholds)

9 Textural tools

The features profile are used as input of Random Forest. We take benefits to extends inputs to other textural attributes.

All textural function offered by Triskele are base on linear algorithms.

9.1 Texture tools addon

This section presents

Non-hierarchical attributes

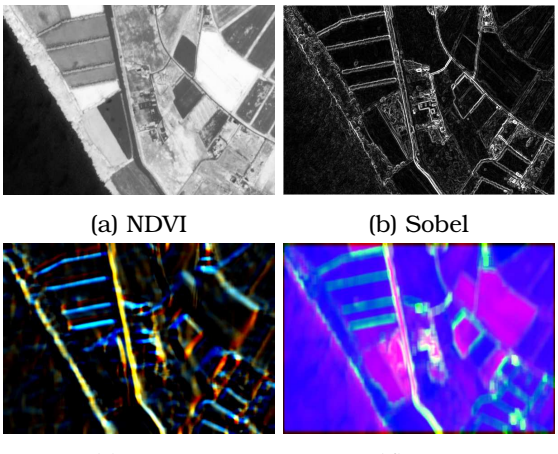
— NDVI vegetation

— Sobel region border

— Pantex urban index

— Haar orientation

— Stat gray-scale statistics



(a) NDVI (b) Sobel

(c) Haar (d) Stat

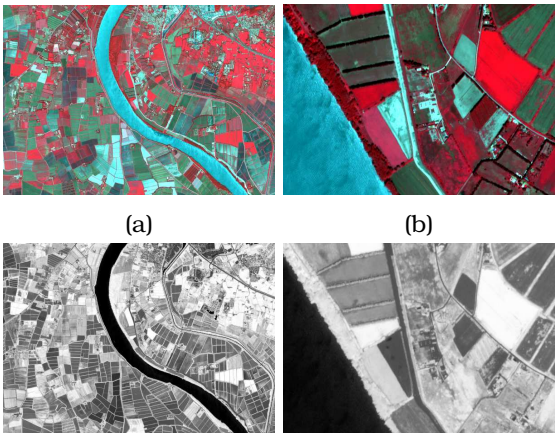
Fig. 46 – Textural addon

9.2 NDVI

The Normalized Difference Vegetation Index (NDVI) is an index related to liquid vegetation. It is based on red and infra-red bands and the result is between -1 and +1.

$$NDVI = \frac{(NIR-Red)}{NIR+Red}$$

```
channelGenerator arles.tif ndvi.tif --ndviBands 0,1 --copyBand 3
```



(a) (b)

(c) (d)

Fig. 47 – a and b: arles and center city, c and d: NDVI respectively from a and b

9.3 NDWI

Normalized Difference Water Index (NDWI) is an index related to liquid water. It can be use to monitor changes related to water content in water bodies, using green and NIR wavelengths, defined by McFeeters (1996). It is based on red and infra-red bands and the result is between -1 and +1.

$NDWI = \frac{(Green-NIR)}{Green+NIR}$ Because it's the same kind equation as NDVI, we can use this equation with the appropriate bands.

9.4 Sobel

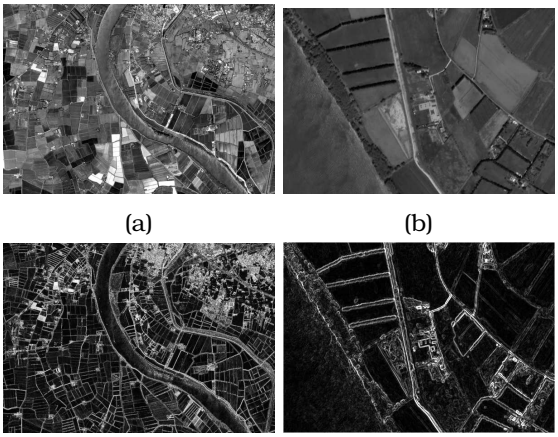
Sobel is used to highlight region border, for instance the road, the fields and so on.

