Component-Tree Simplification through Fast Alpha Cuts

Wilkinson, M.H.F.

Published in:
Discrete Geometry and Mathematical Morphology 2022

Component-Tree Simplification through Fast Alpha Cuts

Michael H. F. Wilkinson

Bernoulli Institute for Mathematics, Computer Science, and Artifical Intelligence, University of Groningen, P.O. Box 407, 9700 AK Groningen, The Netherlands
m.h.f.wilkinson@rug.nl

Abstract. Tree-based hierarchical image representations are commonly used in connected morphological image filtering, segmentation and multi-scale analysis. In the case of component trees, filtering is generally based on thresholding single attributes computed for all the nodes in the tree. Alternatively, so-called shapings are used, which rely on building a component tree of a component tree to filter the image. Neither method is practical when using vector attributes. In this case, more complicated machine learning methods are required, including clustering methods. In this paper I present a simple, fast hierarchical clustering algorithm based on cuts of α-trees to simplify and filter component trees.

Keywords: Connected filters, component trees, α-trees, clustering, algorithms

1 Introduction

Connected filters [1–3] have found many uses in image processing and analysis, and many different types of filters and multi-scale tools have been developed since the introduction of the first connected filters in the form of openings by reconstruction [4], and area openings [5, 6]. Many methods are built using hierarchical image representations in the form of tree structures [7–10], for a recent review see [11]. In the grey-scale case, much work has been done on attribute filters [12], in particular using tree structures variously known as component trees [9], min-trees and max-trees [7]. They have also found use for multi-scale analysis, e.g. through pattern spectra [13] or morphological profiles [14].

This paper will focus on component trees, which are trees containing the connected components of threshold sets of a grey-scale image. Each node represents a single connected component, and usually contains some scalar attribute value like area or elongation to characterise the component. Filtering is done by applying some threshold to the attributes, and removing nodes that have attribute values lower than the threshold.

Selecting the "right" threshold for attribute filtering is not an easy task. Apart from simple trial and error, only a few papers address this issue systematically. Jones [9] notes that threshold on attributes could be chosen automatically by traversing the tree from leaf to root, and choosing thresholds at points where the attribute value changes abruptly. Thresholds could be chosen per branch, or globally in the tree. In [15], various automatic grey-scale thresholding methods are studied, but these do not always take the topology of the tree into account.

In the case of increasing attributes like area, filtering essentially boils down to pruning the tree such that the remaining leaves have an area larger than some threshold. For non-increasing attributes, such as elongation or perimeter, several other filtering strategies have been proposed [7, 13]. Xu et al. [16] introduced the idea of tackling non-increasing scalar attributes in max-trees by computing a max-tree of a max-tree. Because a max-tree of an image with scalar attributes on the nodes is just a node-weighted graph, it is fairly trivial to compute a secondary max-tree of this primary max-tree. The idea is to filter this secondary max-tree, reconstituting the primary max-tree based on the filtering results, and then generating a filtered image from the filtered primary max-tree. The resulting filters were dubbed "shapings". Though interesting new results were obtained, it remains hard to envisage the precise effect of filtering a max-tree based on a secondary max-tree in this way.

None of the above approaches are suitable for so-called vector-attribute filtering [17], in which each connected component has a feature vector instead of a single scalar value. In this paper we will extend the idea of Xu et al. [16] to the vector-attribute case. Rather than building a max-tree of a max-tree, which requires a total order on the attributes, I construct an α-tree [10] of the max-tree. For α-trees, which derive from partition hierarchies described in [18], no total order is needed, which is why they are suitable for graphs with vectorial weights. The aim is to create a hierarchy of simplifications of the input component tree, each of which contains only the nodes at which large transitions in attribute vectors occur. These ideally contain the most essential information in the tree.

