Efficient Computation of Attributes and Saliency Maps on Tree-Based Image Representations

Yongchao Xu, Edwin Carlinet, Thierry Géraud, Laurent Najman

Efficient Computation of Attributes and Saliency Maps on Tree-Based Image Representations

Yongchao Xu[1,2], Edwin Carlinet[1,2], Thierry Géraud[1], Laurent Najman[2]

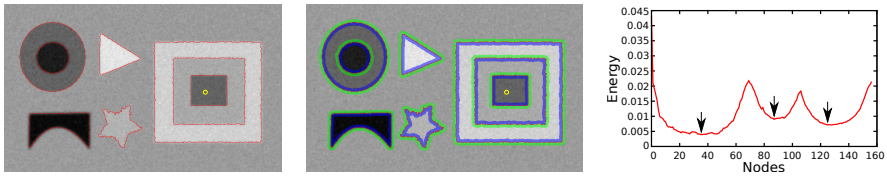
[1] EPITA Research and Development Laboratory (LRDE), France
{yongchao.xu, edwin.carlinet, thierry.geraud}@lrde.epita.fr
[2] Université Paris-Est, LIGM, Équipe A3SI, ESIEE, France
l.najman@esiee.fr

Abstract. Tree-based image representations are popular tools for many applications in mathematical morphology and image processing. Classically, one computes an attribute on each node of a tree and decides whether to preserve or remove some nodes upon the attribute function. This attribute function plays a key role for the good performance of tree-based applications. In this paper, we propose several algorithms to compute efficiently some attribute information. The first one is incremental computation of information on region, contour, and context. Then we show how to compute efficiently extremal information along the contour (*e.g.*, minimal gradient's magnitude along the contour). Lastly, we depict computation of extinction-based saliency map using tree-based image representations. The computation complexity and the memory cost of these algorithms are analyzed. To the best of our knowledge, except information on region, none of the other algorithms is presented explicitly in any state-of-the-art paper.

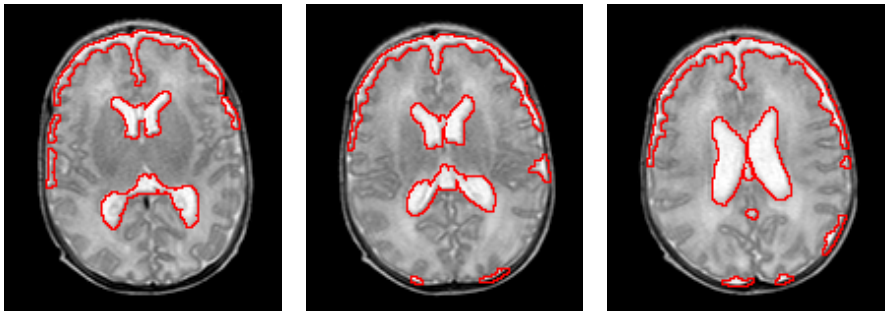
Keywords: Min/Max-tree, tree of shapes, algorithm, attribute, saliency map

1 Introduction

In a large number of applications, processing relies on objects or areas of interest. Therefore, region-based image representations have received much attention. In mathematical morphology, several region-based image representations have been popularized by attribute filters [2,17] or connected operators [13,14], which are filtering tools that act by merging flat zones. Such operators rely on transforming an image into an equivalent region-based representation, generally a tree of components (*e.g.*, the Min/Max-trees [13] or the tree of shapes [9]). Such trees are equivalent to the original image in the sense that the image can be reconstructed from the associated tree. Filtering then involves the design of an attribute function that weighs how important/meaningful a node of the tree is or how much a node of the tree fits a given shape. The filtering is achieved by preserving and removing some nodes of the tree according to the attribute function. This filtering process is either performed classically by thresholding the attribute function [14] or by considering the tree-based image representations as graphs and applying some filters on this graph representation [23,20].



(a) Illustration on a synthetical image. Green: exterior region; Blue: interior region.



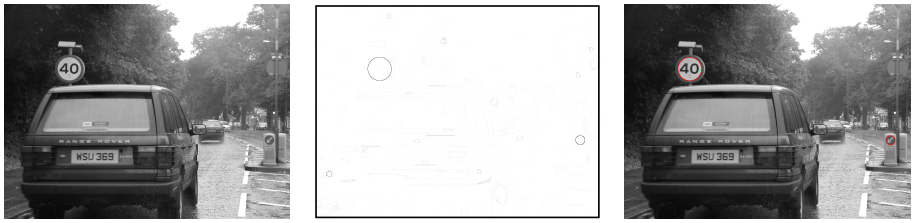
(b) Illustration of cerebrospinal fluid detection on MRI images of a newborn's brain.