In Triskele, we have implemented a linear algorithm.

$$G_x = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \cdot \left(\begin{bmatrix} +1 & 0 & -1 \end{bmatrix} \cdot A \right) \qquad G_y = \begin{bmatrix} +1 \\ 0 \\ -1 \end{bmatrix} \cdot \left(\begin{bmatrix} 1 & 2 & 1 \end{bmatrix} \cdot A \right)$$

$$G = \sqrt{G_x^2 + G_y^2}$$

```
channelGenerator arlesGS.tif sobel.tif -s 0 -c 1
```



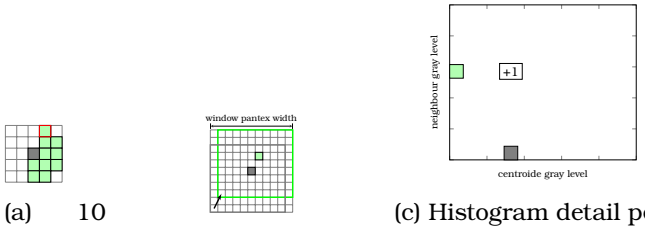
(a) (b) (c) (d)

Fig. 48 – a and b: arles and center city, c and d: Sobel respectively from a and b

9.5 Pantex Index

PanTex is an automatic recognition of built-up areas is based on analysis of image textural measures extracted using an-isotropic rotation-invariant gray-level co-occurrence matrix (GLCM) statistics.

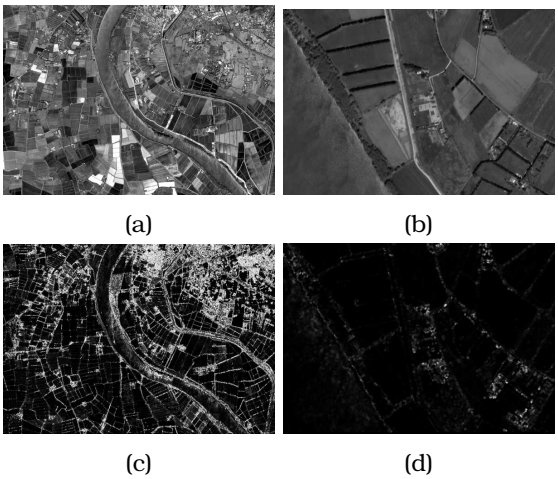
$$contrast = \sum_{i=0}^{127}\sum_{j=0}^{127}(i-j)^2.GLCM_{i,j}$$



(a) 10 vectors (b) One scan per vector scan (c) Histogram detail per

Fig. 49 – Pantex Index steps

```
channelGenerator arlesGS.tif pantex5.tif -p 0 -c 1 --pantexWindowSide 5
```



(a) (b) (c) (d)

Fig. 50 – a and b: arles and center city, c and d: Pantex Index respectively from a and b

9.6 Haar

Haar gives a score according the vertical or horizontal split of an images. We consider 4 axes: X, Y, X+Y, Y-X. Thanks to these values we can detect vertical, horizontal or diagonal items on an image.

We have implemented a linear algorithm based on integral images.

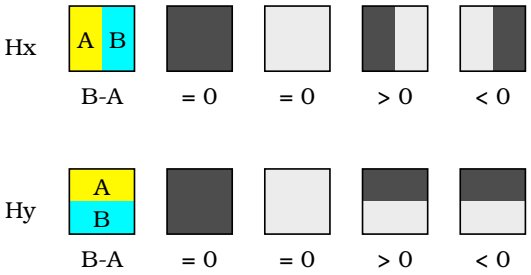


Fig. 51 – Haar values

```
channelGenerator arlesGS.tif o.tif -b 0 --haar 20x20 --time
```

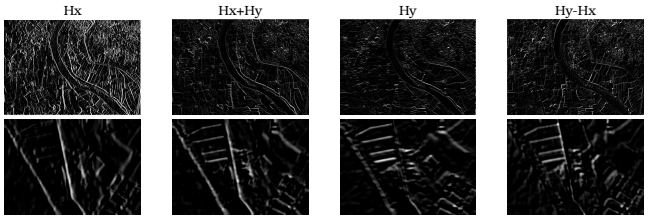


Fig. 52 – Haar visual example

9.7 Stat

— Mean
— Standard-deviation
— Entropy

$$mean = \frac{\sum_{i=0}^{n} p_i}{n}$$

$$var = \frac{\sum_{i=0}^{n-1} p_i^2}{n} - mean^2$$

$$std = \sqrt{var}$$

```
channelGenerator arlesGS.tif o.tif -b 0 --stat 10x20 --time
```

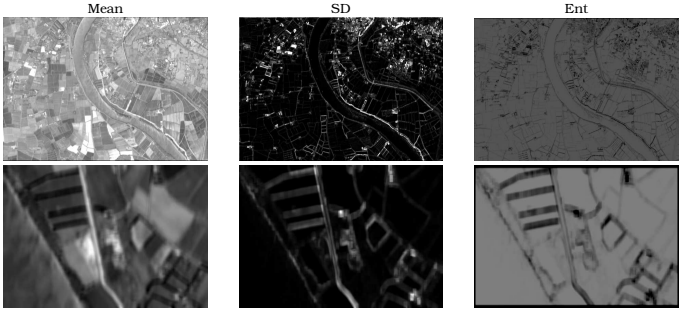


Fig. 53 – Stat visual example

9.8 Training

— NDVI

— Sobel

— Pantex

10 Related work

We didn't produce hierarchical structure just for fun. The final target is analyzed remote sensing images.

