A Comparison of Algorithms for Connected Set Openings and Closings

Arnold Meijster and Michael H.F. Wilkinson, *Member*, *IEEE*

Abstract—The implementation of morphological connected set operators for image filtering and pattern recognition is discussed. Two earlier algorithms based on priority queues and hierarchical queues, respectively, are compared to a more recent union-find approach. Unlike the earlier algorithms which process regional extrema in the image sequentially, the union-find method allows simultaneous processing of extrema. In the context of area openings, closings, and pattern spectra, the union-find algorithm outperforms the previous methods on almost all natural and synthetic images tested. Finally, extensions to pattern spectra and the more general class of attribute operators are presented for all three algorithms, and memory usages are compared.

Index Terms—Mathematical morphology, connected set operators, attribute filters, pattern spectra, multiscale analysis, union-find.

1 Introduction

In mathematical morphology, connected set operators [10], [17] form a versatile class of image operators with a number of desirable properties, most importantly preservation of shape. The earliest members of this class were openings and closings by reconstruction, for which efficient algorithms have been developed [22]. In the binary case, an opening by reconstruction first performs an erosion with some structuring element, and then reconstructs all connected foreground components which were not completely removed by the erosion. Therefore, openings by reconstruction (and the corresponding closings) can either remove image details completely, or leave them intact, but never alter their shape.

An important development was the introduction of area openings and closings; a phrase coined by Vincent [21], though they were introduced by Cheng and Venetsanopoulos a year earlier [6] as NOP and NCP operators. Vincent's algorithm was much more efficient in the gray-scale case than the earlier method of Cheng and Venetsanopoulos. In the binary case, area openings remove all connected *foreground* components with an area smaller than some threshold λ. Binary area closings fill all *background* components with an area smaller than λ. Fig. 1 shows a comparison of applying binary openings by a 7×7 pixel square structuring element, opening by reconstruction by the same structuring element, and an area opening with $\lambda = 49$ of a binary image of bacteria, showing the differences in the action of these filters. An area opening tends to retain long thin objects more than an opening by reconstruction.

- *A. Meijster is with the Centre for High Performance Computing and Visualization, University of Groningen, PO Box 800, 9700 AV Groningen, The Netherlands. E-mail: a.meijster@rc.rug.nl.*
- *M.H.F. Wilkinson is with the Institute for Mathematics and Computing Science, University of Groningen, PO Box 800, 9700 AV Groningen, The Netherlands. E-mail: michael@cs.rug.nl.*

Manuscript received 9 Oct. 2000; revised 28 May 2001; accepted 21 Sept. 2001.
Recommended for acceptance by L. Vincent.
For information on obtaining reprints of this article, please send e-mail to: tpami@computer.org, and reference IEEECS Log Number 112956.

Area openings and closings were later extended to the wider class of attribute openings, closings, thickenings, and thinnings [5], [16]. Breen and Jones [5] extended Vincent's priority-queue algorithm for area opening to attribute openings and thinnings, whereas Salembier et al. [16] developed a new algorithm based on hierarchical queues. The latter also introduced a versatile data structure, dubbed a Max-tree (and its dual the Min-tree), which reduces image filtering to removing nodes from a tree. Attribute openings allow the use of size criteria other than width (used by openings by reconstruction) and area (used by area openings). In the binary case, attribute openings can be made by computing some increasing attribute (such as moment of inertia, diagonal of the smallest enclosing rectangle, etc.) of each connected foreground component and remove the components for which the attribute is smaller than the threshold. An attribute A is increasing if and only if for all sets C and D with $C \subseteq D$, $A(C) \leq A(D)$.

Attribute thinnings work on a similar principle, but can use shape rather than size criteria. In this case, nonincreasing attributes, such as the ratio of the square of the perimeter to the area, can be used. This type of filter allows extraction of all image details of a given shape, regardless of their sizes [19]. A 3D application of this is the extraction of filamentous details (vessels) from angiograms [25]. An example is shown in Fig. 2. Other applications of connected set morphology include filtering [6], [21] and segmentation [7].

Finally, connected set filters are important for multiscale morphology and, in particular, nonlinear scale spaces. These cannot be constructed from openings or closings by structuring elements in more than one dimension, due to problems with causality [12]. Therefore, Bangham et al. [2], [3] extended their 1D nonlinear sieves [1], [4], to an arbitrary number of dimensions using area openings and closings and their higher-dimensional counterparts, such as volume openings and closings. They show that because connected set filters never introduce new edges, they do not violate causality in any number of dimensions.

