An Efficient Algorithm for Connected Attribute Thinnings and Thickenings

David Lesage[1,4], Jérôme Darbon[2,5], and Ceyhun Burak Akgül[1,3]

[1] École Nationale Supérieure des Télécommunications (ENST)
46 rue Barrault, F-75013 Paris, France

[2] EPITA Research and Development Laboratory (LRDE),
14-16 rue Voltaire F-94276 Le Kremlin-Bicêtre, France

[3] Electrical and Electronics Engineering Department, Bogazici University
34342 Bebek, Istanbul, Turkey

[4] in collaboration with Siemens Corporate Research
755 College Road East, Princeton NJ 08540, USA

[5] UCLA Mathematics Department, Box 901555, Los Angeles, USA

E-mails: {lesage,akgul}@enst.fr, jerome@math.ucla.edu

Abstract. Connected attribute filters are morphological operators widely used for their ability of simplifying the image without moving its contours. In this paper, we present a fast, versatile and easy-to-implement algorithm for grayscale connected attribute thinnings and thickennings, a subclass of connected filters for the wide range of non-increasing attributes. We show that our algorithm consumes less memory and is computationally more efficient than other available methods on natural images, for strictly identical results.

1 Introduction

Connected attribute filters are morphological operators enjoying the property of simplifying an image while preserving the contour information, as they work on the connected components of the image level-sets [9, 15, 20]. Such filters are widely used in practice for image classification [1], filtering [18], segmentation [5, 21], feature analysis [23] and shape filtering [26].

Connected attribute filters are first classified according to their theoretical properties. Filters considering image upper level-sets, such as attribute openings and thinnings, are anti-extensive [18]. Intuitively, they are well suited for applications where objects are brighter than their background, as higher level components are seen as included in lower level ones. The dual vision takes lower level-sets into account and leads to extensive filters such as attribute closings and thickenings. A second distinction among connected filters is made on

the type of attribute used. Attribute openings and closings correspond to the case where the attribute associated with a level set is a non-decreasing function w.r.t. set inclusion. A popular example can be found in area opening and closing [24, 20]. A larger class consists of filters using non-increasing criteria, namely attribute thinnings and thickenings. Numerous practically interesting criteria fall in that category, such as elongation criterion from [26], complexity/simplicity, motion estimation and entropy factors from [18].

Efficient algorithms have been described in [6, 14, 18, 24] for connected attribute openings and closings. To our knowledge, few algorithms have been proposed for attribute thinnings and thickenings [3, 18]. In this paper, we present a new algorithm for such filters. It requires *less* memory and exhibits *better* runtime performance compared to other available methods, while yielding strictly identical results.

Defining connected attribute filters for grayscale images with non-increasing criteria is not a straightforward task. Indeed, the threshold decomposition property [8, 12] only holds for increasing criteria, i.e. for attribute openings and closings, allowing for a direction definition using lower- or upper-sets of the image. For thinnings and thickenings, various definitions become suitable. Salembier's *Min/Max-Trees* offer convenient representations to express them [18].

A Max-Tree is an inclusion tree that considers connected components of upper-level sets of an image. A node of the Max-Tree corresponds to a connected component C_0 of an upper-level set. Its parent is defined as the connected component C_1 of the higher upper-level set such that $C_0 \neq C_1$ and $C_0 \subset C_1$. The leaves of a Max-Tree correspond to the regional maxima of the image. An illustration is given in Figure 1-a). The dual representation by a Min-Tree considers lower-level sets instead of upper-level sets. As reported in [18], connected attribute filters for grayscale images can be defined as a *pruning* of its Min- or Max-Tree, in such a way that the threshold decomposition property [8, 12] holds. This enables the straightforward reconstruction of the filtered image from the pruned tree. Given a non-increasing criterion, many different pruning rules can be considered, each leading to a different connected-set filter [18, 26].