Fig. 1. Examples of object detection using the context-based energy estimator [21] relying on contour and context information. An evolution of this attribute along a branch starting from the yellow point to the root is depicted on the right side of (a).

There exist many applications in image processing and computer vision that rely on tree-based image representations (see [20] for a short review). All these applications share a common scheme: one computes a tree representation and an attribute function upon which the tree analysis is performed. The choice of tree representation and the adequacy of attribute function mainly determine the success of the corresponding applications.

Many algorithms for computing different trees have been proposed (see Section 2.2 for a short review). In this paper, we focus on attribute computation, which is also an important step for the tree-based applications. To the best of our knowledge, only the algorithms for information computed on region have been presented [19] so far, none of the existing papers gives explicitly the algorithms computing the other attribute information employed in tree-based applications. In this paper, firstly, we detail explicitly how to incrementally compute some information on region, contour, and context. These informations form the basis for many classical attribute functions (*e.g.*, area, compactness, elongation). Let us remark that contextual information is very adequate for object detection, such as the context-based energy estimator [21] that relies on information computed on contour and context. Two examples of object detection using this attribute are shown in Fig. 1. Another type of interesting information is extremal information along the contour (*e.g.*, the minimal gradient's magnitude along the boundary). An example employing this information is the number of false alarms (NFA) for meaningful level lines extraction [7,3]. Here we propose an efficient algorithm that does not require much memory to compute this kind

(a) Extinction-based saliency map using color tree of shapes [5].



(b) Circular object oriented extinction-based saliency map.

Fig. 2. Illustrations of extinction-based saliency maps from the tree of shapes.

of information. Lastly, we depict an algorithm computing the extinction-based saliency map [20] representing a hierarchical morphological segmentation using tree-based image representations (two examples are illustrated in Fig. 2). These algorithms form the main contribution of this paper.

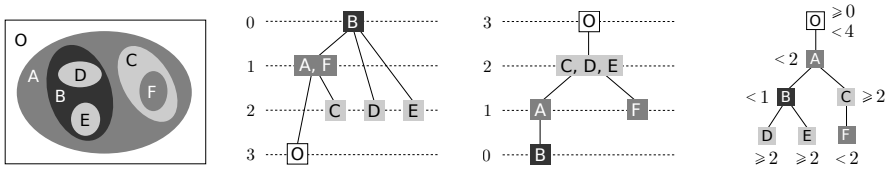
The rest of the paper is organized as follows: A short review of some tree-based image representations and their computations using immersion algorithm are provided in Section 2. Our proposed algorithms to compute some attribute information and saliency maps are detailed in Section 3, and we analyze in Section 4 the complexity and the memory cost of the proposed algorithms. Finally, we conclude and give some perspectives in Section 5.

2 Review of Morphological Trees and their Computations

Region-based image representations are composed of a set of regions of the original image. Those regions are either disjoint or nested, and they are organized into a tree structure thanks to the inclusion relationship. There are two types of such representations: fine to coarse hierarchical segmentations and threshold decomposition-based trees. In this paper, we only consider the threshold decomposition-based trees.

2.1 Tree-based Image Representations

Let f be an image defined on domain Ω and with values on ordered set V (typically $\mathbb{R}$ or $\mathbb{Z}$). For any $\lambda \in V$, the upper level sets $\mathcal{X}_\lambda$ and lower level sets $\mathcal{X}^\lambda$ of an image f are respectively defined by $\mathcal{X}_\lambda(f) = \{p \in \Omega \mid f(p) \geq \lambda\}$ and $\mathcal{X}^\lambda(f) = \{p \in \Omega \mid f(p) \leq \lambda\}$. Both the upper and lower level sets have a natural inclusion structure: $\forall\, \lambda_1 \leq \lambda_2,\ \mathcal{X}_{\lambda_1} \supseteq \mathcal{X}_{\lambda_2}$ and $\mathcal{X}^{\lambda_1} \subseteq \mathcal{X}^{\lambda_2}$, which leads to two distinct and dual representations of the image: Max-tree and Min-tree [13]. The tree of shapes is a fusion of the Max-tree and Min-tree via the



(a) Input image. **(b)** Max-tree. **(c)** Min-tree. **(d)** Tree of shapes.

