University of Murcia (Spain)
Artificial Perception and Pattern Recognition Research Group (PARP)

Component trees: yet another promising way to represent and process gray images

Author:
Pedro E. López-de-Teruel
pedroe@ditec.um.es

PARP

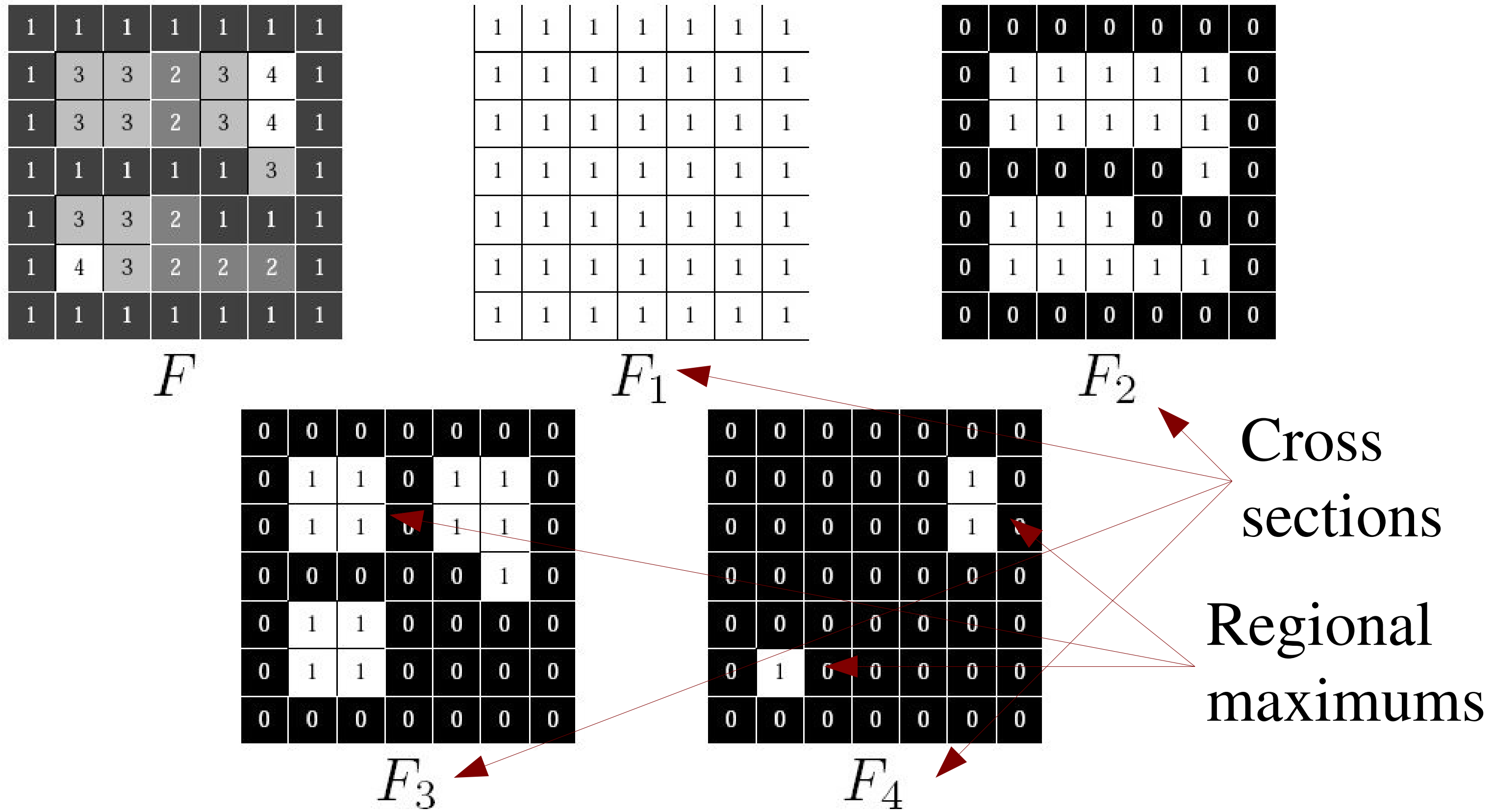
Introduction

- The *component (or confinement, or max) tree* is a hierarchical representation of the *level sets* of an image.
  - ***Level sets***: Sets of points in an image with gray level above a given threshold (*islands covered by water analogy*).
  - The **connected components of those sets**, thanks to the inclusion relation, can be organized in a **tree structure**.
  - Being careful, it can be computed in **quasi-linear time** $O(n\,\alpha(n))$, with $\alpha(n)$ growing very slowly: $\alpha(10^{80}) \cong 4$
- Application domains
  - Image **filtering** (*preservation of shapes*) and **segmentation**.
  - Image **registration** (matching) and **feature detection** (MSER).
  - Image **compression**.
  - Data **visualization**.
  - Also as a help for efficient implementation of other interesting algorithms (**topological watershed**, etc.).

PARP

Definitions (I)

- Basic notions (graph theory related):
  - *Nodes* (pixels), *edges* (neighborhood, 4-connection), *path*, *connected component*, *vertex weighting*, ...