Following strictly this definition, a possible approach to implement a connected attribute filter consists in first building the Min/Max-Tree and then applying a pruning process. The original Min/Max-Tree construction is presented by Salembier *et al.* in [18]. It relies on a recursive flooding procedure using a hierarchical queue data structure. An alternative scheme for Min/Max-Tree construction based on Tarjan's union-find algorithm has been originally proposed by Hesselink in [11] and by Najman *et al.* in [16].

Direct implementations which by-pass tree construction are also possible. The latter require to choose the pruning rule before performing the filtering process. A popular choice for the pruning, known as the *Max rule* [18], consists in removing a node if its associated criteria do not satisfy a prescribed threshold and if all its descendants were removed. In [3], Breen *et al.* propose a queue-based algorithm to perform direct filtering using the *Max rule*. In [25], Wilkinson *et al.* present an union-find based algorithm for performing connected at-

tribute openings and closings. It should be noted that the same algorithm also performs a *Max rule*-pruning for non-increasing criteria.

In this paper, we follow the approach of Darbon and Akgül [6] for connected attribute closings and openings and extend it to non-increasing criteria. The main contributions of this paper are the following. We propose a fast and memory wise algorithm to perform the filtering for the main pruning rules proposed in the literature. We compare our algorithm with other available methods and show that our algorithm is *more efficient* in terms of both memory requirements and computational load. In Section 2, we present our algorithm along with its variations to cope with different pruning rules. Section 3 is devoted to the presentation of numerical experiments and comparisons. Finally in Section 4, we draw some conclusions and sketch future prospects.

2 Algorithm

The proposed algorithm relies on a tree pruning scheme, as in Salembier *et al.*'s original work for the Max-Tree construction [18]. A *flooding* process builds the inclusion tree of connected components and a successive *resolution* stage applies a pruning strategy to recover filtered pixel values. Our work also benefits from optimizations introduced in [6], by avoiding the external hierarchical queue and the sorting procedure used in [18]. We emphasize our algorithm is straightforward to implement given the pseudo-code of this Section and is versatile enough to be suitable for numerous filtering tasks without modification.

In the following, we limit ourselves to attributes that can be computed recursively during the flooding pass, by adding new points or by merging components. For the sake of simplicity, our algorithm is given for anti-extensive, attribute thinning filters. The dual representation for attribute thickenings only requires few tests to be inverted in the flooding procedure. We denote by N the size of the image, L the number of gray levels and A the number of auxiliary values needed to compute the attribute. We first describe the flooding process which builds the tree and then the resolution pass which finalizes the filtering.

2.1 Flooding

Our flooding step is directly inspired by Salembier's Max-Tree one and is generic for any resolution strategy. Following ideas introduced in [6], we use an array `status` to store pointers to points encoded as positive integers (lexicographic rank). In our algorithm, `status` serves two purposes. It first stores the hierarchical queue and eventually turns into the Max-Tree itself, encoding child-to-parent links. It is initialized to a special value `NONE` which signifies that no link was set and that therefore points have not been processed.

We use three arrays of length L, namely `last`, `representative` and `attribute`. The value `last[h]` stores the last point introduced in the `h`-level queue. The value `representative[h]` stores the first encountered point for the current `h`-level component C_h^k, which serves as an unique representative

of C_h^k. The value `attribute[h]` holds the A values needed to compute C_h^k attribute.

Since we seek after a generic thinning procedure, we build the tree explicitly and keep its structure intact until resolution. We cannot afford altering it as done in [6]. To keep time and memory requirements as low as possible, we instead use a boolean array `tag` which points out components not satisfying the thinning criterion λ.

Our flooding process is detailed in Algorithm 1. At level h, the procedure `flood(h)` iterates the queue at the same level, using `last[h]` (line 2), to retrieve current component C_h^k. Point `p` is dequeued (lines 3-4) and is set to point to its representative `r` (line 6). Non-representative points are tagged so they always take their representative's value, without loss of generality w.r.t. the resolution rule. Then non-processed neighbors `q` of `p` are visited (line 7). They are pushed in the queue (lines 10-11) and registered as representative if we reached a new level component (line 14). If `q`'s gray-level `i` is higher than `h`, the flooding procedure is relaunched recursively to extract the subtree (lines 17-19). Back from the neighbors, the attribute of the current `h`-level component is updated (line 21).