Fig. 3. Tree-based image representations relying on threshold decompositions.

notion of *shapes* [9]. A shape is defined as a connected component of an upper or lower level set with its holes filled in. Thanks to the inclusion relationship of both kinds of level sets, the set of shapes can be structured into a tree structure, called the tree of shapes. An example of these trees is depicted in Fig. 3.

2.2 Tree Computation and Representation

There exist three types of algorithms to compute the Min/Max-tree (see [4] for a complete review): flooding algorithms [13,18,11], merge-based algorithms [19,12], and immersion algorithms [1,10]. In this paper, we employ the immersion algorithm to construct the Min/Max-tree. Concerning the tree of shapes [9], there are four different algorithms [9,15,6,8]. We use the one proposed by Géraud *et al.* [8]. It is similar to the immersion algorithms used for the Min/Max-tree computation. All these trees feature a common scheme of process: they start with considering each pixel as a singleton and sorting the pixels in decreasing tree order (*i.e.*, root to leaves order), followed by an union-find process (in reverse order) to merge disjoint sets to form a tree structure.

Let $\mathcal{R}$ be the vector of the N sorted pixels, and $\mathcal{N}(p)$ be neighbors (*e.g.*, 4- or 8-connectivity) of the pixel p. The union-find process is then depicted in Fig. 4 (a), where *parent* and *zpar* are respectively the parenthood image and the root path compression image. The whole process of tree computation is given in Fig. 4 (b), where SORT_PIXELS is a decreasing tree order sorting. The algorithms for computing the Min/Max-tree and the tree of shapes differ in this pixel sorting step. For the Min/Max-tree, they are either sorted in decreasing order (Min-tree) or increasing order (Max-tree). If the image f is low quantized, we can use the Bucket sort algorithm to sort the pixels. Concerning the tree of shapes, the sorting step is more complicated. It first interpolates the scalar image to an image of range using a simplicial version of the 2D discrete grid: the Khalimsky grid as shown in Fig. 5. We note $\mathcal{K}_\Omega$, the domain Ω immersed on this grid. In Fig. 5 (a), the original points of the image are the 2-faces, the boundaries are materialized with 0-faces and 1-faces. The algorithm in [8] ensures that shapes are open connected sets (*e.g.*, the purple shape in Fig. 5 (a)) and that shapes' borders are composed of 0-faces and 1-faces only (*e.g.*, the dark curve in Fig. 5 (a)). We refer the interested reader to the work of Géraud *et al.* [8] for more details on this pixel sorting step.

The tree structure is encoded through the image $parent : \Omega \to \Omega$ or $\mathcal{K}_\Omega \to \mathcal{K}_\Omega$ that states the parenthood relationship between nodes. In *parent*, a node is

```
1  FIND_ROOT(zpar, x)
2  if zpar(x) = x then return x
3  else
4  |  zpar(x) ← FIND_ROOT(zpar, zpar(x));
5  |  return zpar(x)
6  UNION_FIND(R)
7  for all p do zpar(p) ← undef;
8  for i ← N − 1 to 0 do
9  |  p ← R[i], parent(p) ← p, zpar(p) ← p;
10 |  for all n ∈ N(p) if zpar(n) ≠ undef do
11 |  |  r ← FIND_ROOT(zpar, n);
12 |  |  if r ≠ p then
13 |  |  |  parent(r) ← p, zpar(r) ← p;
14 return parent
```

(a) Union-find process.

```
1  CANONIZE_T(f, R, parent)
2  for i ← 0 to N − 1 do
3  |  p ← R[i];
4  |  q ← parent(p);
5  |  if f(parent(q)) = f(q)
   |  then
6  |  |  parent(p) ← parent(q);
7  return parent
8  COMPUTE_TREE(f)
9  R ← SORT_PIXELS(f);
10 parent ← UNION_FIND(R);
11 parent ←
   CANONIZE_T(f, R, parent);
12 return parent
```

(b) Complete tree construction.

Fig. 4. Tree construction relying on union-find process.

represented by a single pixel (a 2-face of the Khalimsky grid in the case of the tree of shapes) called the canonical element, and each non-canonical element is attached to the canonical element representing the node it belongs to. In the following, we denote by getCanonical : $\Omega \to \Omega$ or $\mathcal{K}_\Omega \to \mathcal{K}_\Omega$, the routine that returns the canonical element of each point in the image.