PARP

Definitions (II)

1	1	1	1	1	1	1
1	3	3	2	3	4	1
1	3	3	2	3	4	1
1	1	1	1	1	3	1
1	3	3	2	1	1	1
1	4	3	2	2	2	1
1	1	1	1	1	1	1

F

1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1

F_1

0	0	0	0	0	0	0
0	1	1	1	1	1	0
0	1	1	1	1	1	0
0	0	0	0	0	1	0
0	1	1	1	0	0	0
0	1	1	1	1	1	0
0	0	0	0	0	0	0

F_2

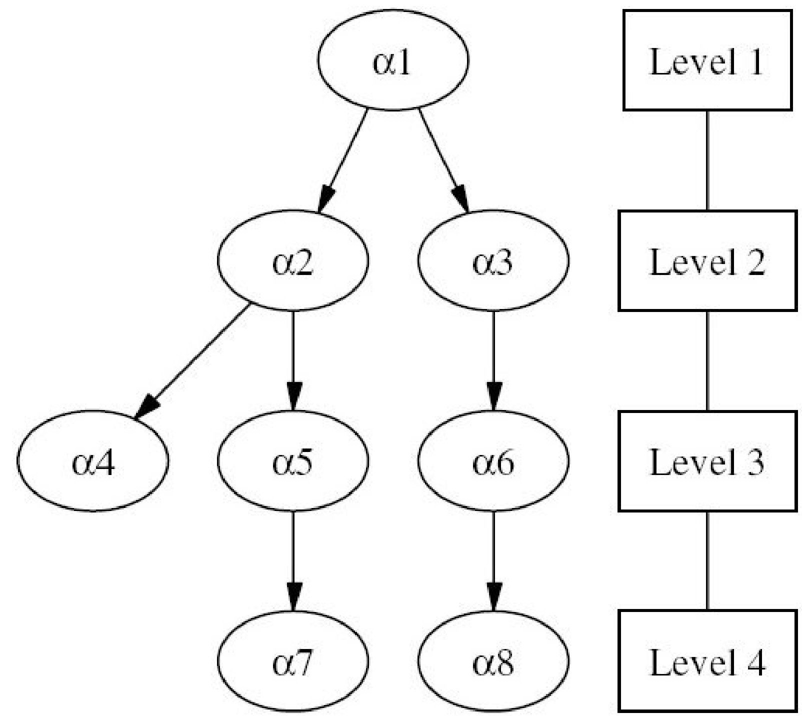
0	0	0	0	0	0	0
0	1	1	0	1	1	0
0	1	1	0	1	1	0
0	0	0	0	0	1	0
0	1	1	0	0	0	0
0	1	1	0	0	0	0
0	0	0	0	0	0	0

F_3

0	0	0	0	0	0	0
0	0	0	0	0	1	0
0	0	0	0	0	1	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	1	0	0	0	0	0
0	0	0	0	0	0	0

F_4

Original image and cross-sections



Component tree

α_1	α_1	α_1	α_1	α_1	α_1	α_1
α_1	α_4	α_4	α_2	α_5	α_7	α_1
α_1	α_4	α_4	α_2	α_5	α_7	α_1
α_1	α_1	α_1	α_1	α_1	α_5	α_1
α_1	α_6	α_6	α_3	α_1	α_1	α_1
α_1	α_8	α_6	α_3	α_3	α_3	α_1
α_1	α_1	α_1	α_1	α_1	α_1	α_1

Component mapping

(associates each pixel to its corresponding node in the tree, at its corresponding level)

PARP

Union-find algorithm for disjoint sets

- *Disjoint set problem*: maintaining a collection of disjoint subsets of a set under the operation of union.
  - Quasi-linear complexity, using three operations: ***MakeSet***, ***Find*** & ***Link***.

Procedure `MakeSet` (*element* x)

  Par(x) := x; Rnk(x) := 0;

Function `element Find` (*element* x)

  if (*Par*(x) $\neq$ x) **then** Par(x) := Find(Par(x));
  return Par(x);

Function `element Link` (*element* x, *element* y)

  if (*Rnk*(x) $>$ *Rnk*(y)) **then** exchange(x, y);
  if (*Rnk*(x) == *Rnk*(y)) **then** Rnk(y) := Rnk(y) + 1;
  Par(x) := y;
  return y;

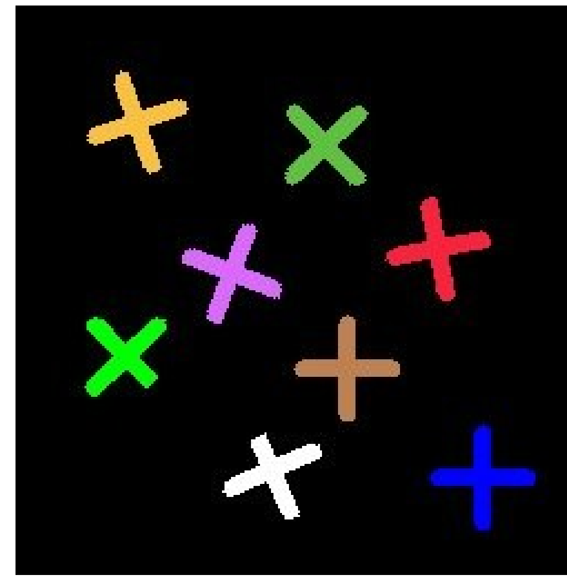
  - Try to **keep each "tree" as flat as possible**, by (1) using always the deepest tree as parent, and (2) linking directly to parent each time we visit a node (while finding a ***canonical element***).