Once the propagation at level `h` is over, C_h^k is completely retrieved and its final attribute value can be tested (line 21). If it does not fulfill λ, C_h^k representative is tagged. Parent relationships are then set in a similar way to [18] and [6] (lines 23-29). Auxiliary arrays at level `h` are finally reset (lines 31-32).

The flooding is initially launched at the lowest level in the image. Eventually, the entire tree structure is encoded in `status`. By construction, non-representative points are directly linked to their representative and representatives point to a lower level representative until root is reached. Additionally, we can directly identify a component that does not fulfill the attribute criterion by checking its `tag`.

2.2 Resolution

Once the tree is built and tagged, a resolution phase eventually retrieves filtered values and stores them in the output image `out`. Depending on the type of non-increasing attribute [18] and the application, several resolution schemes have been proposed in the literature. We implemented classical rules such as *Direct*, *Min* and *Max* rules [10, 19, 20] as well as the *Subtractive* rule from [22]. Due to space restrictions, we only present Min and Max rules. Others can be easily deduced from the mechanisms presented here.

In contrast to Salembier's original Max-Tree approach [18], we do not dispose of a sorted structure allowing tree traversals from parents to children. Our algorithm remains as efficient as possible by altering the tree to avoid redundant operations.

Min Rule A node is removed if it does not satisfy λ or if an ancestor was removed. A loop on all image points launches the recursive procedure of Algorithm 2. The tree is progressively encoded with the final filtered values as neg-

ative integers (`-val-1`) similarly to [6]. A point is set to its parent's value if it is tagged or if its parent's value changed. An illustration of this strategy is given in Figure 1-b).

Max Rule A node is removed if it does not satisfy λ and if all of its children are removed. This rule requires two passes since the tree is encoded with only parent relationships. The first loop launches Algorithm 3 procedure on all image points. It uses a boolean `child_kept` to propagate the information that a child node has kept its value. If it can be resolved during the first pass, the final filtered value is encoded in `status[p]` as a negative integer. If not, the point does not fulfill λ and has all of its children removed. Thus, a second pass (Algorithm 4) simply tracks down non-negative `status` values to set them to their parent's value. An illustration of this strategy is given in Figure 1-c).

Algorithm 1 flood(level h)

```
1: r ← representative[h]
2: while last[h] ≠ NONE do // propagation at level h
3:    p ← last[h] // pop p from the queue
4:    last[h] ← status[p]
5:    if p ≠ r then // set to the representative
6:       status[p] ← r; tag[p] = true
7:    for all neighbors q of p with status[q] = NONE do
8:       i ← ima[q]
9:       status[q] = last[i]; last[i] = q // push q in the queue
10:      if representative[i] = NONE then
11:         representative[i] ← q
12:      m ← i
13:      if i > h then // flood higher level components
14:         repeat
15:            m ← flood(m)
16:         until m > h
17:   attribute[h].update(p) // update the attribute with p
18: if attribute[h].val < λ then // tag as not satisfying λ
19:    tag[r] = true
20: // set parentship
21: m ← h-1
22: while m > 0 and representative[h] = NONE do
23:    m ← m-1
24: if m > 0 then
25:    attribute[m].merge(attribute[h]) // merge attributes
26:    status[r] ← representative[m] // link representatives
27: // reset level h auxiliary values
28: attribute[h].reinit(); last[h] ← NONE;
29: representative[h] ← NONE
30: return m
```

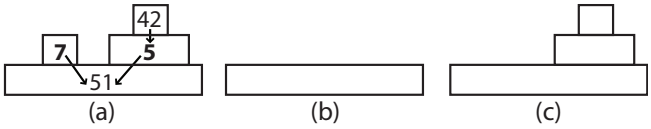

Fig. 1. Max-Tree and attribute values of a 1D signal (a), Min-rule (b) and Max-rule thinning (c) for $\lambda = 10$.