The aim of this paper is to review algorithms for connected set openings and closings and, in particular, attribute openings and closings. The two best known

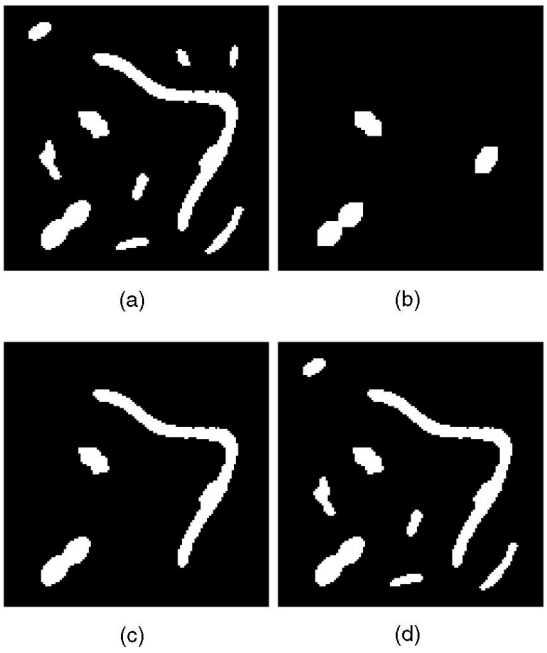

Fig. 1. Area versus structural openings: (a) A binary image of 128×128 pixels. (b) Structural opening with 7×7 structuring element. (c) Opening by reconstruction with same structuring element. (d) Area opening with $\lambda = 49$. Neither openings by reconstruction nor area opening change the shapes of connected foreground components, but the former uses width and the latter area as selection criterion.

algorithms for these image filters are the pixel priority-queue method introduced by Vincent for area operators [21] and extended by Breen and Jones to attribute operators [5], and the Max-tree method of Salembier et al. [16] which uses hierarchical queues. These two methods will be compared to an algorithm developed by the authors, which is based on the classical union-find algorithm presented by Tarjan [18]. We will focus on the area opening at first since this is one of the simplest cases. Besides, it can be extended to the more general case of attribute operators [24] in much the same way as Breen and Jones extended Vincent's algorithm [5]. To demonstrate the versatility of the union-find method, we will show how the same principles can be applied to computation of area pattern spectra. The key difference between the algorithm based on union-find and the earlier methods of Vincent [21] and Salembier et al. [16] is that we process multiple maxima simultaneously, rather than sequentially. We show how this may lead to a parallel implementation of the algorithm. Both earlier algorithms are described briefly in this paper, followed by a more detailed description of the union-find approach. Their respective performances and memory usages are compared on a range of images.

2 Theory

2.1 Connected Set Operators

The distinguishing characteristic of connected set operators is that they operate on the flat zones of images, rather than on individual pixels [17]. Let $\{\alpha_i\}$ be the partition of the domain M of image I formed by its flat zones α_i and $\{\beta_j\}$ the partition of M formed by the flat zones β_j of image $\gamma(I)$, with γ some image operator. A partition of M is any set of

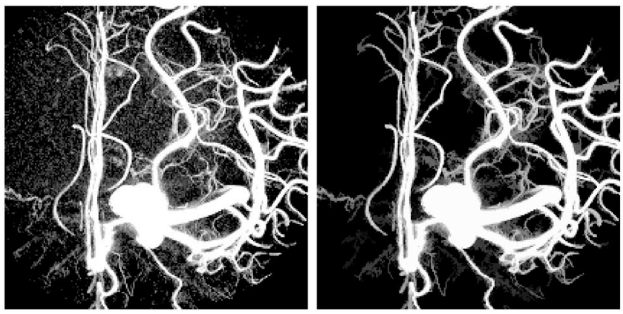

Fig. 2. An application of connected set filters (gray-scale attribute thinning) to filament extraction: (a) Maximum intensity projection of magnetic resonance angiogram 256^3 volume data set. (b) Result using an attribute thinning as shape filter. The attribute used was $I/V^{5/3}$, with I the moment of inertia and V the volume of a peak component; the attribute threshold was 2.0.

disjoint sets $\{S_i\}$ (i.e., $S_i \cap S_j = \emptyset$ for any $i \neq j$), which together form the entire set M, i.e., $\cup S_i = M$.

The operator γ is said to be a connected set operator if and only if the partition $\{\alpha_i\}$ is finer than $\{\beta_j\}$, i.e., for all sets α_i there exists a β_j such that $\alpha_i \subseteq \beta_j$. Thus, the only operations connected filters can perform is merging flat zones, and assigning new gray levels to them. Because flat zones of an image are disjoint sets and set-union of these sets is a key action taken by connected set operators, we propose to implement connected filters using Tarjan's union-find algorithm [18]. This is an efficient algorithm for maintaining *disjoint sets* under the set-union operation and has been adapted to connected component labeling and flat zone labeling [8], [9], [11]. Other applications of union-find in image processing include the computation of the watershed transform [14].

In the next sections, we will focus on how antiextensive connected set filters, in particular, attribute openings are implemented in three previously published algorithms. For the sake of simplicity, we will concentrate on area openings. These serve as a simple case to compare the efficiencies of the three approaches.

2.2 Area Openings and Closings

The theory of area operators is given only briefly here. For a more thorough discussion, the reader is referred to [21]. We will first discuss binary area openings and closings and then the extension to the gray-scale case. Binary area openings are based on binary connected openings. Let the set $X \subseteq \mathbf{M}$ denote a binary image with domain $\mathbf{M}$. The binary connected opening $\Gamma_x(X)$ of X at point $x \in \mathbf{M}$ yields the connected component of X containing x if $x \in X$ and $\emptyset$ otherwise. Thus, Γ_x extracts the connected component to which x belongs, discarding all others.