In the rest of the paper, I will focus on max-trees, although the method described will work on other tree structures as well. I will first discuss attribute filters and max-trees, and hierarchical clustering using α-trees, and the principles behind the method. I will then present a fast algorithm to generate simplified max-trees at any level of the α-trees, by cutting at a particular dissimilarity threshold α, and where necessary correct the attribute values in the case of attributes that depend on grey-scale content. An algorithm for pattern spectra based on α-cuts is also presented, along with a discussion of the computational complexity. A simple experiment showing the effect of applying α-cuts to pattern spectra is presented, followed by a discussion future and plans for future work are given in the final section.

2 Attribute filters and component trees

Breen and Jones [12] introduced attribute filters, which are attribute thinnings in the non-increasing, anti-extensive case we will focus on here. In the binary case they remove connected foreground components that do not meet some non-increasing criterion T.

Definition 1 *The binary attribute thinning Φ^T of set X with criterion T is given by*

$$\Phi^T(X) = \{x \in X \mid T(\Gamma_x(X))\} \tag{1}$$

where Φ_T is the trivial thinning with non-increasing criterion T, and Γ_x is the connectivity opening at point x. The latter returns the connected component to which x belongs if $x \in X$ and $\emptyset$ otherwise. As can be seen, only those points x that are members of a connected component that meets criterion T are retained.

Vector-attribute filtering [17] was introduced as an extension to attribute filtering. Rather than computing a single attribute, a vector of attributes is computed for each node. As it would be impractical to set thresholds for each vector, Urbach et al. [17] proposed using thresholds to distances to some collection of prototypes to detect objects in images. Later Naegel et al. [19] used Mahalanobis distances to a set of prototypes for segmentation of dermatological images. Formally, vector-attribute thinnings can be defined as follow:

Definition 2 *The vector-attribute thinning $\Phi^{\vec{\tau}}_{\vec{r},\epsilon}$ of X with respect to a reference vector $\vec{r}$ and using vector-attribute $\vec{\tau}$ and scalar value ϵ is given by*

$$\Phi^{\vec{\tau}}_{\vec{r},\epsilon}(X) = \{x \in X \mid T^{\vec{\tau}}_{\vec{r},\epsilon}(\Gamma_x(X))\}. \tag{2}$$

The criterion $T^{\vec{\tau}}_{\vec{r},\epsilon}$ is defined as

$$T^{\vec{\tau}}_{\vec{r},\epsilon}(C) = \rho(\vec{\tau}(C), \vec{r}) > \epsilon. \tag{3}$$

with ρ some metric or dissimilarity function.

The above definitions can be generalised to grey scale by the usual threshold superposition method [21], and implemented using max-trees in the anti-extensive case [7]. Max-tree nodes represent the connected foreground components of threshold sets at all threshold levels in the image. The connected components of the threshold levels are referred to as *peak components*. A simple example

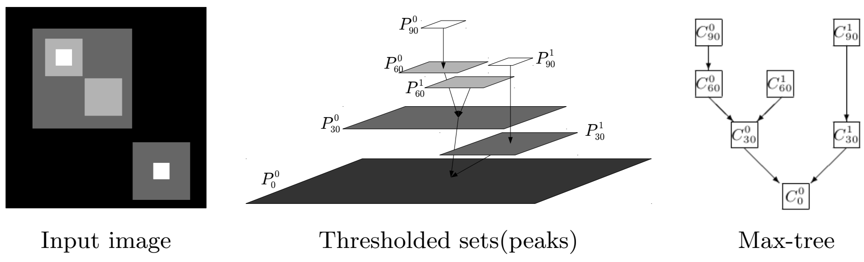

Fig. 1. A simple grey-scale image, the foreground components of each threshold set, also know as *peak components*, and the resulting component tree, which is referred to as a *max-tree* in this case. Figure from [20].

is shown in Figure 1. Each node may be assigned one or more attributes, and filtering the image applies the attribute criterion to all of the nodes, and reconstituting the image based on which nodes are preserved and which removed. In the case of an increasing criterion T, i.e. if $C \subseteq D$ then $T(C) \leq T(D)$, filtering corresponds to pruning the tree. In the non-increasing case, more complex strategies are used [7, 13, 16].