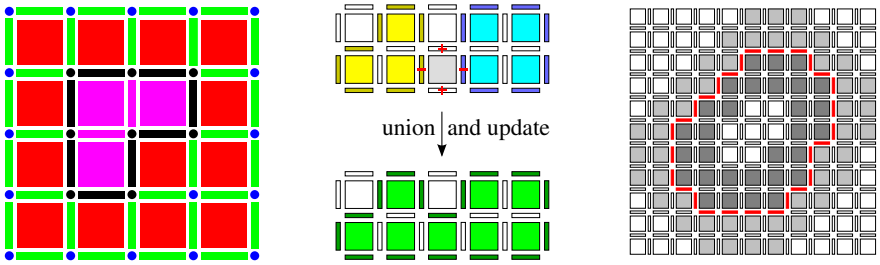
3 Proposed Algorithms

In this section, we detail several algorithms related to some applications using tree-based image representations, including computation of some classical information used in many attribute functions (accumulated information in Section 3.1, and extremal information along the contour in Section 3.2), and computation of extinction-based saliency maps [20] in Section 3.3. For the sake of simplicity, we consider the Min-tree or Max-tree representation. The algorithms for the tree of shapes construction share the same principle.

3.1 Incremental Computation of Some Accumulated Information

There are three main types of accumulated information: computed on region A (*e.g.*, area), on contour L (*e.g.*, length), and on context X (interior context X^i or exterior context X^e).

Attributes computed on regions. During the tree construction process, the algorithm starts with the pixels lying on the leaves, and the union-find acts as a region merging process. The connected components in the tree are built during this region growing process. We are able to handle information computed on region efficiently, such as its size, the sum of gray level or sum of square of



(a) Khalimsky grid. **(b)** Updating contour. **(c)** Regional context.

Fig. 5. (a): A point in a 2D image is materialized with 0-faces (blue disks), 1-faces (green strips), and 2-faces (red squares). (b): Updating contour information when an union between two components (yellow and blue) occurs thanks to a pixel (gray). (c): The approximated interior and exterior regional context of the red level line is respectively the dark gray region and the light gray region.

gray level that can be used to compute the mean and the variance inside each region, the moments of each region based on which we can compute some shape attribute that measures how much a node fits a specific pattern. The algorithm for computing these information is depicted in Fig. 6 by adding some additional operations (red lines) to the union-find process during the tree construction, where i_A encodes information on pixels (*i.e.*, 2-faces). For example if A is the size or the sum of gray level, then i_A would be 1 (size of a pixel) or the pixel value. The operator $\widehat{+}$ is a binary commutative and associative operator having a neutral element $\widehat{0}$ [19]. For example, if A is the size, then the operator $\widehat{+}$ and $\widehat{0}$ would be the classical operator $+$ and 0 for the initialization.

Attributes computed on contours. Attribute functions relying on contour-related information are also very common, such as average of gradient's magnitude along the contour. Information accumulated on contour can be managed in the same way as information computed on region. The basic idea is that during the union-find process, every time a pixel p is added to the current region to form a parent region, process the four 1-faces which are the four neighbors (4-connectivity) of the current pixel (*i.e.*, 2-face in the Khalimsky grid in Fig. 5 (a)). If a 1-face e is already added to the current region (*i.e.*, belongs to its boundary), then remove e after adding p, since that 1-face e will be inside the parent region, consequently it is no longer on the boundary. Otherwise, add this 1-face e. This process is illustrated in Fig. 5 (b). It relies on an image *is_boundary* defined on the 1-faces that indicates if the 1-face belongs to the boundary of some region. Information on contour is computed by adding some supplementary process (green and gray lines in Fig. 6) to the union-find process, where i_L encodes information defined on 1-faces. For example if L is the contour length or the sum of gradient's magnitude, then i_L would be 1 (size of a 1-face) or the gradient's magnitude on the 1-faces. The operator $\widehat{-}$ is the inverse of the operator $\widehat{+}$.