The binary area opening can now be defined as:

Definition 1. *Let $X \subseteq \mathbf{M}$ and $\lambda \geq 0$. The binary area opening of X with scale parameter λ is given by*

$$\Gamma^a_\lambda(X) = \{x \in X | A(\Gamma_x(X)) \geq \lambda\}. \quad (1)$$

The binary area closing can be defined by duality

$$\Phi^a_\lambda(X) = [\Gamma^a_\lambda(X^c)]^c. \quad (2)$$

```
/* List F contains the local maximum components */
while (F not empty) do
  {
     extract C from F;
     area =  A(C);
     curlevel = grey level of component;
     while (area < lambda)
       { n = neighbor of C with I[n]
             is maximum of all neighbors;
         if (I[n] > curlevel)
           break;
         else { add n to C;
                curlevel = I[n];
              }
       }
     for all p in C do
       {  I[p] = curlevel;
          L[p] = PROCESSED;
       }
  }
```

Fig. 3. The core of Vincent's area opening algorithm. The parameter `lambda` in the code equals the area threshold λ. For closing, the local minima components are scanned, the minimum neighbor of C must be sought, and the test before the break must be $I[n] < curlevel$.

The definition of an area opening of a gray-scale image f is usually derived from binary images $T_h(f)$ obtained by thresholding f at h. These are defined as

$$T_h(f) = \{x \in \mathbf{M} | f(x) \geq h\}. \tag{3}$$

Definition 2. *The area opening for a mapping $f : \mathbf{M} \to \mathbb{R}$ is given by*

$$(\gamma_\lambda^a(f))(x) = \sup\{h | x \in \Gamma_\lambda^a(T_h(f))\}. \tag{4}$$

The gray-scale area closing ϕ_r^a is defined by using a duality relationship similar to (2)

$$\phi_\lambda^a(f) = -\gamma_\lambda^a(-f). \tag{5}$$

Thus, the area opening of an image assigns each point the highest threshold at which it still belongs to a connected foreground component of area λ or larger. The area closing assigns each point the lowest threshold at which it belongs to a connected background component of area λ or larger.

3 Computing Area Openings and Closings

Before describing the individual algorithms, we first define a *flat zone* L_h at level h of a gray-scale image f as a connected component of the set of pixels $\{p \in \mathbf{M} | f(p) = h\}$. A *regional maximum* M_h at level h is a level component no members of which have neighbors larger than h. A *peak component* P_h at level h is a connected component of the thresholded image $T_h(f)$. At each level h, there may be several such components, which will be indexed as L_h^i, P_h^j, and M_h^k, respectively, with i, j, and k from some index set. Any regional maximum M_h^k is also a peak component but the reverse is not true.

3.1 The Pixel-Queue Algorithm

The pixel-queue algorithms for morphological area and attribute operators are given in some detail elsewhere [5], [20], [21], so we will describe them only briefly here. The source code of our implementations is available on request. The version described here only performs area openings, but more general attribute openings require little modification [5].

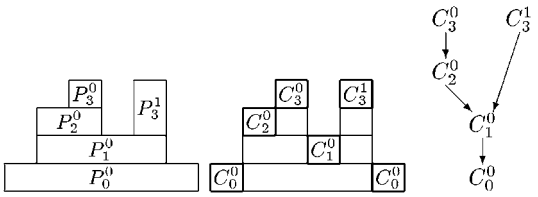

Fig. 4. The Max-tree structure: the peak components P_h^k of a 1D signal level image X (left); the corresponding pixels in each Max-tree node C_h^k (middle); and the Max-Tree itself (right).

Briefly, the image is first scanned using a pixel queue to create a list of all regional maxima M_h^k. After this, all M_h^k are processed sequentially. This is done by growing a peak component $P_{h_1}^j, h_1 \leq h$ around a seed pixel within the maximum M_h^k using a priority queue. As each pixel is added to the growing region, its neighbors which do not (yet) belong to the region are put in the priority queue, from which they are retrieved in reverse gray-level order (highest first). The process of adding pixels continues until the area is larger than λ or when the next pixel taken from the priority queue has a gray-level h'' larger than the current level h'. In the first case, the peak component is large enough, and should be set to the current gray-level h'. In the latter case, the region grown so far is not a peak component $P_{h'}^j$ *at level* h'. Note that, in this case, the component may be flooded again later on. In either case, the gray levels of all pixels $p \in P_{h'}^j$ are set to h' and M_h^k is removed from the list. The algorithm terminates when all maxima have been processed.

The part of the algorithm after the regional maxima have been found is shown in pseudocode in Fig. 3

3.2 The Max-Tree Approach

Max-trees were introduced by Salembier et al. [16] as a versatile data structure for antiextensive connected set operators. A Max-tree is a rooted tree in which each of the nodes C_h^k corresponds to a peak component P_h^k. However, C_h^k contains only those pixels in P_h^k which have gray-level h. In other words, it is the union of all $L_h^j \subseteq P_h^k$. Each node except for the root points towards its parent $C_{h'}^j$ with $h' < h$. An example of a 1D signal, its peak components and its Max-tree nodes is shown in Fig. 4. It can be seen that multiple flat zones may be contained in a single node, as in the case of C_0^0 in this example. Furthermore, in a given branch, not all gray levels have to be occupied, as in the case of the branch ending at C_3^1.