Example: connected components

Algorithm 1: ConnectedComponents

Data: (V, E) - graph
Data: A set $X \subseteq V$
Result: M - map from X to V

foreach $p \in X$ **do** MakeSet(p);
foreach $p \in X$ **do**
    compp := Find(p);
    foreach $q \in \Gamma(p) \cap X$ **do**
        compq := Find(q);
        if *(compp $\neq$ compq)* **then**
            compp := Link(compq, compp);
foreach $p \in X$ **do** M(p) := Find(p);



Output

Component tree algorithm (I)

- Simulation of an ***emergence process*** (islands uncovered by progressively decreasing level of water):
  - Two disjoint sets:
    - $\mathbf{Q}_{\mathbf{node}}$, for vertices belonging to the same component and with the same gray level (*flat zones,* each component has its corresponding *canonical node*), and
    - $\mathbf{Q}_{\mathbf{tree}}$, with each tree corresponding to an *emerged island*; nodes are progressively linked together to form partial trees (= *islands*)
  - An auxiliary map called ***LowestNode***, that associates the root of the corresponding partial tree to each canonical element of $\mathbf{Q}_{\mathbf{tree}}$ (needed because, by construction, this canonical element is not guaranteed to be the root of the corresponding partial tree).

PARP

Component tree algorithm (II)

- Example:

110	90	100
50	50	50
40	20	50
50	50	50
120	70	80

Gray level image

0	1	2
3	4	5
6	7	8
9	10	11
12	13	14

Lexicographic order

Processing order: $\underbrace{12}_{120}, \underbrace{0}_{110}, \underbrace{2}_{100}, \underbrace{1}_{90}, \underbrace{14}_{80}, \underbrace{13}_{70}, \underbrace{3, 4, 5, 8, 9, 10, 11}_{50}, \underbrace{6}_{40}, \underbrace{7}_{20}$

PARP

DITEC DIS Universidad de Murcia

Component tree algorithm (III)

- Beginning of 7th step (going to process level 50):

1	1	1
3	4	5
6	7	8
9	10	11
13	13	13

Par_{tree}

0	1	2
3	4	5
6	7	8
9	10	11
12	13	14

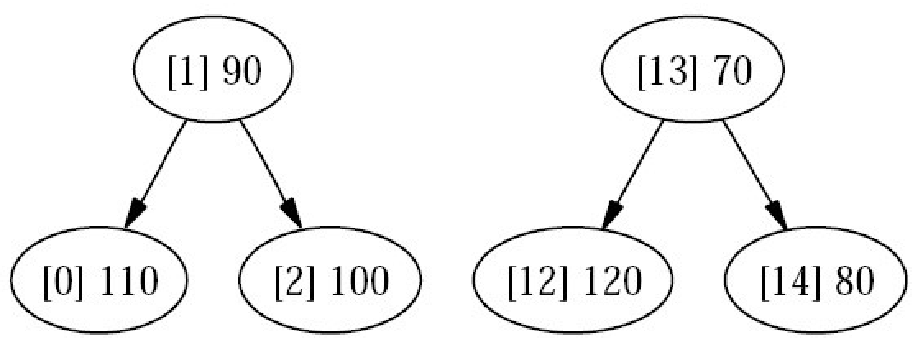
Par_{node}

0	1	2
3	4	5
6	7	8
9	10	11
12	13	14

$lowestNode$

(a)

Gray: updated values



(b)

Gray: values that will not get updated

Component tree algorithm (IV)

- Beginning of 13th step (going to process node 11):

1	3	3
3	3	3
6	7	3
9	9	11
13	9	13

Par_{tree}

0	1	2
3	3	3
6	7	3
9	9	11
12	13	14

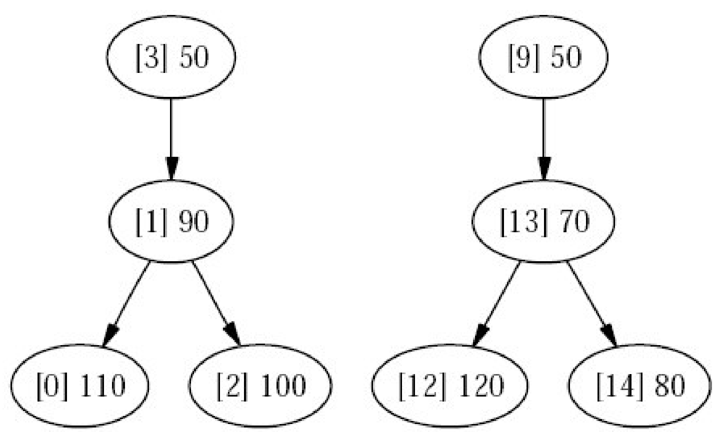
Par_{node}

0	1	2
3	4	5
6	7	8
9	10	11
12	13	14

$lowestNode$

(a)

110	90	100
50	50	50
40	20	50
50	50	50
120	70	80



(b)

Component tree algorithm (V)

- End of 13th step (after processing node 11 & merging level 50 components):