Algorithm 2 resolve_min_rec(point p)

```
if status[p] ≥ 0 then // not processed
   parent_val ← resolve_min_rec(status[p])
   if tag[p] = true or parent_val ≠ ima[status[p]] then
      status[p] ← -parent_val-1; out[p] ← parent_val
   else // p keeps its value
      status[p] ← -ima[p]-1; out[p] ← ima[p]
return out[p]
```

Algorithm 3 resolve_max_rec1(point p, boolean child_kept)

```
if status[p] ≥ 0 then // not processed
   if tag[p] = false or child_kept then // p keeps its value
      status[p] ← -ima[p]-1; out[p] ← ima[p]
      resolve_max_rec1(status[p], true) // propagate info
return
```

Algorithm 4 resolve_max_rec2(point p)

```
if status[p] ≥ 0 then // not processed, assign parent's value
   out[p] ← resolve_max_rec2(status[p])
   status[p] ← -out[p]-1
return out[p]
```

3 Analysis and Experiments

In this Section, we compare our approach to Salembier *et al.*'s Max-Tree algorithm [18] and Wilkinson *et al.*'s method based on Tarjan's union-find method for attribute openings [25]. The latter is not a thinning algorithm, but is strictly equivalent to a thinning with a Max rule for non-increasing attributes. We voluntarily omit priority queue-based algorithms [24, 3] which exhibit a high dependance w.r.t the threshold value λ [14]. Experiments show that our algorithm is faster for natural images and requires less memory than other other available algorithms. We first compare memory requirements and then present time results along with theoretical time complexity.

3.1 Memory requirements

Since $N \gg L$ in general, we neglect memory used by arrays of size L for all three algorithms. Salembier *et al.*'s Max-Tree algorithm is the most demanding one in terms of memory. It uses a queue of N integers and a tree of size N. Each tree node contains at least a pointer to the parent, the final attribute value and the filtered gray level. Min and Substractive rules also require the original gray level to be stored, totalizing $5N$ values plus a mandatory output image.

Wilkinson *et al.*'s union-find algorithm uses two arrays, `PixelSort` and `Parent`, of N integers each. The array `Parent` eventually stores the output image. Unlike both other algorithms, attributes cannot be computed in an array of size L. It requires an array `auxdata` of N pointers and a dynamic allocation/deallocation scheme to limit the memory consumption to $N + (A * N)/2$ values in the worst case. This totals to at least $3N$, at worst $3N + (A * N)/2$ [14].

Naïve implementation of our approach requires an integer array `status`, a boolean array `tag` and an output image of size N. Similarly to Meijster *et al.*'s method [14], a straightforward modification allows for encoding filtered values in `status`. The array `tag` can also be avoided by using a bit of each `status[p]` value with bit-masking operations, for an unsignificant computational cost. Memory requirements can thus easily be lowered to only N integers.

Memory load can become critical for applications such as medical imaging. Dealing with volumes of nearly $N = 512^3$ voxels, each integer array represents 512MB in memory. Our algorithm allows for processing such volumes with reasonable requirements using computers of the present era.

3.2 Computational Load

From a theoretical point of view, the computational cost of both Salembier *et al.*'s Max-Tree and our algorithm is dominated by the linear flooding. The search of parent nodes is linear w.r.t to L and the resolution is also linear w.r.t N, leading to a complexity of $\Theta(N \cdot L)$. The complexity of Wilkinson *et al.*'s union-find method is $\Theta(N \cdot \log(N))$ as shown in [14]. Salembier *et al.*'s Max-Tree and our approach are unattractive in theory because their complexity is pseudo-polynomial [2]. For their complexity to be polynomial, it should be polynomial w.r.t $\log_2(L)$ since it requires only $\lceil \log_2(L) \rceil$ bits to encode the integer L. They can be made polynomial by using a binary heap, but it seems to be practically uninteresting [14]. Even if pseudo-polynomial algorithms are theoretically worse than polynomial ones, they may be more attractive in practice. A well known example is Dial's pseudo-polynomial algorithm [7] for solving the shortest path problem. For many practical problems it outperforms other polynomial algorithms such as Dijkstra's one, as reported in [2, 7].