In the case of the area opening, the area of the peak component P_h^k is stored in the Max-tree structure in node C_h^k. Once a Max-tree of an image has been computed, computing an area opening reduces to removing all nodes which have an area smaller than λ from the tree. Because the area must increase when descending any branch towards the root, the opening reduces to pruning branches at the appropriate points and merging the pixels in the removed nodes with their smallest surviving ancestor.

Salembier et al. [16] use an array `STATUS` of the same size as the image to determine to which node a pixel belongs. A pixel

p with gray-level h belong node C_h^k if STATUS[p] $= k$. Initially, all elements in STATUS are set to NOTPROCESSED(< 0). Besides this array, the algorithm uses a hierarchical queue of G levels, with G the number of gray levels. One array of G integers called NodesAtLevel is used to store the number of Max-tree nodes detected so far at each gray level. A further array of G Booleans NodePresent stores at which levels below the current gray-level nodes have been detected in the path from current node to root. Finally, we store the Max-tree itself in an array of N nodes (with N the number of pixels in the image).

In our implementation, we first compute a histogram of the image, in order to compute the space needed in each of the levels of the hierarchical queue and maximum possible number of nodes in each of the levels of the Max-tree. All elements of the NodesAtLevel array are set to zero, those of the NodePresent array to false. Then, the Max-tree is built by first selecting a pixel with the smallest gray-level hmin which must belong to the root of the tree and inserting it at level hmin in the hierarchical queue, and NodePresent[hmin] is set to true. Using this pixel as a seed, a recursive flood filling process is started. At each step, a pixel p at the highest occupied level h in the hierarchical queue (starting at hmin obviously) is retrieved. Then STATUS[p] is set to NodesAtLevel[h], which labels it as a member of C_h^k, and the area in the appropriate Max-tree node is incremented. Then, all neighbors of p for which STATUS[n] is NOTPROCESSED are put in the queue at level I[n], STATUS[n] is set to INTHEQUEUE(< 0), and NodePresent[I[n]] is set to true. If the neighbor's gray-level I[n] is larger than the current level h the flooding proceeds at level I[n]. If, at a given level h, no more pixels can be retrieved from the queue at level h, NodePresent[h] is set to false, and the parent node is sought by finding the element in NodePresent with the highest index m which is true. Once m has been determined, the parent node $C_{h'}^j$ is found by letting $h' =$ m, and $j =$ NodesAtLevel[m]. The current nodes area is added to its parent and flooding proceeds at level m. Once all pixels have been processed, the Max-tree is complete.

The filtering is carried out by visiting all nodes of the tree once from low gray level to high. We can do this because we store the nodes sorted per gray level in an array and we know how many nodes exist at each level. For each node, we check whether its area is smaller than λ. If so, its output gray level is set to that of its parent (which has already been assigned the correct gray level). The output image O is made by visiting all pixels p in the image, determining its node from I[p] and STATUS[p], and assigning the the output gray level of that node to O[p].

For details and pseudocode, the reader is referred to [16].

3.3 The Union-Find Method

A key property of the algorithms described above is that they process the image one peak component at a time. Furthermore, both use a flooding method. The approach described in this section can process multiple peak components simultaneously. Pixels are processed in gray-level order. During this process, peak components are created and merged as needed, while keeping track of their areas. Once a peak component has an area of at least λ, it ceases to grow.

Tarjan's union-find algorithm for keeping track of disjoint sets [18] is used to implement merging in an efficient way. For each set, an arbitrary member is chosen as representative for that set. The algorithm uses rooted trees to represent sets in which the root is chosen as the representative. Each nonroot node in a tree points to its parent, while the root points to itself. Two objects x and y are members of the same set if and only if x and y are nodes of the same tree, which is equivalent to saying that they share the same root of the tree they are stored in. There are four basic operations.

- MakeSet(x): Create a new singleton set {x}. This operation assumes that x is not a member of any other set.
- FindRoot(x): Return the root of the tree containing x.
- Union(x,y): Form the union of the two sets that contain x and y.
- Criterion(x,y): a symmetric criterion which determines whether x and y belong to the same set.

For flat zone labeling the algorithm becomes:

```
for all pixels p do
  { MakeSet(p);
   for all neighbors n<p do
     if ( I[n]==I[p] )
      Union( n, p );
  }
```

Note that, in this context, the condition n<p means that n is a pixel which has been processed before p. In this case, Criterion(n,p) is true if the image value I[n] equals I[p]. Union uses FindRoot internally to determine the root nodes of the trees containing n and p. After this scan, a second "resolving" scan assigns each root pixel a unique label and to each nonroot pixel the label of its root.

Before going into the details of the area opening algorithm itself, we will discuss the general framework for storing the disjoint sets and the auxiliary functions needed for attribute openings and closings.