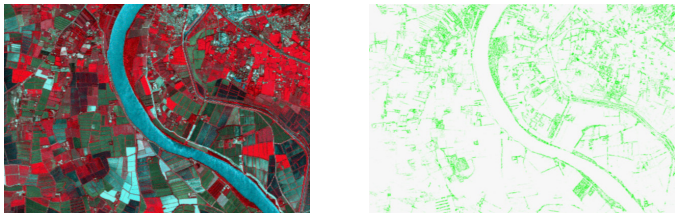
Attribute profiles or feature profiles and textural data as well are used as input of mechine learning for classification (for instance).

Note some textural products could be use to generate new hierarchical data. We can produce a max-tree from NDVI or Sobel.

10.1 Land cover map

Broceliande This is an extention of Triskele. It combines hierarchical representation and machines learning (Random Forest).

- Random forest
- Prediction
- Example small woody feature



(a) Input (b) Output

Fig. 54 – Broceliande: Classification with hierarchical representation using random forest

10.2 Find patches

Korrigan

We presently work on an other extention call Korrigan. It use Triskele to produce Pattern Spectra. They are used as signature of region of interest.

The method is the following:

- pre-compute all atomic PS data of a remote sensing database
- user define a seeking region
- compute atomic PS of this region
- find the candidate region in linear time in database

10.3 Training

- find Small Woody Features

References

[1] Erchan Aptoula, Mauro Dalla Mura, and Sébastien Lefèvre. Vector attribute profiles for hyperspectral image classification. *IEEE Transactions on Geoscience and Remote Sensing*, 54(6):3208–3220, 2016.

[2] Petra Bosilj, Erchan Aptoula, Sébastien Lefèvre, and Ewa Kijak. Retrieval of remote sensing images with pattern spectra descriptors. *ISPRS Int. J. Geo-Information*, 5(12):228, 2016.

[3] Petra Bosilj, Bharath Bhushan Damodaran, Erchan Aptoula, Mauro Dalla Mura, and Sébastien Lefèvre. Attribute profiles from partitioning trees. In *Mathematical Morphology and Its Applications to Signal and Image Processing - 13th International Symposium, ISMM 2017, Fontainebleau, France, May 15-17, 2017, Proceedings*, pages 381–392, 2017.

[4] Petra Bosilj, Ewa Kijak, and Sébastien Lefèvre. Partition and inclusion hierarchies of images: A comprehensive survey. *J. Imaging*, 4(2):33, 2018.

[5] Gabriele Cavallaro, Mauro Dalla Mura, Jon Atli Benediktsson, and Antonio J. Plaza. Remote sensing image classification using attribute filters defined over the tree of shapes. *IEEE Trans. Geoscience and Remote Sensing*, 54(7):3899–3911, 2016.

[6] Mauro Dalla Mura, Jón Atli Benediktsson, Björn Waske, and Lorenzo Bruzzone. Morphological attribute profiles for the analysis of very high resolution images. *IEEE Transactions on Geoscience and Remote Sensing*, 48(10):3747–3762, 2010.

[7] Bharath Bhushan Damodaran, Joachim Höhle, and Sébastien Lefèvre. Attribute profiles on derived features for urban land cover classification. *Photogrammetric engineering and remote sensing*, 83(3):183–193, 2017.

[8] Arundhati Das, Kaushal Bhardwaj, and Swarnajyoti Patra. A comparison of different filtering strategies used in attribute profiles for hyperspectral image classification. In *International Conference on Intelligent Computing and Smart Communication 2019*, pages 1017–1026. Springer, 2020.

[9] Nicola Falco, Mauro Dalla Mura, Francesca Bovolo, Jon Atli Benediktsson, and Lorenzo Bruzzone. Change detection in VHR images based on morphological attribute profiles. *IEEE Geosci. Remote Sensing Lett.*, 10(3):636–640, 2013.