1	3	3
9	3	3
6	7	3
9	9	3
13	9	13

$\mathrm{Par}_{\mathrm{tree}}$

0	1	2
9	3	3
6	7	3
9	9	3
12	13	14

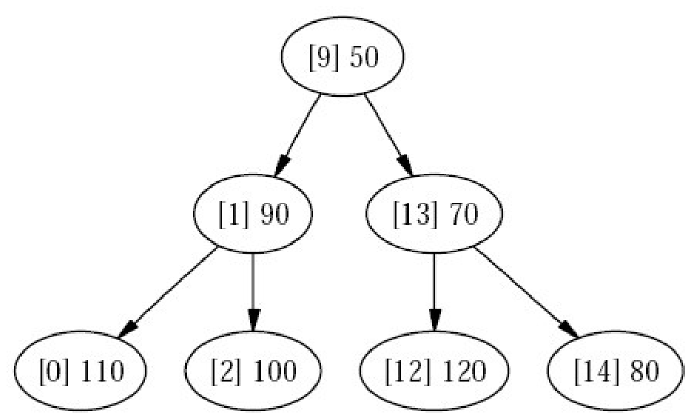
$\mathrm{Par}_{\mathrm{node}}$

0	1	2
3	4	5
6	7	8
9	10	11
12	13	14

$lowestNode$

(a)

110	90	100
50	50	50
40	20	50
50	50	50
120	70	80



(b)

Component tree algorithm (VI)

- Final result:

1	3	3
9	9	3
9	9	9
9	9	9
13	9	9

$\mathrm{Par}_{\mathrm{tree}}$

0	1	2
9	3	3
6	7	3
9	9	3
12	13	14

$\mathrm{Par}_{\mathrm{node}}$

0	1	2
3	4	5
6	7	8
7	10	11
12	13	14

$lowestNode$

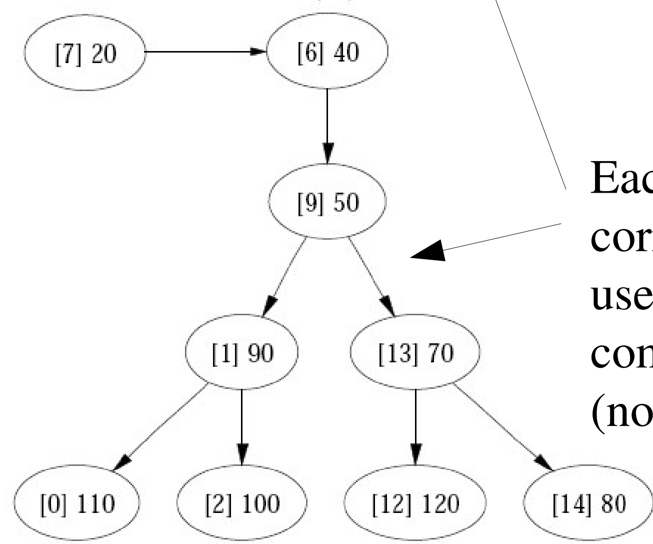
(a)

Not useful anymore (just one component)

Used to find the root of the tree

Each canonical element corresponds to a node; used to finally compute component mapping (not shown).

110	90	100
50	50	50
40	20	50
50	50	50
120	70	80



(b)

Component tree algorithm (VII)

Algorithm 2: BuildComponentTree

Data: (V, E, F) - vertex-weighted graph with N points.
Result: $nodes$ - array $[0 \ldots N-1]$ of nodes.
Result: $Root$ - Root of the component tree
Result: M - map from V to $[0 \ldots N-1]$ (component mapping).
Local: $lowestNode$ - map from $[0 \ldots N-1]$ to $[0 \ldots N-1]$.

Data structures

Sort the points in decreasing order of level for F;
foreach $p \in V$ **do** {MakeSet$_{\text{tree}}(p)$; MakeSet$_{\text{node}}(p)$; $nodes[p]$:= MakeNode($F(p)$); $lowestNode[p] := p$;};
foreach $p \in V$ *in decreasing order of level for* F **do**
  curTree := Find$_{\text{tree}}(p)$;
  curNode := Find$_{\text{node}}(lowestNode$[curTree]);
  foreach *already processed neighbor* q *of* p *with* $F(q) \geq F(p)$ **do**
    adjTree := Find$_{\text{tree}}(q)$;
    adjNode := Find$_{\text{node}}(lowestNode$[adjTree]);
    if *(curNode $\neq$ adjNode)* **then**
      if *($nodes$[curNode]$\rightarrow$level == $nodes$[adjNode]$\rightarrow$level)* **then**
        curNode := MergeNodes(adjNode, curNode);
      else
        // We have $nodes$[curNode]$\rightarrow$level $<$ $nodes$[adjNode]$\rightarrow$level
        $nodes$[curNode]$\rightarrow$addChild($nodes$[adjNode]);
        $nodes$[curNode]$\rightarrow$area := $nodes$[curNode]$\rightarrow$area + $nodes$[adjNode]$\rightarrow$area;
        $nodes$[curNode]$\rightarrow$highest := max($nodes$[curNode]$\rightarrow$highest, $nodes$[adjNode]$\rightarrow$highest);
      curTree := Link$_{\text{tree}}$(adjTree, curTree);
      $lowestNode$[curTree] := curNode;

$Root := lowestNode$[Find$_{\text{tree}}$(Find$_{\text{node}}$(0))] ; → Root of the tree
foreach $p \in V$ **do** $M(p)$:= Find$_{\text{node}}(p)$; → Component mapping

PARP

Component tree algorithm (VIII)

- Auxiliary procedures:

```
Function node MakeNode(int level)
  Allocate a new node n with an empty list of children;
  n→ level := level; n→ area := 1; n→ highest := level;
  return n;
```

```
Function int MergeNodes(int node1, int node2)
  tmpNode := Link_node(node1,node2);
  if (tmpNode == node2) then
    Add the list of children of nodes[node1]
        to the list of children of nodes[node2];
    tmpNode2 := node1;
  else
    Add the list of children of nodes[node2]
        to the list of children of nodes[node1];
    tmpNode2 := node2;
  nodes[tmpNode]→area :=
      nodes[tmpNode]→area + nodes[tmpNode2]→area;
  nodes[tmpNode]→highest :=
       max(nodes[tmpNode]→highest,
           nodes[tmpNode2]→highest);
  return tmpNode;
```