The disjoint sets we have to find are all flat zones $L_h^i \subset \Gamma_\lambda^a(X_h(f))$, which are not altered by the area opening γ_λ^a, and, for all other L_h^i, the smallest peak component $P_{h'}^j \supset L_h^i$ which has area λ or more. To store the trees representing these sets for the entire image, we use an integer array parent of the same size as the image (i.e., N) in which parent[p] is the parent of pixel p. Pixels are stored as width*y+x, with x and y the pixel's x and y coordinates, and width the image width. If a pixel is a root of a tree, i.e., it has no parent, we flag this by setting parent[p] < 0, rather than letting it point to itself. We could use an auxiliary array area in which area[p] (for root nodes) stores the area of each set. However, for a set of area A with root node p, we can also set parent[p] to $-A$, which saves memory space. We define an *active root* as a root of a peak component with area smaller than λ.

The code for the MakeSet, FindRoot, Criterion, and Union routines is shown in Fig. 5. In this case, the

```
void MakeSet ( int x )
{ parent[x] = -1;
}

int FindRoot ( int x )
{ if ( parent[x] >=0 )
    { parent[x] = FindRoot( parent[x] );
      return parent[x];
    }
  else return x;
}

boolean Criterion ( int x, int y )
{ return ( (I[x] == I[y]) ||
           ( -parent[x] < lambda ) ) ;
}

void Union ( int n, int p )
{ int r=FindRoot(n);
  if ( r != p )
    { if ( Criterion(r, p) )
        { parent[p] = parent[p] + parent[r];
          parent[r] = p;
        }
      else
        parent[p] = -lambda;
    }
}
```

Fig. 5. Implementation of the basic operations for area openings and closings. Note that the areas of components are stored as negative numbers in the corresponding roots. The variable `lambda` is equal to the parameter λ. The parameters for `Criterion` must be root nodes.

`Criterion` and `Union` routines are asymmetrical. This is done to ensure that, if the set we are dealing with is a peak component P_h at level h, the root element `r` has a gray-level `I[r]`$= h$. Therefore, we process the pixels in decreasing gray-level order, and always make the last pixel processed the root of the new tree. We do this by sorting the pixels using counting-sort and storing the coordinates in an array `SortPixels` of length N. Pixels of the same gray level are processed in scan line order. Scanning of peak components from high to low gray levels is guaranteed, without finding regional maxima explicitly.

As each pixel `p` is processed, the `MakeSet` routine labels `p` as a singleton set, setting `parent[p]` to -1. The `Union` procedure is now called for each neighbor `n` which has already been processed. We briefly describe this procedure here. Since `p` is *always* a root, `FindRoot` is only called to find the root pixel `r` of `n`. If `r` equals `p` nothing needs to be done, because `n` and `p` are already in the same set. Otherwise, `Criterion` is called with `r` and `p` as parameters. If the gray-level `I[r]` of `r` is equal to that of `p` *or* if `r` is an active root, `Criterion` returns "true" and the two trees are merged. Merging is done by adding the area of `r` to that of `p`, and making `p` the parent of `r`. If `Criterion` returns "false", a neighbor has a root gray level higher than `I[p]` *and* has a sufficiently large area and, therefore, `p` $\in L_h^i \subset \Gamma_\lambda^a(X_h(f))$. In this case, `p` is made inactive by setting `parent[p]` to $-\lambda$.

Note that the `FindRoot` routine, apart from finding the root of a tree, also performs path compression. This is a technique that was used by Tarjan to reduce the average cost of `FindRoot`. It does this by setting the parent pointer directly to the root for all pixels visited along the path to the root.

```
/* array S contains sorted pixel list */
for (p=0; p<Length(S); p++)
  {
    pix = S[p];
    MakeSet(pix);
    for all neighbors nb of pix do
      if ((I[pix] < I[nb]) ||
          ((I[pix] == I[nb]) && (nb<pix)))
        Union(nb,pix);
  }
/* Resolving phase in reverse sort order */
for (p=Length(S)-1; p>=0; p--)
  {
    pix = S[p];
    if (parent[pix] >= 0)
        parent[pix] = parent[parent[pix]];
    else
        parent[pix] = I[pix];
  }
```

Fig. 6. Code showing how to perform an area opening using the operations of Fig. 5.

At the end of this part of the algorithm, we have found two kinds of disjoint sets: 1) those with constant gray level, which are level components $L_h^i \subset \Gamma_\lambda^a(X_h(f))$ and 2) those with varying gray level, which are peak components P_h^j, with h the maximum gray value for which the area criterion is satisfied. Because the root r of these peak components is always the last pixel processed, its gray level in the input image satisfies $f(r) = h$. Therefore, if we set the gray level of each pixel in the output image to that of its root in the input image, all $L_h^i \subset \Gamma_\lambda^a(X_h(f))$ remain unchanged, whereas all P_h^j are filled uniformly with a gray level of h. This can be done in linear time. The most memory efficient approach is to store the output image in the `parent` array, which is possible because we can visit the pixels in reverse processing order. For each pixel `p`, we inspect its parent. If `parent[p]` is negative, `p` is a root, and `I[p]` is the correct gray level for the component. If `parent[p]` is nonnegative, then it is pointing to a pixel in `parent` which has already been resolved, and already has been assigned the correct root gray level. In Fig. 6, the above described algorithm is listed.