Algorithms were tested on a variety of natural images, using various criteria. Due to space restriction, we only present performance results obtained on four images, the well-known *Lena*, a highly textured image of a *carpet*, a large *satellite* image with few gray levels and a *cardiac Computed Tomography* (CT) medical volume. Original images are given in Figure 2. Their properties

are summed up in Table 1 where *Complexity* stands for the ratio between the number of iso-level connected components and image size. Note that the medical volume was cropped (the original counts 400 slices) in order to allow Salembier *et al.*'s Max-Tree and Wilkinson *et al.*'s union-find algorithms to run properly under our memory constraints (1GB RAM).

Tests presented in Figure 3 were performed using the non-increasing, 2D and 3D *elongation* criteria proposed in [26]. Resolutions for Salembier *et al.*'s Max-Tree and our algorithm used Max rule to be strictly equivalent to Wilkinson *et al.*'s union-find method. This way, results obtained by either of these schemes are identical, as they match the exact same theoretical definition and only differ by the mechanisms involved. Note that applying such a criterion and such a resolution rule may not be meaningful for all the images presented here. These tests are presented for the sake of fair performance comparison only. However, we emphasize they are representative of the behaviors observed for different criteria and different resolution rules. A result sample is depicted in Figure 4, where the elongation criterion is applied on the carpet image. Another result is given in Figure 5, with the same criterion and on the satellite image. Such a filtering could be used to help further extraction of road networks. Please refer to the works cited in introduction for additional illustrations of the numerous applicative possibilities offered by connected attribute filters.

All three algorithms have been implemented in C++ and compiled with full optimization. Timings were obtained by averaging 100 runs on a Pentium 4 2.4GHz (1MB cache and 1GB RAM). As depicted in Figure 3, our method outperforms other algorithms for natural images. Execution time of Salembier *et al.*'s Max-Tree and our algorithm is independent of λ. A slight dependence to λ is observed for the Wilkinson *et al.*'s union-find approach, confirming conclusions of [14].

Since it is based on a similar core flooding scheme, our algorithm exhibits a limited performance gain (5 to 30%) compared to Salembier *et al.*'s Max-Tree algorithm, mainly due to a faster tree construction (no pixel sorting, no external queue and less memory used). Our resolution step is slower than Salembier *et al.*'s Max-Tree's one because of our non-ordered tree traversal. It seems that the bigger the image, the less our advantage on the original algorithm becomes significant, as can be seen for the medical volume. However, memory consumption for Salembier *et al.*'s Max-Tree algorithm can meanwhile become prohibitive for such applications. We emphasize the effect of memory load and 'cache-friendliness' on performance. The less memory an algorithm needs and the more spatially coherent memory accesses are, the less often cache faults will occur. In this regard, our algorithm combines all these advantages.

Wilkinson *et al.*'s union-find approach, on the other hand, seems to suffer from a construction process more prone to important memory jumps. It may also be penalized by its dynamic allocation/deallocation scheme. The medical volume example suggests that the performance gap narrows when the number of gray levels increases, supposedly because of the dependance on L affecting both other algorithms [14, 17]. To draw a definitive conclusion, one should

adapt Wilkinson *et al.*'s union-find attribute opening to thinning operations. To our knowledge, this would unfortunately require additional equipment, following for instance ideas from [16]. Finally, the union-find approach holds a clear advantage in terms of parallelization possibilities [14]. This may turn it into a candidate of choice for highly-parallelized environments.