3 Hierarchical clustering of max-trees

Previously, the use of cluster analysis on max-tree nodes has been explored in [22]. In that case, nodes were clustered purely based on the attribute vectors, completely disregarding the tree structure. Here I propose to use hierarchical clustering of nodes taking the tree structure into account explicitly, through the use of α-trees [10] of max-trees.

Ouzounis and Soille [10] introduced term α-tree as a way of representing hierarchies of α-connected components (α-CCs) of images, suitable for vector images such as colour and hyperspectral images. The α-CCs of an edge-weighted graph are connected subgraphs of maximal extent such that there exists a path within the α-CCs between each pair elements, such that the edge weights in the path are all smaller than or equal to some threshold α [18]. An α-tree can be created on any graph with vector weights on the vertices by assigning weights to the edges between any two vertices, based on some dissimilarity measure. This allows α-trees to be built on any image, whereas max-trees are restricted to those cases where a total order can be imposed upon the pixel values. It has been shown that α-trees are equivalent to min-trees of an edge-weighted graph [23], and the computational complexity of building one is therefore equivalent to that of building a max-tree.

If we have a max-tree of an image, with vector attributes on the nodes, we cannot readily compute a max-tree of this max-tree to compute a shaping [16]. However, we can obviously compute an α-tree, using any of the existing algorithms. This does not make use of the fact that the max-tree is a tree, not a general graph, which means that there is just a single shortest path connecting any two nodes, and any longer path (taking detours to the root) must traverse the edges in the shortest path. This in turn means that the dissimilarity δ of the edge linking a node to its parent forms a boundary between two α-CCs for any $\alpha < \delta$. In the following I will discuss the special case of α-trees, and in particular α-cuts of max-trees.

Let us assign a weight δ on the edge between current node and its parent. I will refer to these edges and weights as parent edges and parent weights respectively. The weights can be computed using some dissimilarity measure $\rho : \mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R}$, assuming n-dimensional attribute vectors on each of the nodes.

Definition 3 *An α-connected component of any tree with weights δ on the edges is a subtree of maximal extent containing no edges with $\delta > \alpha$.*

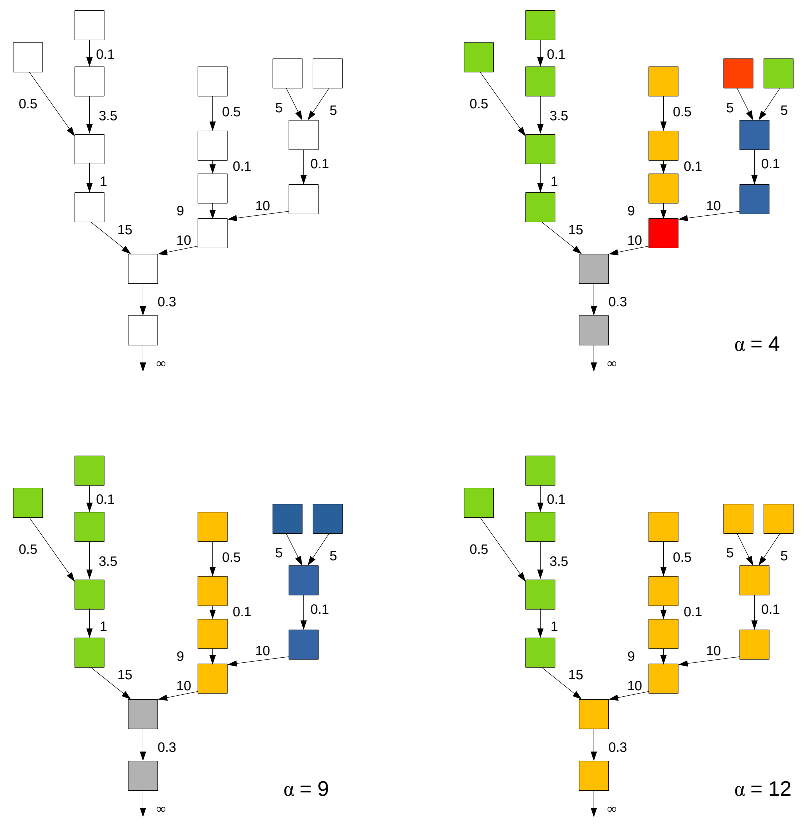

Fig. 2. Hierarchical clustering of a max-tree (top left) using increasing α-cuts. At $\alpha = 15$ the tree would become a single connected component