For area closings, we must change each test for gray level in the main loop of the routine to a test for *smaller than*. The `FindRoot` and `Criterion` procedures and the resolving stage need not be changed.

4 Performance Testing and Results

The three algorithms were implemented in C. For efficiency reasons the tail recursion in `FindRoot` was removed and all procedures in Fig. 5 were inlined. The code is available on request. The algorithms were tested on a range of synthetic images with different sizes and numbers of extrema. Furthermore, a number of natural images at various sizes was used. Mean CPU times over multiple runs on HPPA RISC-processor-based machines were determined for each image and for different values of λ.

Three-dimensional versions of the algorithms for volume openings were also implemented and tested on a number of angiograms.

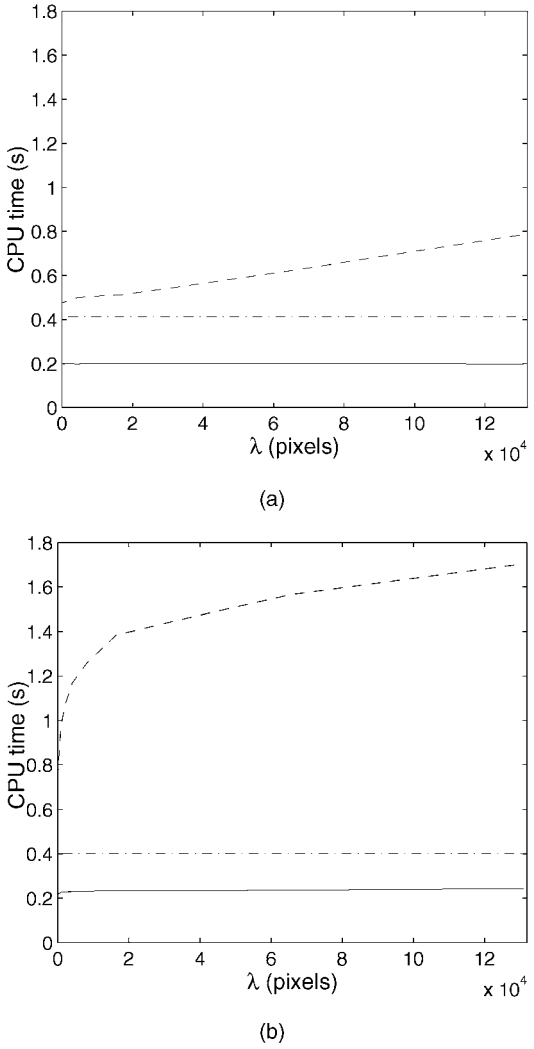

Fig. 7. Timing results for distance map images. CPU timing for the priority-queue algorithm (dashed), the union-find method (solid), and the Max-tree algorithm as a function of area (λ) of the closing for 512×512 distance map images for different numbers N_{min} of local minima: (a) $N_{min} = 2$; (b) $N_{min} = 2000$. Note the strong dependence of the timings of the priority-queue algorithm on both image content (N_{min}) and λ. The timings for the union-find and Max-tree methods are practically independent of λ, and only slightly dependent on N_{min}.

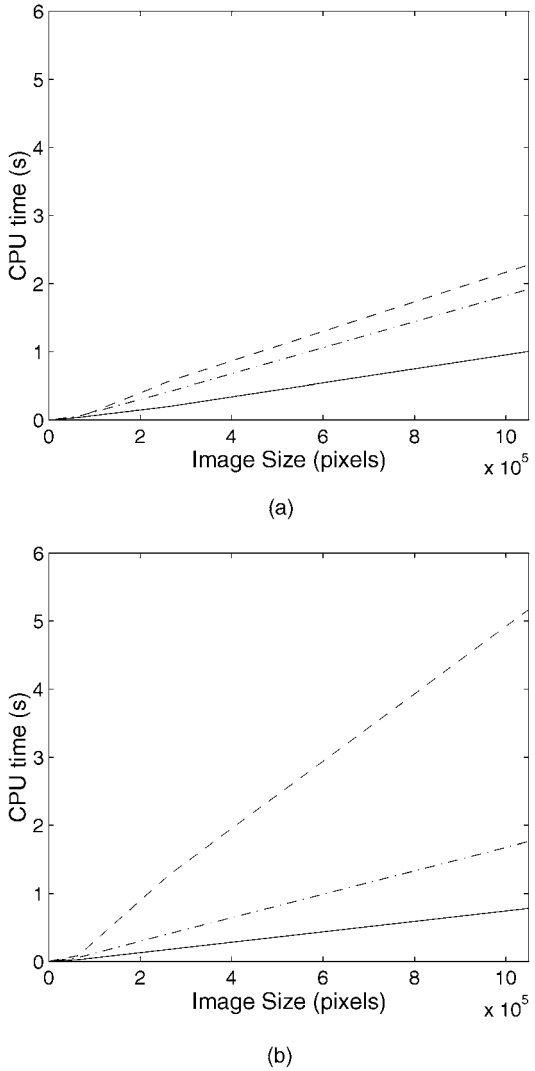

Fig. 8. Image size dependence of computing times for distance map images for the pixel-queue (dashed), Max-tree (dash-dot), and union-find (solid) algorithm for $\lambda = 256$ and different values of the density α of local minima: (top) $\alpha = 1/1,024$, (bottom) $\alpha = 100/1,024$.