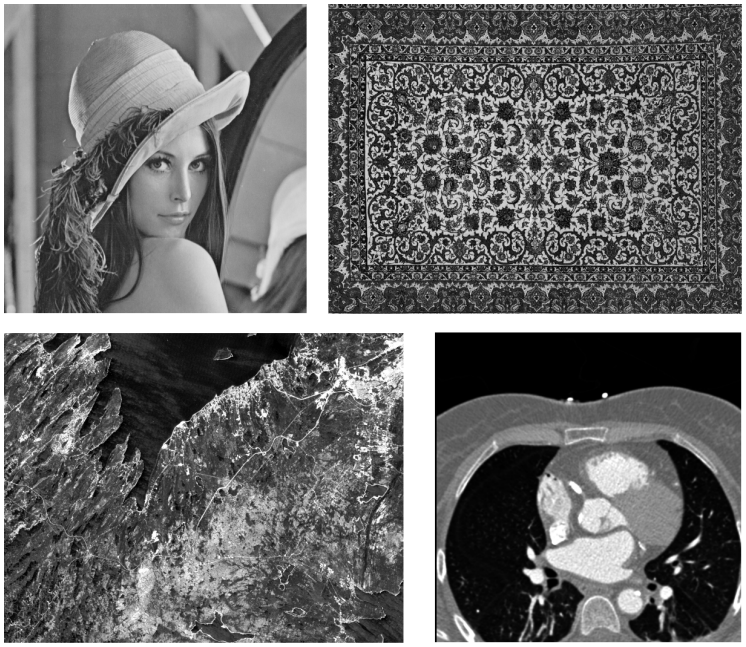

Fig. 2. Test images. From left to right, top to bottom: Lena, carpet, satellite and cardiac CT (2D slice detail).

Image	**Size**	**Levels**	**Complexity**
lena	512×512	216	0.38
carpet	1024×715	255	0.84
satellite	1380×1780	30	0.24
cardiac CT	$512 \times 512 \times 100$	5000	0.11

Table 1. Test images.

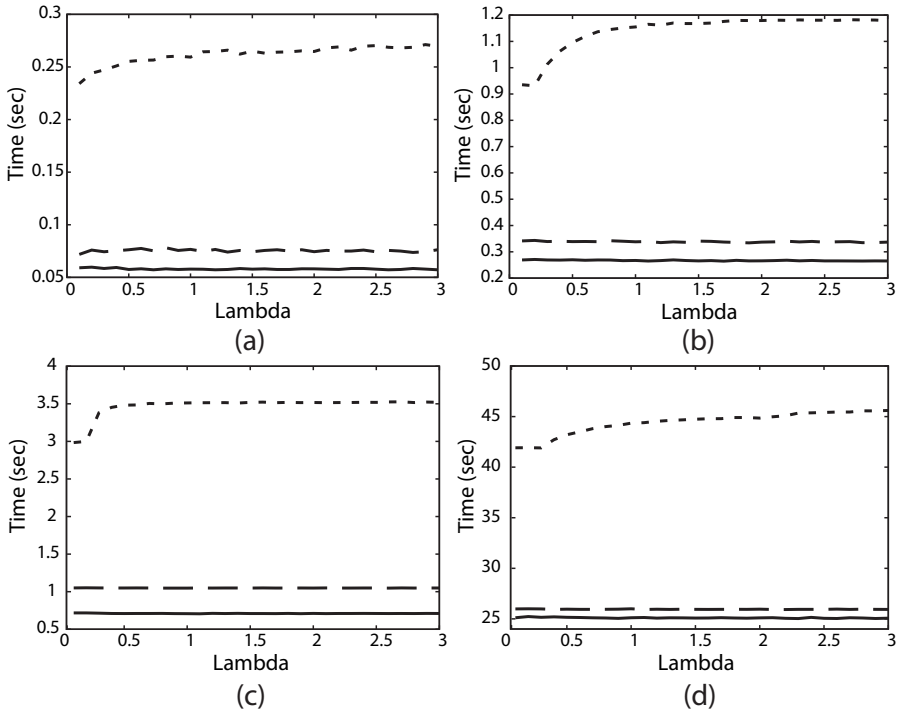

Fig. 3. Timing results for our algorithm (solid), Max-Tree (dashed) and Union-Find attribute opening (dotted) in function of λ. Elongation attribute thinning [26] on (a) lena, (b) carpet, (c) satellite and (d) cardiac CT.