PARP

DITEC DIS Universidad de Murcia

Component tree algorithm (IX)

- Some comments:
  - The ***LowestNode* array is not neccessary**: we can modify the code so that we store its contents as negative values in Par_{tree}.
  - **Another algorithm [Salembier et alt.]** could be teoretically faster in "normal" images, but it also has a quadratic *worst case* cost (in noisy images, with many local peaks). See also [Meijster et alt.].
  - Nodes can be augmented with several **interesting attributes**. For example:
    - height([k,c]) = $\max\{F(x)-k+1,\ x \in c\}$
    - area([k,c]) = $|c|$
    - volume([k,c]) = $\sum_{x\in c} (F(x)-k+1)$

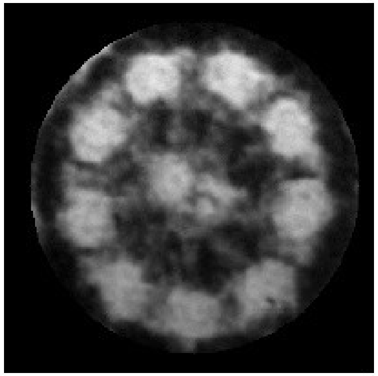
Computable while building the tree

Easily computable (using area) with recursive function

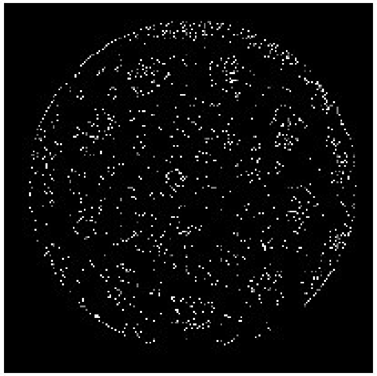
PARP

N most significant lobes (I)

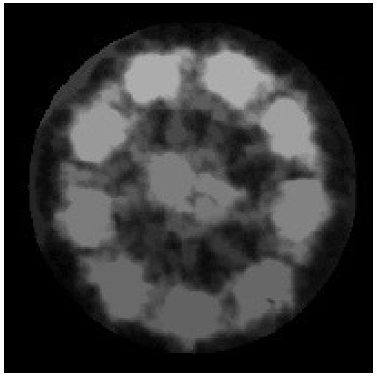
- We want to find the *N* **most significant components** with respect some attribute (for example, height, area or volume).
- Of course, components **should not be bounded with each other** by the inclusion relation.
- Example of application (10 most significant lobes, using volume):



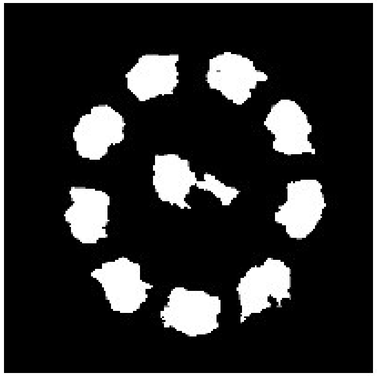
Original gray image



Maxima of gray image



Filtered image



Maxima of filtered image (corresponds to 10 most significant lobes)

N most significant lobes (II)

- Algorithm to filter the tree (pruning until only N leaves remain):

Algorithm 3: Keep_N_Lobes

Data: A vertex-weighted graph (V, E, F), its component tree T with attribute value for each node, and the associated component mapping M

Data: The number N of wanted lobes.

Result: The filtered map F

Sort the nodes of T by increasing order of attribute value;
$Q := \emptyset$; $L :=$ number of leaves in T;
forall n **do** $nodes$[n]→mark := 0;
while $L > N$ **do**
    Choose a (leaf) node c in T with smallest attribute value;
    p := $nodes$[c]→parent ;
    $nodes$[p]→nbChildren := $nodes$[p]→nbChildren-1;
    if *(nodes[p]→nbChildren > 0)* **then** L := L-1;
    $nodes$[c]→mark := 1 ; $Q := Q \cup \{c\}$;
while $\exists c \in Q$ **do**
    $Q := Q \setminus \{c\}$; RemoveLobe(c);
foreach $x \in V$ **do** $F(x) :=$ $nodes$[M[x]]→level;