It can readily be seen that the root element of any α-connected component must have a parent weight $\delta > \alpha$ on its parent-edge. Furthermore, these root nodes uniquely identify the α-connected components of the tree. Indeed, to determine which α-connected component any node with parent weight $\delta \leq \alpha$ belongs to we simply need to find its α-parent.

Definition 4 *The α-parent of a node with parent weight $\delta \leq \alpha$ is the nearest ancestor with parent weight $\delta > \alpha$.*

Thus, the nodes with parent weights $\delta > \alpha$ can be seen as the canonical elements of the nodes at level α in the α-tree of the max-tree. An example of a series of α-cuts of a simple max-tree is shown in Figure 2. The assignment of α-parents and reduction to a simplified max-tree is shown in Figure 3.

4 Algorithm

We assume that we have a max-tree built using any one of many available algorithms [24]. Without loss of generality, we can assume the nodes are stored in an array `node`. For the moment we assume each node has a vector of attributes `attr` which do not depend on grey level, like area or various moment invariants used previously [13, 25]. Each node also contains an index `par` to its parent, and a field `delta` containing the dissimilarity measure between its attribute vector and that of its parent in the max-tree. Finally, we add a field `alpha_par` to each node, which is initialized to be equal to that of the `par` field. To simplify the algorithm, the root node is initialized to have a maximal value of its `delta` field and both its `par` and `alpha_par` point to root node itself.

The algorithm to compute an α-cut of a max-tree now boils down to the following steps:

1. Create an index array `Index` of max-tree nodes, sorted in increasing order of their `delta` field.
2. For all nodes in the max-tree set the `alpha_par` field to its α-parent.

The first step ensures we can easily select all the roots of the α-CCs for a given value of α, simply by using e.g. binary search in the index array to find the first node in which `node[index[i]].delta` $> \alpha$.

The latter step ensures the α-parents of the tree are properly set. This can be achieved in linear time by calling function `find_alpha_par` shown in Algorithm 1 for each node of the tree. Each call to this function follows the root path until the α-parent has been found, and sets all the `alpha_par` field along the root path. This means that subsequent calls that explore the same root path will essentially yield a shortcut to the correct α-parent. Note that before the initial construction of the α-cut, each `alpha_par` was set to the value of the `par` field, which is the correct value for $\alpha = 0$. Therefore, the algorithm follows the usual root paths in the max-tree initially, and should process at most all the nodes in the max-tree once. Once a particular α-cut has been computed, and we wish to compute a new cut with $\alpha' > \alpha$, we need only call `find_alpha_par` for the nodes with `node[index[i]].delta` $> \alpha'$, and these calls would only traverse those nodes with `node[index[i]].delta` $> \alpha$.

The result of calling the above algorithm for $\alpha = 4$ on the max-tree from Figure 2 is shown in Figure 3. It also shows how limiting the tree to only the nodes with `node[index[i]].delta` $> \alpha$ yields a simplified version of the max-tree.