[10] Thierry Géraud, Edwin Carlinet, Sébastien Crozet, and Laurent Najman. A quasi-linear algorithm to compute the tree of shapes of *n*d images. In *Mathematical Morphology and Its Applications to Signal and Image Processing, 11th International Symposium, ISMM 2013, Uppsala, Sweden, May 27-29, 2013. Proceedings*, pages 98–110, 2013.

[11] P. Ghamisi, M. Dalla Mura, and J. A. Benediktsson. A survey on spectral–spatial classification techniques based on attribute profiles. *IEEE Transactions on Geoscience and Remote Sensing*, 53(5):2335–2353, May 2015.

[12] Florent Guiotte, Sébastien Lefèvre, and Thomas Corpetti. Attribute filtering of urban point clouds usingmax-tree on voxel data. In *International Symposium on Mathematical Morphology (ISMM)*, 2019.

[13] Florent Guiotte, Sébastien Lefèvre, and Thomas Corpetti. Rasterization strategies for airborne lidar classification using attribute profiles. In *Joint Urban Remote Sensing Event*, 2019.

[14] D. Mongus, N. Lukač, D. Obrul, and B. Žalik. Detection of planar points for building extraction from lidar data based on differential morphological and attribute profiles. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, II-3/W1:21–26, 2013.

[15] Mauro Dalla Mura, Jon Atli Benediktsson, Björn Waske, and Lorenzo Bruzzone. Morphological attribute profiles for the analysis of very high resolution images. *IEEE Trans. Geoscience and Remote Sensing*, 48(10):3747–3762, 2010.

[16] Mauro Dalla Mura, Alberto Villa, Jon Atli Benediktsson, Jocelyn Chanussot, and Lorenzo Bruzzone. Classification of hyperspectral images by using extended morphological attribute profiles and independent component analysis. *IEEE Geosci. Remote Sensing Lett.*, 8(3):542–546, 2011.

[17] G Ouzounis, P Soille, and M Pesaresi. Rubble detection from vhr aerial imagery data using differential morphological profiles. In *34th Int. Symp. Remote Sensing of the Environment*, 2011.

[18] Claudia Paris, Davide Valduga, and Lorenzo Bruzzone. A hierarchical approach to three-dimensional segmentation of lidar data at single-tree level in a multilayered forest. *IEEE Trans. Geoscience and Remote Sensing*, 54(7):4190–4203, 2016.

[19] Minh-Tan Pham, Erchan Aptoula, and Sébastien Lefèvre. Feature profiles from attribute filtering for classification of remote sensing images. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 11(1):249–256, 2017.

[20] Minh-Tan Pham, Erchan Aptoula, and Sébastien Lefèvre. Classification of remote sensing images using attribute profiles and feature profiles from different trees: a comparative study. In *IGARSS 2018-2018 IEEE International Geoscience and Remote Sensing Symposium*, pages 4511–4514. IEEE, 2018.

[21] Minh-Tan Pham, Erchan Aptoula, and Sébastien Lefèvre. Feature profiles from attribute filtering for classification of remote sensing images. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 11(1):249–256, 2018.

[22] Minh-Tan Pham, Sébastien Lefèvre, and Erchan Aptoula. Local feature-based attribute profiles for optical remote sensing image classification. *IEEE Transactions on Geoscience and Remote Sensing*, 56(2):1199–1212, 2018.

[23] Minh-Tan Pham, Sébastien Lefèvre, Erchan Aptoula, and Lorenzo Bruzzone. Recent developments from attribute profiles for remote sensing image classification. *arXiv preprint arXiv:1803.10036*, 2018.

[24] Minh-Tan Pham, Sébastien Lefèvre, Erchan Aptoula, and Lorenzo Bruzzone. Recent developments from attribute profiles for remote sensing image classification. *CoRR*, abs/1803.10036, 2018.

[25] Minh-Tan Pham, Sébastien Lefèvre, and François Merciol. Attribute profiles on derived textural features for highly textured optical image classification. *IEEE Geoscience and Remote Sensing Letters*, 15(7):1125–1129, 2018.

[26] Philippe Salembier and Luis Garrido. Binary partition tree as an efficient representation for image processing, segmentation, and information retrieval. *IEEE Trans. Image Processing*, 9(4):561–576, 2000.