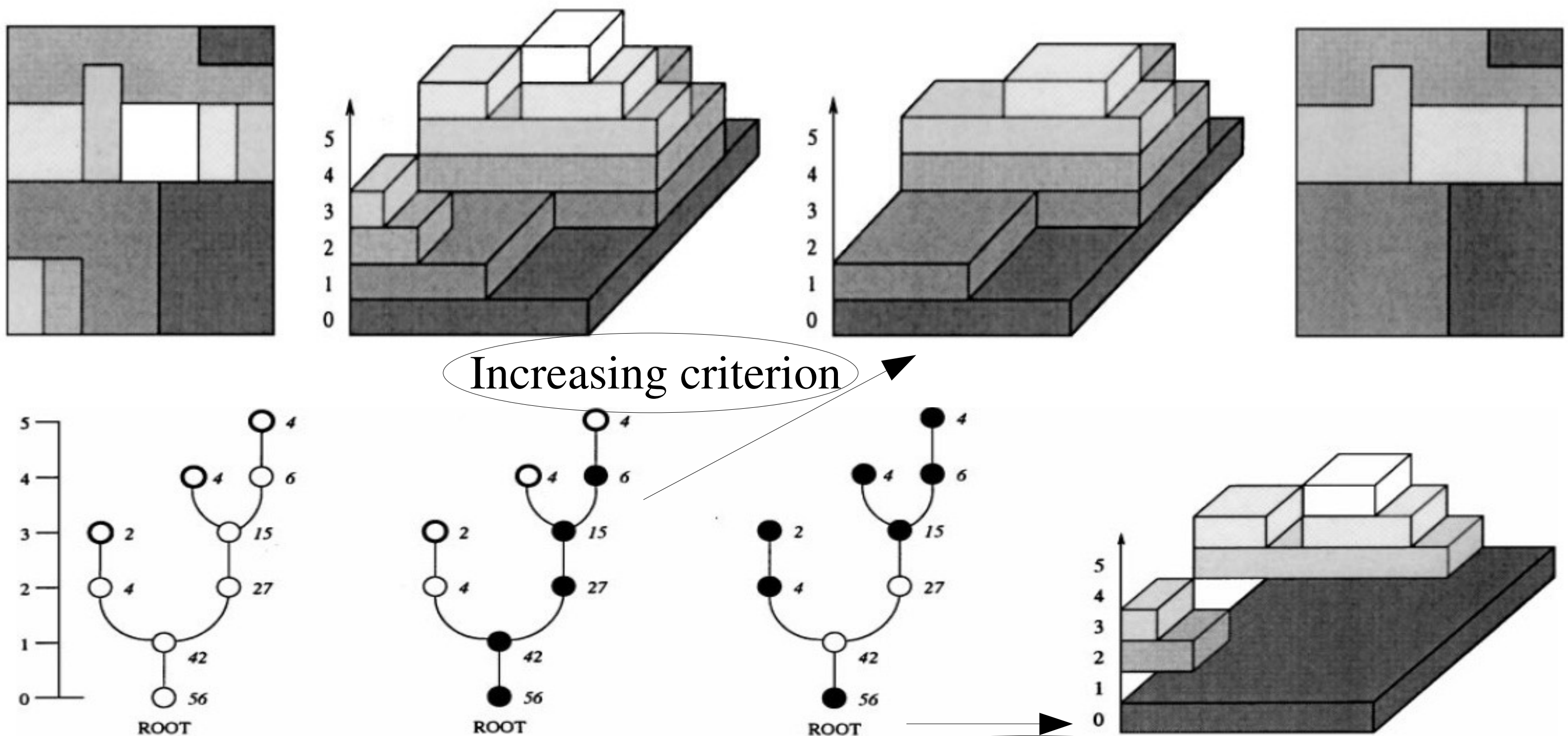
Function `int RemoveLobe` (*int n*)

if *(nodes[n]→mark == 1)* **then**
    $nodes$[n] := $nodes$[RemoveLobe($nodes$[n]→parent)];
return n;

PARP

Filtering using the component tree (I)

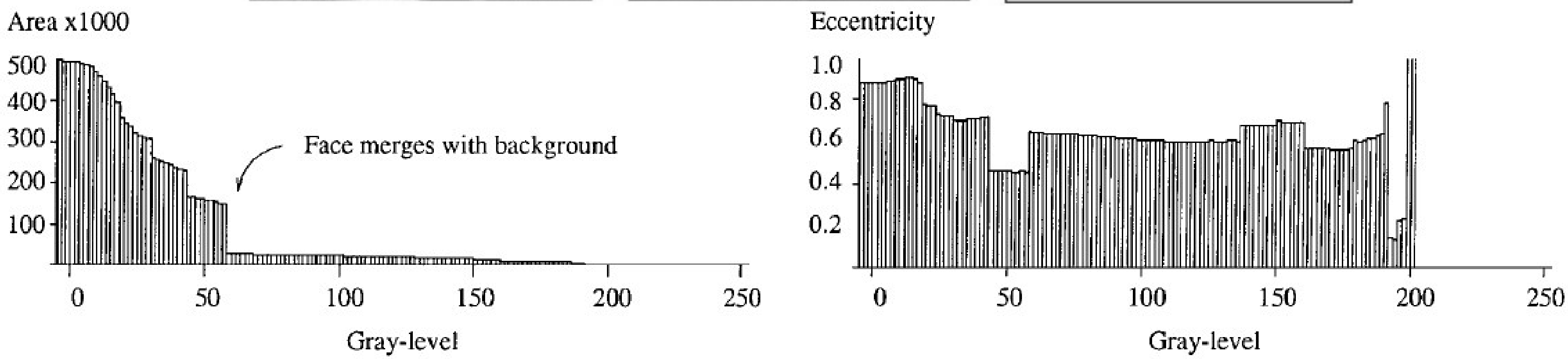
- ***Flat filters***
  - Cannot use links between components (applied on the binary image as result of thresholding; for example, using *area*):



- ***Non flat filters***
  - Can use link information between components (from *leave to root* → *attribute signature*; for example, using area;):

Filtering using the component tree (II)

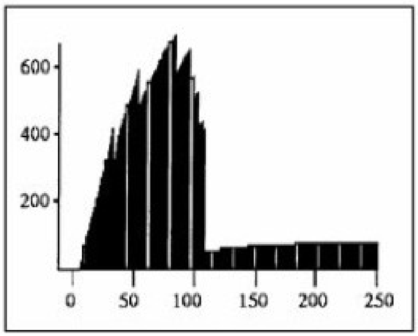
- Example
  - **Filtering & segmenting** Mona Lisa using a certain criterion on *area* and *eccentricity* (searching elliptical regions) signatures.