Attributes computed on contexts. In [21], we have presented a context-based energy estimator that is adequate for object detection (see Fig. 1 for some

```
UNION_FIND(R)
for all p do
    zpar(p) ← undef;
    A(p) ← 0̂; //information computed on region (e.g., area, sum of gray level)
    L(p) ← 0̂; //information computed on contour (e.g., contour length)
    X^i(p) ← 0̂, X^e(p) ← 0̂; //information computed on context
    V_L(p) ← M̂; //extremal information along the contour
for all e do is_boundary(e) ← false;
for i ← N − 1 to 0 do
    p ← R[i], parent(p) ← p, zpar(p) ← p;
    A(p) ← A(p) +̂ i_A(p); //i_A: information on pixels (i.e., 2-faces)
    for all n ∈ N(p) such as zpar(n) ≠ undef do
        r ← FIND_ROOT(zpar, n);
        if r ≠ p then
            parent(r) ← p, zpar(r) ← p;
            A(p) ← A(p) +̂ A(r);
            L(p) ← L(p) +̂ L(r);
            X^i(p) ← X^i(p) +̂ X^i(r), X^e(p) ← X^e(p) +̂ X^e(r);
    for all e ∈ N_4(p) do
        if not is_boundary(e) then
            is_boundary(e) ← true;
            L(p) ← L(p) +̂ i_L(e); //i_L: information on 1-faces
            //i_X^tr and i_X^dl: top-right and down-left context of 1-faces
            if e is above or on the right of p then
                X^i(p) ← X^i(p) +̂ i_X^dl(e), X^e(p) ← X^e(p) +̂ i_X^tr(e);
            else X^i(p) ← X^i(p) +̂ i_X^tr(e), X^e(p) ← X^e(p) +̂ i_X^dl(e);
            appear(e) ← p;
        else
            is_boundary(e) ← false;
            L(p) ← L(p) −̂ i_L(e);
            if e is above or on the right of p then
                X^i(p) ← X^i(p) −̂ i_X^tr(e), X^e(p) ← X^e(p) −̂ i_X^dl(e);
            else X^i(p) ← X^i(p) −̂ i_X^dl(e), X^e(p) ← X^e(p) −̂ i_X^tr(e);
            vanish(e) ← p;
for all e do
    N_a ← appear(e), N_v ← vanish(e);
    while N_a ≠ N_v do
        V_L(N_a) ← update(V_L(N_a), i_L(e)); //update: either min or max
        N_a ← parent(N_a);
return parent
```

Fig. 6. Incremental computation of information on region (in red), contour (in green), and context (in blue). The computation of extremal information is in magenta. The black lines represent the original union-find process, and the gray lines are used for the computation of contour, context, and extremal information.

examples). It relies on regional context information. The interior and exterior contextual region of a given region S (*e.g.*, a shape) is defined as the set of pixels respectively inside and outside the region with a distance to the boundary less than a given threshold ε. More formally, given a ball B_ε of radius ε, the exterior and interior of the shape S are defined as $Ext_B(S) = \delta_B(S) \setminus S$ and $Int_B(S) = S \setminus \epsilon_B(S)$ where δ and ϵ denote the dilation and erosion.

An approximated interior and exterior contextual region is illustrated in Fig. 5 (c) with $\varepsilon = 2$. As shown in this figure, we approximate the interior region and the exterior region of each level line by only taking into account the pixels which are aligned perpendicularly to each 1-face of the level line. Note that some pixels may be counted several times. Information on context can be computed in the same way as information on contour. But one has to attend closely to interior and exterior information while doing the update operation. The algorithm for computing interior (resp. exterior) contextual information X^i (resp. X^e) is shown in Fig. 6 by adding the gray and blue lines to the union-find process. This algorithm relies on two pre-computed images defined on 1-faces: i_X^{tr} and i_X^{dl} that encode information of ε pixels above (horizontal 1-face) or on the right side (vertical 1-face) of e, and respectively below (horizontal 1-face) or on the left side (vertical 1-face) of e.

Contextual information can be retrieved exactly at cost of a higher computation complexity. For every point p, we aim at finding all the shapes for which p is in the interior or the exterior. Given two points p and q such that $q \in B(p)$, we note S_p and S_q their respective shapes (nodes). We also note $Anc = LCA(S_p, S_q)$ where LCA stands for the least common ancestor of the two nodes and finally, let $[A \rightsquigarrow B) = \{S \mid A \subseteq S \subset B\}$ denotes the path from A to B in the tree. For all shapes $S \in [S_p \rightsquigarrow LCA(S_p, S_q))$, we have $p \in S$, but $q \notin S$, thus $p \in Int_B(S)$ and $q \in Ext_B(S)$ (see Fig. 7). The algorithms in Fig. 8 use the above-mentioned idea to compute contextual information, where i_X stands for information on pixels. A set of nodes $DjVu$ is used to track the shapes for which the current point has already been considered. If for neighbors q_1 and q_2, $[S_p \rightsquigarrow LCA(S_p, S_{q1}))$ and $[S_p \rightsquigarrow LCA(S_p, S_{q2}))$ have shapes in common, they will not be processed twice.