4.1 Synthetic Images

To test the sensitivity of the algorithm to the image complexity in a controlled manner, synthetic images were used. To generate these, N_{min} randomly placed dots on a black background were generated. After this, Euclidean distance maps were generated from these random dot images. These distance map images allow controlled testing of the performance of each algorithm as a function of the number of extrema (minima) of roughly circular shape. Because the distance map images contain controllable numbers of minima, we decided to perform the timings on closings rather than openings. However, we could in principle have inverted these images and done the timings on openings, obtaining the same results.

The CPU times for area closings with increasing λ for 512×512 distance map images with different numbers of local minima M are shown in Fig. 7. Quite clearly, the CPU times for the priority-queue algorithm depend strongly on both λ and N_{min}, unlike either the union-find method or the Max-tree method. In all cases, the union-find method outperformed the priority-queue algorithm, by a factor ranging from 2.4 for small λ, to about seven for $\lambda = 131,072$ and $N_{min} = 2,000$. The λ dependency of the priority queue algorithm is clearly nonlinear for smaller λ. By contrast, the λ dependence of the modified Tarjan method is very weak, rising by at most 17 percent over λ ranging from 2 to 131,072. The Max-tree algorithm has no significant λ-dependence. However, it is slower than the union-find approach by a factor of 2.1 at small λ and 1.65 at $\lambda = 131,072$ for $N_{min} = 2,000$. For the union-find method timings range from 0.200 s to 0.244 s from $N_{min} = 2$ to $N_{min} = 2,000$, or an 22 percent increase for $\lambda = 131,072$. The Max-tree method shows no significant change over the same range. For $\lambda = 131,072$, Vincent's algorithm shows an increase in CPU time from 0.786 s to 1.704 s, or a 117 percent increase, over the same range of N_{min}.

To measure dependence of CPU times on the image size N, distance map images of different sizes with the same density α of minima ($\alpha = N_{min}/N = constant$) were used. The results are shown in Fig. 8. All three methods appear to

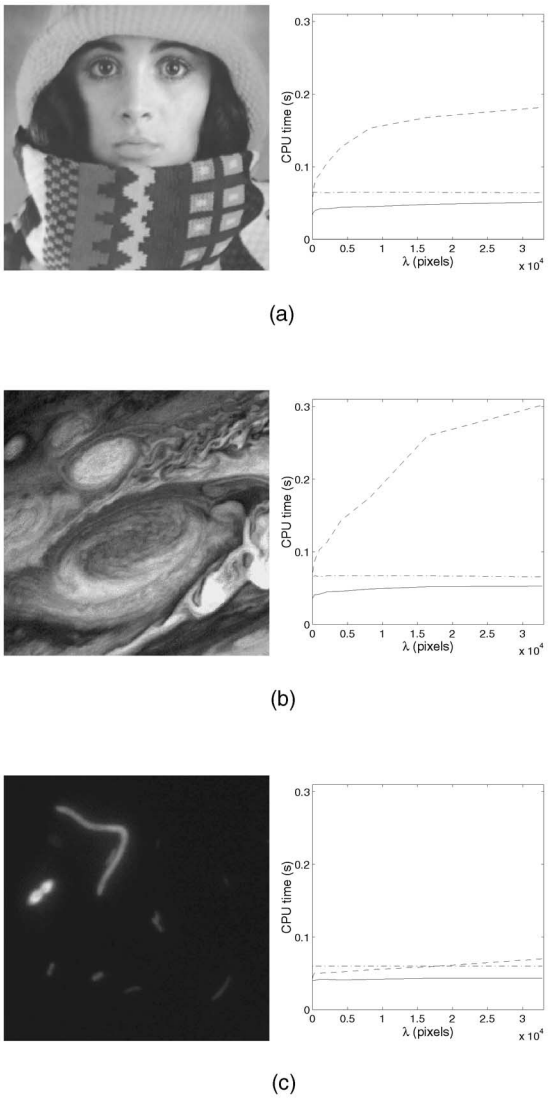

Fig. 9. Some natural images used for performance testing algorithms for area closings and the corresponding results for the Max-tree algorithm (dash-dot), priority-queue (dashed), and the union-find approach (solid): (a) face, (b) red spot, and (c) bacteria.

be linear in N for all practical purposes. On certain machines, deviations were observed for very small images, probably due to caching.

4.2 Natural Images and Volume Openings

A selection of 40 natural images with different image characteristics was used to assess the performance under more realistic conditions. The images include images of faces, a house, landscapes, airplanes, an aerial photograph of a city, various astronomical images, and microscopic images of diatoms, bacteria and skin tissue. Some of these images are shown in Fig. 9.

The results are similar to those for synthetic images. CPU times for the union-find and Max-tree methods depend mainly on image size, and only slightly on image content or λ. By contrast, CPU times for the priority-queue algorithm depend strongly on both image content and λ. The image of Jupiter's red spot in Fig. 9b shows a very strong λ-dependence at all scales. This may be caused by the many nested structures in this image.