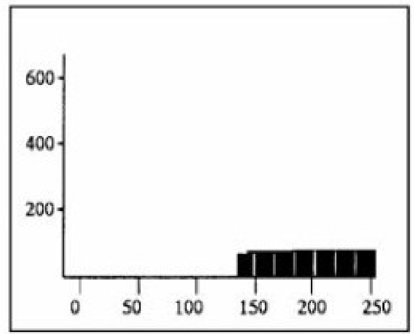
"Signature must contain a node with area between 5000-10000, and eccentricity 0.6-0.7"

Filtering using the component tree (III)

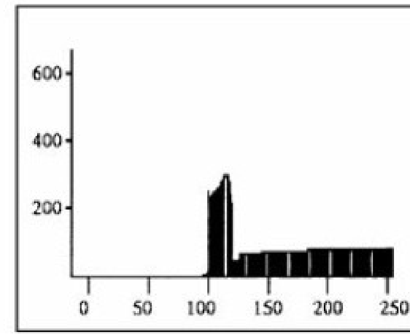
- Another example
  - Filtering *vessels in wood* texture using “area per regional minimum” signature:



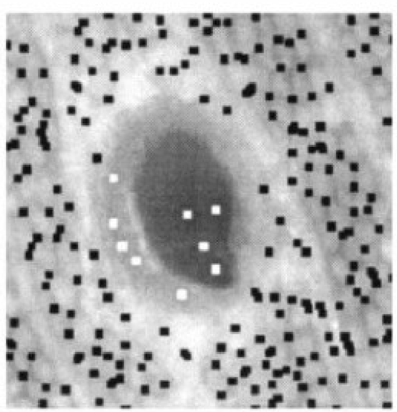
For regional minima INSIDE the vessel



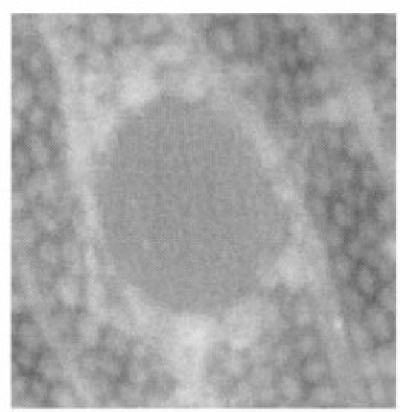
For regional minima OUTSIDE the vessel



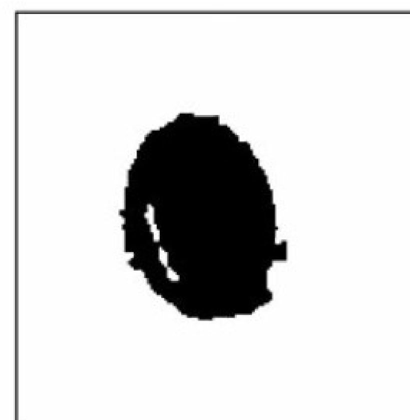
For regional minima in RING around the vessel