[27] Gunho Sohn, Xianfeng Huang, and Vincent Tao. Using a binary space partitioning tree for reconstructing polyhedral building models from airborne lidar data. photogrammetric engineeering and remote. *Sensing*, 2008.

[28] Ashkan Taghipour and Hassan Ghassemian. Hyperspectral anomaly detection using attribute profiles. *IEEE Geosci. Remote Sensing Lett.*, 14(7):1136–1140, 2017.

[29] Robert Endre Tarjan. Efficiency of a good but not linear set union algorithm. *J. ACM*, 22(2):215–225, 1975.

[30] Ayse Tombak, Erchan Aptoula, and Koray Kayabol. Pixel-based classification of SAR images using features. In *26th Signal Processing and Communications Applications Conference, SIU 2018, Izmir, Turkey, May 2-5, 2018*, pages 1–4, 2018.

[31] V. Vilaplana, F. Marques, and P. Salembier. Binary partition trees for object detection. *IEEE Transactions on Image Processing*, 17(11):2201–2216, 2008.

QUIZZ

(This page intentionally left blank)

+1/1/60+

☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9
☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9
☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9
☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9
☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9
☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9
☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9

⟵ fill your student ID on left (one digit per line), and your name bellow

Firstname and name :

.....

.....

Question 1 Some trees are named max-tree because the max values are located in

☐ links ☐ root ☐ nodes ☐ leaves

Question 2 With full connectivity in a 2D image, a pixel is linked with

☐ 4 neighbors ☐ 26 neighbors ☐ 9 neighbors ☐ 8 neighbors

Question 3 In the following propositions, what is an inclusion tree

☐ oak ☐ tree-of-shape ☐ α-tree ☐ min-tree

Question 4 The Chebyshev distance is

☐ $\#(i|x_i \neq 0)$
☐ $\sum_i |x_i|$
☐ $\sqrt[p]{\sum_i |x_i|^2}$
☐ $\max_i(|x_i|)$

Question 5 A tree structure is

☐ a graph ☐ classifier ☐ set of graphs ☐ optimizer

Question 6 A tree structures have

☐ cycled path ☐ disconnected nodes ☐ oriented links ☐ some roots

Question 7 The best tree structure to find dark cars on street is

☐ max-tree ☐ tree-of-shape ☐ min-tree ☐ med-tree

Question 8 The best tree to find white cars on street is

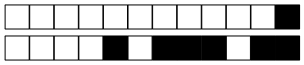
☐ tree-of-shape ☐ max-tree ☐ med-tree ☐ min-tree

Question 9 The minimum complexity to build a tree from a n-pixels image is

☐ n ☐ $n \times log(n \times n)$ ☐ $n \times log(n)$ ☐ $n \times n$

Question 10 An attribute is a property associate to

☐ a tree ☐ an edge ☐ leaves ☐ a node



Question 11 The Normalized Difference Vegetation Index is obtain with

- ☐ $\frac{(NIR+Red)}{NIR-Red}$
- ☐ $\frac{(Green-NIR)}{Green+NIR}$
- ☐ $\frac{(NIR+Green)}{NIR-Green}$
- ☐ $\frac{(NIR-Red)}{NIR+Red}$

Question 12 The NDWI gives information about

- ☐ Wind
- ☐ Wood
- ☐ Vegetation
- ☐ Water

Question 13 The PanTex is based on

- ☐ water
- ☐ gray-level
- ☐ infra-red
- ☐ vegetation

Question 14 What is the main quality of α-tree ?

- ☐ support chain effect
- ☐ support shadow
- ☐ highlight dark pixels
- ☐ highlight bright pixels

Question 15 Pixels data cubes are not appropriate for :

- ☐ time series
- ☐ volume
- ☐ multi-spectral
- ☐ stream

Question 16 Random Forest manage

- ☐ wood samples
- ☐ spars wood
- ☐ heterogeneous wood
- ☐ decision trees

Question 17 The perimeter is not necessary for

- ☐ rectangularity
- ☐ compactness
- ☐ complexity
- ☐ simplicity

Question 18 You must not used licensed images (payed by your company) in

- ☐ Google Colab
- ☐ company cloud
- ☐ personal computer
- ☐ company server

Question 19 Why it's not possible to build a tree from multi-spectral image ?

- ☐ Yes. It's possible with data cube
- ☐ lack of ordering pixels
- ☐ color artifacts
- ☐ lack of memory

Question 20 Pruning a tree

- ☐ remove items
- ☐ enhance diversity
- ☐ enhance nodes number
- ☐ highlight items