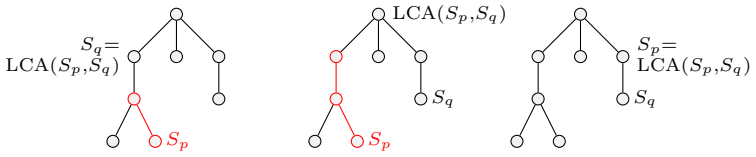


Fig. 7. Three cases for contextual computation. p and q are two neighbors (w.r.t. B). The red path denotes the nodes in $[S_p \rightsquigarrow LCA(S_p, S_q))$ for which p is in the interior and q in the exterior. Left: case $S_p \subset S_q$, middle: case S_p and S_q are in different paths, right: case $S_q \subset S_p$.

```
1  EXTERNAL_CONTEXT(parent)
2  foreach node x do X^e(x) ← 0̂;
3  foreach point q in Ω do
4  |  DjVu ← ∅;
5  |  foreach point p in B_ε(q) do
6  |  |  N_p ← getCanonical(p);
7  |  |  N_q ← getCanonical(q);
8  |  |  Anc ← LCA(N_p, N_q);
9  |  |  while N_p ≠ Anc do
10 |  |  |  if N_p ∉ DjVu then
11 |  |  |  |  X^e(N_p) ←
   |  |  |  |  X^e(N_p) ∔̂ i_X(q);
12 |  |  |  DjVu ← DjVu ∪ {N_p};
13 |  |  |  N_p ← parent(N_p);
14 return X^e
```

```
1  INTERNAL_CONTEXT(parent)
2  foreach node x do X^i(x) ← 0̂;
3  foreach point p in Ω do
4  |  DjVu ← ∅;
5  |  foreach point q in B_ε(p) do
6  |  |  N_p ← getCanonical(p);
7  |  |  N_q ← getCanonical(q);
8  |  |  Anc ← LCA(N_p, N_q);
9  |  |  while N_p ≠ Anc do
10 |  |  |  if N_p ∉ DjVu then
11 |  |  |  |  X^i(N_p) ←
   |  |  |  |  X^i(N_p) ∔̂ i_X(p);
12 |  |  |  DjVu ← DjVu ∪ {N_p};
13 |  |  |  N_p ← parent(N_p);
14 return X^i
```

Fig. 8. Algorithms for exact computation of contextual information X^i and X^e.

3.2 Computation of Extremal Information along the Contour

Apart from those attributes based on accumulated information, the number of false alarms (NFA) [7,3] (see [3] for several examples of meaningful level lines selection using NFA) requires to compute the minimal gradient's magnitude along the boundary of each region. Here we propose an efficient algorithm that requires low memory to handle this extremal information along the contour V_L. It relies on two images *appear* and *vanish* defined on the 1-faces. $appear(e)$ encodes the smallest region $\mathcal{N}_a$ in the tree for which the 1-face e lies on its boundary, while $appear(e)$ stands for the smallest region $\mathcal{N}_v$ for which e is inside it. Note that $\mathcal{N}_a$ and $\mathcal{N}_v$ might be equal, *e.g.*, in the case of 1-faces in the interior of a flat zone. The computation of extremal information along the contour V_L is depicted in Fig. 6 by adding the gray and magenta lines to the union-find process, where $\widehat{M}$ in the initialization step is the maximal (resp. minimal) value for minimal (resp. maximal) information computation, and the operator "update" is a "min" (resp. "max") operator for the minimal (resp. the maximal) information.

3.3 Computation of the Saliency Map

As shown in [22,20], the saliency map introduced in the framework of shape-based morphology relies on the extinction values $\mathcal{E}$ defined on the local minima [16]. Once the extinction values computed for all the minima (see [16] for details about the computation of the extinction values COMPUTE_EXTINCTION), we can weigh the extinction values on the region boundaries corresponding to the minima. Each 1-face takes the maximal extinction value of those minima for which this 1-face is on their boundaries. This can be achieved via two images *appear* and *vanish* that have been used in the computation of extremal information along the contour (as shown in Fig. 6). For each 0-face o, it takes the

```
1  COMPUTE_SALIENCY_MAP(f)
2  (T, A) ← COMPUTE_TREE(f);
3  E ← COMPUTE_EXTINCTION(T, A);
4  for all e do M_E(e) ← 0;
5  for all e do
6  |  N_a ← appear(e), N_v ← vanish(e);
7  |  while N_a ≠ N_v do
8  |  |  M_E(N_a) ← max(E(N_a), M_E(e)), N_a ← parent(N_a);
9  for all 0-face o do M_E(o) ← max(M_E(e_1), M_E(e_2), M_E(e_3), M_E(e_4));
10 return M_E
```

Fig. 9. Computation of extinction-based saliency map $\mathcal{M}_\mathcal{E}$.

maximal value among the four 1-faces $e_1, e_2, e_3,$ and e_4 that are neighbors (4-connectivity) of o in the Khalimsky grid. Finally, the extinction-based saliency map $\mathcal{M}_\mathcal{E}$ is obtained. The computation of the saliency map is given in Fig. 9.