4 Conclusion

In this paper, we have presented a new algorithm for attribute thinnings and thickenings. It is easy to implement and enjoys the versatility and generality of Salembier's original algorithm. Experiments on natural images show that our algorithm exhibits *better* performance than others while consuming considerably *less* memory. In a forthcoming paper, we will present new algorithms for efficient interactive filtering and visualization, as originally proposed in [13] with the Max-Tree algorithm. Extension of this approach to granulometries [14, 22] and grain filters [4] will also be described.

Acknowledgements

The authors thank Isabelle Bloch and Elsa Angelini from ENST Paris, Gareth Funka-Lea from Siemens Corporate Research and Bülent Sankur from Bogazici University for their valuable suggestions. The cardiac CT volume is courtesy of Siemens Medical Solutions.

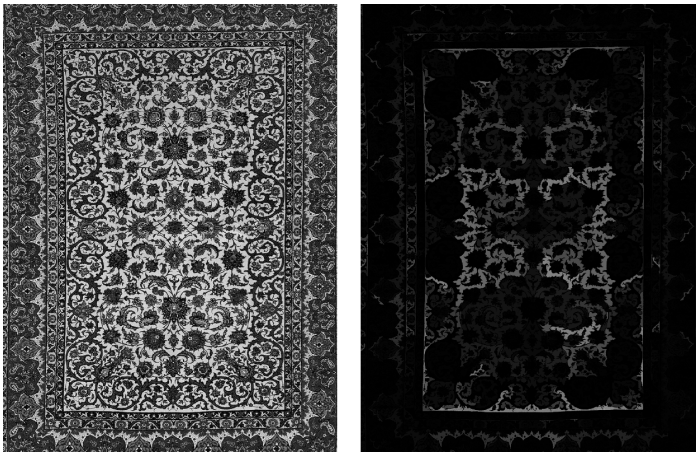

Fig. 4. Elongation attribute thinning [26] ($\lambda = 1$) on the carpet image. Left: original image. Right: filtered result.

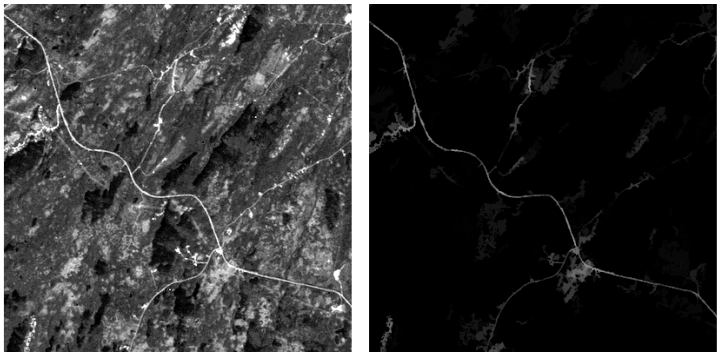

Fig. 5. Elongation attribute thinning [26] ($\lambda = 1$) on the satellite image (detail). Left: original image. Right: filtered result.

References

1. S.T. Acton and D.P. Mukherjee. Scale space classification using area morphology. *IEEE Transactions on Image Processing*, 9(4):623–635, April 2000.
2. R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network Flows*. Prentice Hall, 1993.
3. E.J. Breen and R. Jones. Attribute openings, thinnings and granulometries. *Computer Vision and Image Understanding*, 64(3):377–389, 1996.
4. V. Caselles and P. Monasse. Grain filters. *J. of Math. Imaging and Vision*, 17(3):249–270, 2002.
5. J. Crespo, R. Schafer, J. Serra, C. Gratin, and F. Meyer. The flat zone approach: A general low-level region merging segmentation method. *Signal Processing*, 62(1):37–60, 1998.