4.1 Attributes of α-CCs of max-trees

Until now, we have only considered the structure of the simplified trees, but not the attributes. If we consider all "flat" attributes, i.e. those that only depend of the shape of the peak component, but not the grey-levels within it, nothing needs to be done, as each node contains all the information pertaining to that

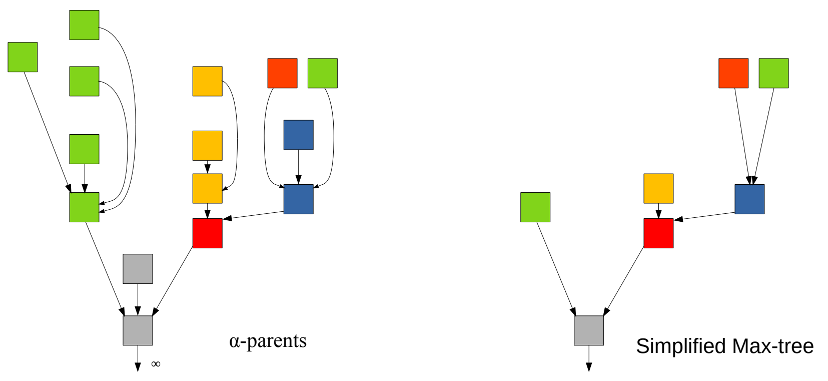

Fig. 3. The result of the `find_alpha_par` function for $\alpha = 4$: (left) `alpha_par` pointers of max-tree, (right) Simplified max-tree.

shape. In the case of non-flat attributes, i.e. those that depend on the grey-level content of the node, not just its shape, some extra post-processing needs to be done, if we want to represent the properties of each partition of the max-tree from the computed attributes. Here I will restrict myself to the attributes based on the first and second moments of the grey level distribution, although the method can be extended to higher orders, and in principle also to certain other attributes.

We assume that each node stores the sum of grey levels and sum of squared grey levels of all the pixels within the peak component in fields `SumGrey` and `SumGreySquare` respectively. A field `Gval` stores the grey value of each node.

The process traverses the tree in increasing grey level order. Whenever it finds a node such that `node[i].delta` $> \alpha$ it calls function `correct_alpha_par` on that node. This looks up the α-parent, and subtracts the sum of (squared) grey levels of the current node from the sum of (squared) grey levels of the α-parent.

Algorithm 1 The `find_alpha_par` function. Note that `MTnode` is a max-tree node struct type, and `*node` is the array representing the max-tree.

```
index find_alpha_par(float alpha, MTnode *node, index current){
  index alpha_par = node[current].alpha_par;

  if (node[alpha_par].delta <= alpha)
    node[current].alpha_par = find_alpha_par( alpha, node, alpha_par );

  return node[current].alpha_par;
}
```

It then adds the product of the area of the current node and the (squared) grey level of the α-parent to the sum of the (squared) grey levels of the α-parent. This effectively clips off the contribution of the current node to these sums in its α-parent at the level of the latter's grey level. The algorithm is shown in Algorithm 2.

Algorithm 2 The `correct_alpha_par` function

Precondition: `node[current].delta` $>$ `alpha`

```
void correct_alpha_par ( MTnode *node, index current ){
  index alpha_par = node[current].alpha_par;

  node[alpha_par].SumGrey =
     node[alpha_par].SumGrey  - node[current].SumGrey
     + node[current].area]*node[alpha_par].Gval;
  node[alpha_par].SumGreySquare =
     node[alpha_par].SumGreySquare - node[current].SumGreySquare
     + node[current].area * node[alpha_par].Gval * node[alpha_par].Gval;

}
```

After application of this algorithm, attributes like power [26] and volume (or flux) [27] can be computed in the usual way from these corrected sums. Once the attributes have been computed, we can in principle filter the original max-tree based on only the simplified α-cut version of the tree, by only applying a criterion T to the nodes with $\delta > \alpha$, and for all other nodes copy the decision made for their α-parent. Likewise, granulometries based on α-cuts might also reveal more structure, as single objects are not smeared out over a range of attributes.

4.2 Pattern Spectra

Computation of pattern spectra using alpha-cuts of max-trees can be done with a very minor adaptation of the original code from [13], as shown in Algorithm 3. After α-parents have been assigned, and the array containing the pattern spectrum has been set to zero, we compute each node's contribution to the total sum of grey levels in the image, add it to the appropriate bin of the pattern spectrum. The only difference with the original algorithm is the if-statement in Algorithm 3. If $\delta > \alpha$ the node is also a node in the simplified tree, and we add the flux to its bin in the spectrum in the usual way. If not, we add it to its α-parent.