4 Complexity Analysis

We use the algorithms based on the Tarjan's Union-Find process to construct the Min-tree and Max-tree [10,1,4] and the tree of shapes [8]. These approaches would take $O(n \log(n))$ time, where n is the number of pixels of the image f. For low quantized images (typically 12-bit images or less), the complexity of the computation of these trees is $O(n\,\alpha(n))$, where α is a very slow-growing diagonal inverse of the Ackermann's function. In this section, we analyze the additional complexity and the memory usage of the algorithms proposed in Section 3.

4.1 Accumulated Information on Region, Contour, and Context

As described in Section 3.1 and shown in Fig. 6, information computed on regions, contours, and contexts (the approximated version) are computed incrementally during the union-find process. Consequently, they have the same complexity as the union-find which is $O(n\,\alpha(n))$. Besides, the pre-computed images (*e.g.*, i_L or i_X^{tr}) can be obtained in linear time, so the $O(n\,\alpha(n))$ complexity is maintained. To compute exactly contextual information as described in Fig. 8, for each pixel p, we have to compute the least common ancestor Anc of p and any $q \in B_\varepsilon(p)$ and propagate from N_p to Anc. The computation of the least common ancestor has a $O(h)$ complexity if a depth image is employed, where h is the height of the tree. Consequently, the total complexity is $O(n\varepsilon^2 h)$.

Apart from the necessary memory of the union-find process, the computation of information on regions does not require auxiliary memory. For information computed on contours and contexts (approximated), the auxiliary memory usage is $4n$ for the intermediate image *is_boundary* (defined on the Khalimsky grid). For the exact computation of contextual information, we need the depth image (n pixels) used by the least common ancestor algorithm and the intermediate set *DjVu* ($O(h)$ elements). The total auxiliary memory cost is thus $n + h$.

4.2 Extremal Information along the Contour

The algorithm computing extremal information along the contour relies on two auxiliary images *appear* and *vanish*. As described in Section 3.2 and shown in Fig. 6, these two images are computed incrementally during the union-find process. The complexity of this step is $O(n\,\alpha(n))$. Then, to compute the final extremal information, for each 1-face e, we have to propagate the value to a set of node (from $appear(e)$ to $vanish(e)$). In the worst case, we have to traverse the whole branch of the tree. Consequently, the complexity would be $O(nh)$. In terms of auxiliary memory cost, it would take $4n$ for each intermediate image *appear*, *vanish*, and *is_boundary*. So the total additional memory cost would be $12n$. Such extra cost is acceptable for 2D cases, but become prohibitive for very large or 3D images. Actually, we could avoid the extra-memory used for the storage of *appear* and *vanish* as the information they provide could be computed on the fly in each algorithm. Nevertheless, for the purpose of clarification, we have chosen to compute these information one for all to avoid code redoundancy in the algorithms we have proposed.

4.3 Saliency Map

The computation of extinction-based saliency map given in Section 3.3 and depicted in Fig. 9 also relies on the two temporary images *appear* and *vanish*. Suppose that we have the extinction values $\mathcal{E}$ for all the local minima. In the same way as the computation of extremal information along the contour, for each 1-face e, we have to propagate from $appear(e)$ to $vanish(e)$. The worst time complexity would be $O(nh)$. The computation of extinction values $\mathcal{E}$ relies on a Max-tree computation process, which is quasi-linear. The auxiliary memory cost would be $12n$ ($4n$ for each temporary image *appear*, *vanish*, and *is_boundary*). Yet, the remark about the memory usage given in Section 4.2 holds for this complexity analysis.

5 Conclusion

In this paper, we have pesented several algorithms related to some applications using tree-based image representations. First of all, we have shown how to incrementally compute information on region, contour, and context which forms the basis of many widely used attribute functions. Then we have proposed an algorithm in order to compute extremal information along the contour (required for some attribute functions, such as the number of false alarms (NFA)), which requires few extra memory. Finally, we have depicted how to compute extinction-based saliency maps from tree-based image representations. The time complexity and the memory cost of these algorithms are also analyzed. To the best of our knowledge, this is the first time that these algorithms (except for information computed on region) are explicitly depicted, which allows reproducible research and facilitates the development of some novel interesting attribute functions. In the future, extension of these algorithms to 3D images will be studied. And we would like to study some more attribute functions: learning attribute functions in particular would be one interesting future work.