Input image with regional minima



Filtered image

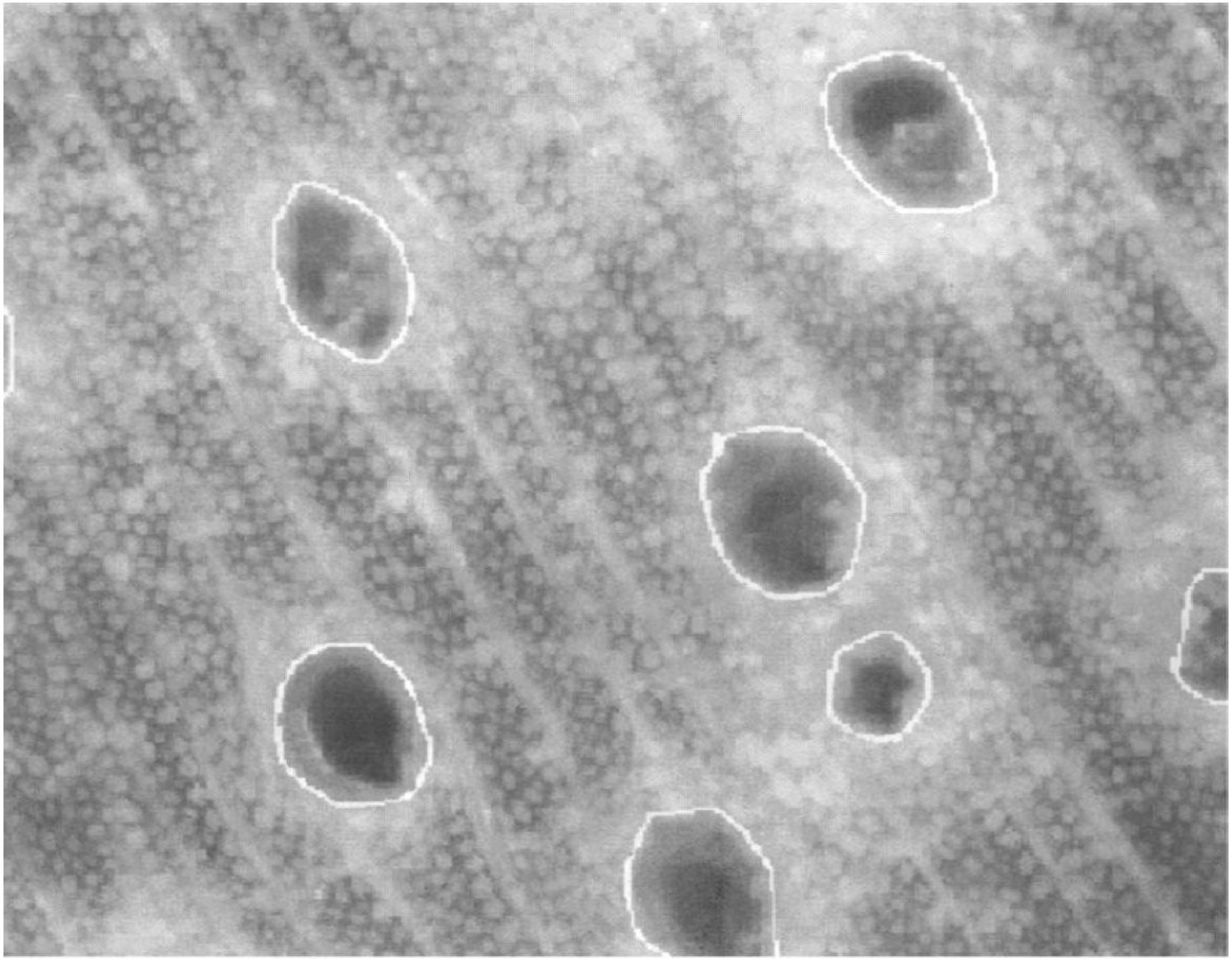


Segmented image

PARP

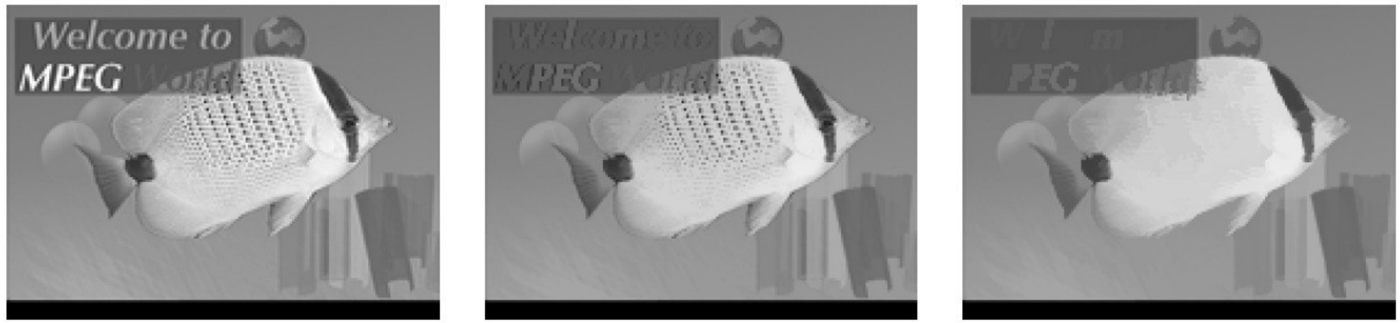
Filtering using the component tree (IV)

- Final result:



Filtering using the component tree (V)

- Some other *(flat)* criteria examples [Salembier et alt.]:
  - **Geometrical**: *simplicity* (ratio perimeter$^{\text{(squared or not)}}$/area):



A) Original Image B) Simplicity operator C) Dual operator

  - **Gray level**: contrast (λ-max), or *entropy*:

A) Original B) Entropy operator + dual C) Original D) Entropy operator + dual

  - **Sequences**: *motion* (using sequences of images).

PARP

Key ideas on tree filtering

- Merging of flat zones → Do **preserve contour information**
- The **definition of "flat zone" can be relaxed** (to cope with textured areas).
- **Leakage problem** (thin connections between different objets) → Could (in principle) be coped with during tree construction.
- **Restitution process**: for each image pixel, assign the gray level of the component it belongs to (but this could be changed, for example, trying to "substitute" the removed area using the gray levels "behind" the object).
- **Duality**: working on *Min-tree* as well as *Max-tree*.
- **Increasing criteria**: if region $X \subseteq Y$ (X descendant of Y), the criteria M(.) should (in principle) be monotonical ($M(X)<M(Y)$) (to be "stable").

PARP

DITEC DIS Universidad de Murcia

Tree matching

- Coming soon...

PARP

References

- L. Najman and M. Couprie, *"Building the component tree in quasi-linear time"*, IEEE Trans. on Image Processing, vol. 15, n. 11, November 2006.
- R. Jones, *"Connected Filtering and Segmentation Using Component Trees"*, CVIU, vol.75, n. 3, September 1999.
- P. Salembier, A. Oliveras and L. Garrido. *"Anti-extensive connected operators for image and sequence processing"*, IEEE Trans. on Image Processing, vol. 7, n. 4, April 1998.
- H. Nattesm, M. Richard and J. Demongeot, *"Tree representation for image matching and object recognition"*, DGCI'99, LNCS 1568, pp. 298-309, 1999.
- A. Meijster and M. Wilkinson, *"A comparison of algorithms for connected sets openings and closings"*, IEEE Trans. PAMI, vol. 24, n. 4, April 2002.

PARP

DITEC DIS Universidad de Murcia