Algorithm 3 The `alpha_cut_pattern_spectrum` function

Precondition: `find_alpha_par` has been applied to all nodes in the max-tree array `node`. `BinFunc` computes the bin in the spectrum to which a node should be assigned.

```
void alpha_cut_pattern_spectrum ( MTnode *node,
                                  greyval *Spectrum
                                  float   alpha){

  Set all elements of Spectrum to zero

  for all node[i] except root {
     par = node[i].parent;
     flux = (node[i].Gval - node[par].Gval) * node[i].area;

     if (node[i].delta <= alpha)
       bin = BinFunc(node, node[i].alpha_par );
     else
       bin = BinFunc(node, i);

     Spectrum[bin] = Spectrum[bin] + flux;
  }
}
```

4.3 Computational Complexity

Assuming the computation of the dissimilarities δ between node and parent is independent of the number of node N, the initial step of computing the dissimilarities is $O(N)$, as each node need only inspect its own parent, and compute a single value. Sorting the edges by δ is simply $O(N \log N)$. Note, however that this sorting is not necessary in all cases. It is useful if we want to choose α as a percentile of the distribution of δ values. The `find_alpha_par` function of Algorithm 1 is essentially the same as the restitution stage of regular max-tree filtering, which is also linear in N [7]. Indeed, the entire process (without sorting) is essentially the same as that of the entire filtering phase of a max-tree, which in practise is between 1 and 5 % of the total compute time. The building phase is the costly phase. By contrast, if we explicitly built an α-tree of the max-tree this is equivalent to building a min-tree of N items [23] which is evidently more costly, both computationally and in terms of memory use.

The complexity of `correct_alpha_par` in Algorithm 2 is independent of N, so applying it in grey-level order to the entire tree is $O(N \log N)$ if the nodes are not sorted in grey level order by the max-tree building algorithm, and $O(N)$ if they are. This too is quite similar to the compute load of filtering an existing max-tree. Likewise, the complexity of Algorithm 3 is $O(N)$, assuming `BinFunc` is independent of N (which is usually the case).

5 Some initial results

Algorithm 1 and 3 were implemented in C, within the code base for 2D pattern spectra of [13], and applied to a 621 × 501 image of a diatom from the ADIAC data set [28]. The δ value was computed as

$$\delta(C) = \frac{\text{area}(\text{parent}(C)) - \text{area}(C)}{\text{area}(\text{parent}(C))} \tag{4}$$

Timings revealed small differences in timings between computation of pattern spectra with and without α-cuts, using a desktop PC with an Intel® Core™ i7-6700 CPU at 3.40GHz. Running 100 iterations of the code with and without α-cuts resulted in a difference of 58 ms on average (or 0.58 ms for a single iteration), out of a total compute time of 929 ms. Thus, roughly 6% of compute time is spent on the computation of δ values, α-parents, and the modifications needed for the α-cut pattern spectra. There seems to be a slight decreasing trend as a function of α, with 64 ms required at α=0, and 52 ms at α=0.8. The average compute time of 0.58 ms is much smaller than building an α-tree of the same image (not its max-tree), which took around 110 ms. Given that the image has 311,121 pixels, vs the max-tree having 70,296 nodes, we can roughly estimate the required time for computing an α-tree of the max-tree as around 24 ms, or some 40× slower than computing a single α-cut.

Figure 4 shows the resulting spectra for α=0, 0.2, and 0.4. These show a pattern reminiscent of a skewed butterfly, in particular at α=0. The left-hand "wing" mainly represents the structures within the diatom, whereas the right-hand side, with larger areas, mainly shows the structure in the background. By increasing α, the flux in these background structures almost all become focused in the top right corner of the spectrum. Changes on the left are subtler, suggesting the detail in the diatom cell is preserved. Much more extensive tests are needed to draw any further conclusions.