References

1. Berger, C., Géraud, T., Levillain, R., Widynski, N., Baillard, A., Bertin, E.: Effective component tree computation with application to pattern recognition in astronomical imaging. In: Proc. of IEEE ICIP. vol. 4, pp. 41–44 (2007)
2. Breen, E., Jones, R.: Attribute openings, thinnings, and granulometries. CVIU 64(3), 377–389 (1996)
3. Cao, F., Musé, P., Sur, F.: Extracting meaningful curves from images. JMIV 22, 159–181 (2005)
4. Carlinet, E., Géraud, T.: A comparative review of component tree computation algorithms. IEEE Transactions on Image Processing 23(9), 3885–3895 (Sep 2014)
5. Carlinet, E., Géraud, T.: A color tree of shapes with illustrations on filtering, simplification, and segmentation. Submitted for publication (2015)
6. Caselles, V., Monasse, P.: Geometric Description of Images as Topographic Maps. Springer Publishing Company, Incorporated, 1st edn. (2009)
7. Desolneux, A., Moisan, L., Morel, J.: Edge detection by helmholtz principle. JMIV 14(3), 271–284 (2001)
8. Géraud, T., Carlinet, E., Crozet, S., Najman, L.: A quasi-linear algorithm to compute the tree of shapes of nd images. In: ISMM. pp. 98–110 (2013)
9. Monasse, P., Guichard, F.: Fast computation of a contrast-invariant image representation. IEEE Trans. on Image Processing 9(5), 860–872 (2000)
10. Najman, L., Couprie, M.: Building the component tree in quasi-linear time. IEEE Trans. on Image Processing 15(11), 3531–3539 (2006)
11. Nistér, D., Stewénius, H.: Linear time maximally stable extremal regions. In: ECCV. pp. 183–196. Springer-Verlag, Berlin, Heidelberg (2008)
12. Ouzounis, G.K., Wilkinson, M.H.F.: A parallel implementation of the dual-input max-tree algorithm for attribute filtering. In: ISMM. pp. 449–460 (2007)
13. Salembier, P., Oliveras, A., Garrido, L.: Antiextensive connected operators for image and sequence processing. ITIP 7(4), 555–570 (1998)
14. Salembier, P., Wilkinson, M.H.F.: Connected operators. IEEE Signal Processing Mag. 26(6), 136–157 (2009)
15. Song, Y.: A topdown algorithm for computation of level line trees. IEEE Transactions on Image Processing 16(8), 2107–2116 (Aug 2007)
16. Vachier, C., Meyer, F.: Extinction values: A new measurement of persistence. IEEE Workshop on Non Linear Signal/Image Processing pp. 254–257 (1995)
17. Westenberg, M.A., Roerdink, J.B.T.M., Wilkinson, M.H.F.: Volumetric attribute filtering and interactive visualization using the max-tree representation. ITIP 16(12), 2943–2952 (2007)
18. Wilkinson, M.H.F.: A fast component-tree algorithm for high dynamic-range images and second generation connectivity. In: Proc. of ICIP. pp. 1021–1024 (2011)
19. Wilkinson, M.H.F., Gao, H., Hesselink, W.H., Jonker, J.E., Meijster, A.: Concurrent computation of attribute filters on shared memory parallel machines. PAMI 30(10), 1800–1813 (2008)
20. Xu, Y.: Tree-based shape spaces: Definition and applications in image processing and computer vision. Ph.D. thesis, Université Paris Est, Marne-la-Vallée, France (Dec 2013)
21. Xu, Y., Géraud, T., Najman, L.: Context-based energy estimator : Application to object segmentation on the tree of shapes. In: ICIP. pp. 1577–1580. IEEE (2012)
22. Xu, Y., Géraud, T., Najman, L.: Two applications of shape-based morphology: Blood vessels segmentation and a generalization of constrained connectivity. In: ISMM. Lecture Notes in Computer Science, vol. 7883, pp. 390–401 (2013)
23. Xu, Y., Géraud, T., Najman, L.: Morphological Filtering in Shape Spaces: Applications using Tree-Based Image Representations. In: ICPR. pp. 485–488 (2012)