6. J. Darbon and C.B. Akgül. An efficient algorithm for attribute openings and closings. In *Proceedings of the 13th European Signal Processing Conference (EUSIPCO)*, Electronic proceedings, 2005.
7. R. Dial, F. Glover, D. Karney, and D. Klingman. A computational analysis of alternative algorithms and labeling techniques for finding shortest path trees. *Networks*, 9:215–248, 1979.
8. F. Guichard and J.M. Morel. Mathematical morphology "almost everywhere". In *Proceedings of ISMM*, pages 293–303. Csiro Publishing, April 2002.
9. H. Heijmans. Connected morphological operators for binary images. *Computer Vision and Image Understanding*, 73(1):99–120, 1999.
10. Henk J. A. M Heijmans. Theoretical aspects of gray-level morphology. *IEEE Trans. Pattern Anal. Mach. Intell.*, 13(6):568–582, 1991.
11. W.H. Hesselink. Salembier's min-tree algorithm turned into breadth first search. *Information Processing Letters*, 88(5):225–229, December 2003.
12. P. Maragos and R.D. Ziff. Threshold superposition in morphological image analysis systems. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 12(5):498–504, 1990.
13. A. Meijster, M.A. Westenberg, and M.H.F. Wilkinson. Interactive shape preserving filtering and visualization of volumetric data. In *Proc. of the Conf Comp. Signal Image Proc. (SIP)*, pages 640–643, Kauai, Hawaii, USA, August 2002.
14. A. Meijster and M.H.F. Wilkinson. A comparison of algorithms for connected set openings and closings. *IEEE Trans. on PAMI*, 24(4):484–494, April 2002.
15. F. Meyer. Levelings, image simplification filters for segmentation. *Journal of Mathematical Imaging and Vision*, 20:59–72, 2004.
16. L. Najman and M. Couprie. Quasi-linear algorithm for the component tree. In *SPIE Symposium on Electronic Imaging*, pages 18–22, 2004.
17. L. Najman and M. Couprie. Building the component tree in quasi-linear time. *Accepted to IEEE Transactions on Image Processing*, 2006.
18. P. Salembier, A. Oliveras, and L. Garrido. Antiextensive connected operators for image and sequence processing. *IEEE Trans. on Image Processing*, 7(4):555–570, 1998.
19. P. Salembier and J. Serra. Flat zones filtering, connected operators and filters by reconstruction. *IEEE Transactions on Image Processing*, 4:1153–1160, 1995.
20. J. Serra and P. Salembier. Connected operators and pyramids. In *Proc. SPIE Image Algebra Math. Morphology*, volume 2030, pages 65–76, 1993.
21. P. Soille. *Morphological Image Analysis Principles and Applications*. Springer-Verlag, 1999.
22. E. R. Urbach and M. H. F.Wilkinson. Shape-only granulometries and grey-scale shape filters. In *Proceedings of the International on Mathematical Morphology*, pages 305–314, 2002.
23. C. Vachier. Morphological scale-space analysis and feature extraction. In *Proceedings of the ICIP'01*, pages 676–679, October 2001.
24. L. Vincent. Grayscale area openings and closings, their efficient implementation and applications. In *Proc. EURASIP Mathematical Morphology and Its Application to Signal Processing*, pages 22–27, 1993.
25. M.H.F. Wilkinson and J.B.T.M. Roerdink. Fast morphological attribute operations using tarjan's union-find algorithm. In *Proceedings of the ISMM*, pages 311–320, Palo Alto, CA, June 2000.
26. M.H.F. Wilkinson and M.A. Westenberg. Shape preserving filament enhancement filtering. In *Proc. of the International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 770–777, 2001.