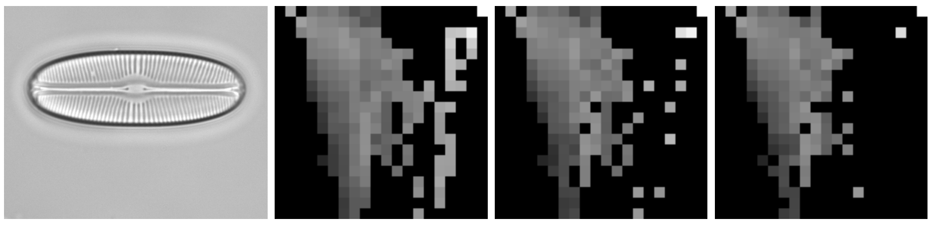
Fig. 4. Pattern spectra of a diatom image for α-cuts at 0, 0.2, and 0.4

6 Conclusions

I have presented a simple algorithm to compute α-cuts of component trees, which are horizontal cuts through α-trees of component trees. These provide an easily tunable simplification of the component trees. Apart from selecting a value of α and searching for the appropriate location in the index array, we could simply just use the top 10% of the nodes in term of their δ value. A simple slider in an interactive tool would readily allow finding the appropriate level of simplification. Tree simplification could also simplify the analysis and visualization of these trees. Besides, it is hoped that this will allow better selection of meaningful nodes and their attribute vectors for training of machine learning methods. This in turn could lead to better integration of machine learning methods with morphological connected filtering.

Obviously, α-trees and level-line trees can be simplified in the same way, as the presented algorithm for α-cuts carries over without modification to these tree structure, although some modifications would be needed for the grey-scale attribute correction. In principle binary partition trees could be processed this way, if you allow for the fact that the resulting simplified tree might no longer be binary. It should even be possible to extend the technique to the distributed component graph used for distributed computing of attribute filters [20, 29], as each local modified component tree contains all the data necessary to compute any filtering or analysis step.

In the near future we will apply this method to detection of important structures in CT, MRI and PET scans, and to detection and analysis of astronomical objects. Given the speed of the simplification method, we aim at building interactive tools for adaptation of the α values to the task at hand.

References

1. Salembier, P., Wilkinson, M.H.F.: Connected operators: A review of region-based morphological image processing techniques. IEEE Signal Processing Magazine **26**(6) (2009) 136–157
2. Salembier, P., Serra, J.: Flat zones filtering, connected operators, and filters by reconstruction. IEEE Trans. Image Proc. **4** (1995) 1153–1160
3. Heijmans, H.J.A.M.: Connected morphological operators for binary images. Comp. Vis. Image Understand. **73** (1999) 99–120
4. Klein, J.C.: Conception et réalisation d'une unité logique pour l'analyse quantitative d'images. PhD thesis, Nancy University, France (1976)
5. Cheng, F., Venetsanopoulos, A.N.: An adaptive morphological filter for image processing. IEEE Trans. Image Proc. **1** (1992) 533–539
6. Vincent, L.: Morphological area openings and closings for grey-scale images. In O, Y.L., Toet, A., Foster, D., Heijmans, H.J.A.M., Meer, P., eds.: Shape in Picture: Mathematical Description of Shape in Grey-level Images. NATO (1993) 197–208
7. Salembier, P., Oliveras, A., Garrido, L.: Anti-extensive connected operators for image and sequence processing. IEEE Trans. Image Proc. **7** (1998) 555–570
8. Salembier, P., Garrido, L.: Binary partition tree as an efficient representation for image processing, segmentation and information retrieval. IEEE Trans. Image Proc. **9**(4) (April, 2000) 561–576
9. Jones, R.: Connected filtering and segmentation using component trees. Comp. Vis. Image Understand. **75** (1999) 215–228
10. Ouzounis, G.K., Soille, P.: The Alpha-Tree algorithm. Publications Office of the European Union (Dec. 2012)

11. Bosilj, P., Kijak, E., Lefèvre, S.: Partition and inclusion hierarchies of images: A comprehensive survey. Journal of Imaging **4**(2) (2018) 33
12. Breen, E.J., Jones, R.: Attribute openings, thinnings and granulometries. Comp. Vis. Image Understand. **64**(3) (1996) 377–389
13. Urbach, E.R., Roerdink, J.B.T.M., Wilkinson, M.H.F.: Connected shape-size pattern spectra for rotation and scale-invariant classification of gray-scale images. IEEE Trans. Pattern Anal. Mach. Intell. **29** (2007) 272–285
14. Wilkinson, M.H.F., Pesaresi, M., Ouzounis, G.K.: An efficient parallel algorithm for multi-scale analysis of connected components in gigapixel images. ISPRS International Journal of Geo-Information **5**(3) (2 2016) 22
15. Kiwanuka, F.N., Wilkinson, M.H.F.: Automatic attribute threshold selection for blood vessel enhancement. In: Proc. 20th Int. Conf. Pattern Rec. (2010) 2314–2317
16. Xu, Y., Géraud, T., Najman, L.: Connected filtering on tree-based shape-spaces. IEEE Trans. Pattern Anal. Mach. Intell. **38**(6) (2016) 1126–1140
17. Urbach, E.R., Boersma, N.J., Wilkinson, M.H.F.: Vector-attribute filters. In: Mathematical Morphology: 40 Years On, Proc. Int. Symp. Math. Morphology (ISMM) 2005, Paris (18-20 April 2005) 95–104
18. Soille, P.: Constrained connectivity and connected filters. IEEE Trans. Pattern Anal. Mach. Intell. **30**(7) (July 2008) 1132–1145
19. Naegel, B., Passat, N., Boch, N., Kocher, M.: Segmentation using vector-attribute filters: Methodology and application to dermatological imaging. In: Proc. Int. Symp. Math. Morphology (ISMM) 2007. (2007) 239–250
20. Gazagnes, S., Wilkinson, M.H.F.: Distributed component forests in 2-d: Hierarchical image representations suitable for tera-scale images. International Journal of Pattern Recognition and Artificial Intelligence **33**(11) (2019) 1940012
21. Maragos, P., Ziff, R.D.: Threshold decomposition in morphological image analysis. IEEE Trans. Pattern Anal. Mach. Intell. **12**(5) (1990) 498–504
22. Kiwanuka, F.N., Wilkinson, M.H.F.: Cluster based vector attribute filtering. In: Mathematical Morphology and Its Applications to Signal and Image Processing. Lecture Notes in Computer Science, Springer (2015) 277–288
23. Soille, P., Najman, L.: On morphological hierarchical representations for image processing and spatial data clustering. In: International Workshop on Applications of Discrete Geometry and Mathematical Morphology, Springer (2010) 43–67
24. Carlinet, E., Géraud, T.: A comparative review of component tree computation algorithms. IEEE Trans. Image Proc. **23**(9) (2014) 3885–3895
25. Westenberg, M.A., Roerdink, J.B.T.M., Wilkinson, M.H.F.: Volumetric attribute filtering and interactive visualization using the max-tree representation. IEEE Trans. Image Proc. **16** (2007) 2943–2952
26. Young, N., Evans, A.N.: Psychovisually tuned attribute operators for preprocessing digital video. IEE Proceedings-Vision, Image and Signal Processing **150**(5) (2003) 277–286
27. Tushabe, F.B.: Extending Attribute Filters to Color Processing and Multi-Media Applications. PhD thesis, University of Groningen (2010)
28. du Buf, H., Bayer, M., Droop, S., Head, R., Juggins, S., Fischer, S., H. Bunke, M.W., Roerdink, J., Pech-Pacheco, J., Christobal, G., Shahbazkia, H., Ciobanu, A.: Diatom identification: A double challenge called ADIAC. In: Proceedings ICIAP, Venice (September 1999) 734–739
29. Gazagnes, S., Wilkinson, M.H.F.: Distributed connected component filtering and analysis in 2d and 3d tera-scale data sets. IEEE Trans. Image Proc. **30** (2021) 3664–3675