In certain images, such as the image of bacteria in Fig. 9c, the dependence of CPU time on λ is only slight. Indeed, based on this image alone there would be little reason to prefer any algorithm. However, even here, the union-find method is faster than the others for all $\lambda > 8$. In this case alone, it is the Max-tree method which is slowest, except for $\lambda > 16,384$.

The volume openings yielded slightly different results. On a $128 \times 128 \times 62$ volume, the union-find method (CPU time 1.29 s) is 3.5 times faster than the Max-tree (4.51 s), and 3.3 times faster than the priority-queue method (4.26 s), both for $\lambda = 256$. At $\lambda = 131,072$, the union-find method takes 1.34 s, Max-tree algorithm 4.53 s, and the priority-queue method falls far behind with 72 s. We performed the same tests on the $256 \times 256 \times 256$ volume of Fig. 2 on a Pentium II at 400 MHz, with 640 MB RAM. At $\lambda = 256$ and $\lambda = 131,072$, respectively, the union-find method needs 6.35 s and 6.68 s, the Max-tree algorithm needs 20.37 s and 20.47 s, and the priority-queue method needs 19.01 s and 108 s. Apparently, the advantage of the union-find algorithm is greater in these 3D data sets. This is probably due to its smaller memory requirements, especially given the limited bandwidth of the memory bus of personal computers, as is discussed in the next section.

5 Computational Complexity and Memory Use

In the case of the union-find approach, complexity analysis is straightforward if we assume that the union-find part is the most costly. This is the case when we can use a linear time sorting algorithm as counting-sort, which only works well for 8-16 bit integers. Even though $O(N)$ sorting algorithms exist for floating point numbers or larger integer ranges (e.g., radix-sort), it may be quicker to use one of many $O(N \log N)$ methods such as heap-sort. For an image of N pixels, using C-connectivity, we perform N `MakeSets` and at most $NC/2$ `FindRoots`. This is because for all pixels we do a `FindRoot` on those of the C neighbors which have been processed before the current pixel. Disregarding image edge effects, one half of the neighbors of each pixel will have been processed beforehand on average. For these numbers of `FindRoots` and `MakeSets`, Tarjan [18] derives a worst case complexity of $\Theta(NC \log_{1+C/2} N)$, or $\Theta(N \log N)$ for fixed connectivity, when path compression is used, as is the case in our algorithm. In our measurements, CPU-time was a linear function of image size N, though an $N \log N$ relationship cannot be ruled out from the data.

Explicitly including the connectivity in the computational complexity is particularly necessary when contemplating extension to more than two dimensions. Extension to three dimensional images is straightforward, since all that has to be changed is the way pixel coordinates are stored in integers, and 6, 18, or 26 connectivity must be used. This does increase the computing time somewhat, but not prohibitively.

The computational complexity of the priority-queue algorithm is $O(N\lambda \log \lambda)$, which becomes $O(N^2 \log N)$ if $\lambda = N$. The latter is the case when computing pattern spectra (see Section 7). An example of this $O(N^2 \log N)$ behavior is seen when the number M of regional maxima in

an image is maximal. In the case of 4-connectivity, we have $M = N/2$ points in a checkerboard pattern. Suppose all these points have the maximum gray level, that the nonregional maximum pixels are assigned strictly descending gray levels in scan-line order and that $\lambda = N$. As each of the $N/2$ maxima is processed, *all* the previously processed pixels are flooded, before the algorithm encounters the next regional maximum, and stops the flooding process. Therefore, on average, a region of $N/2$ pixels is flooded for each of the $N/2$ maxima. Each visit to a pixel bears a cost of $O(\log N)$ due to the use of a priority queue. Therefore, CPU times are expected to increase proportional to $N^2 \log N$ in this case. This has been verified experimentally [24].

The computational cost of the Max-tree algorithm is dominated by the flood filling process which is inherently linear in both the number of pixels and in the connectivity. Pruning the Max-tree requires visiting all nodes in the tree at most twice, so it is also $O(N)$. Computing the output image is also linear in the number of pixels.

One somewhat peculiar feature in the computational complexity of the Max-tree algorithm is its dependency on the number of unoccupied gray levels between parent and child nodes. As stated, the flood filling procedure uses an array of Booleans to store which levels below the current are occupied. Insertion into this array is done in constant time, but retrieval of the highest gray level below the current takes a time proportional to the number of unoccupied levels between Max-tree nodes in the same branch. Thus, a contrast stretch followed by a Max-tree filter is slower than the reverse order. We tested this in a particular bad case: A checkerboard pattern consisting of alternating pixels either of gray levels 0 and 1 or of 0 and 255. In this case, $N/2$ maxima must be processed. For each maximum there are four insertions into the array, followed by one retrieval. In the first case, the retrieval needed to inspect only one element, in the second 255 elements. For a 256×256 image, the first case took 80 ms, the second 222 ms. The other two algorithms show no differences in CPU times, as expected. It is possible to replace the Boolean array by other data structures in which retrieval is cheaper, such as a heap ($\log n$), but only at the expense of higher insertion cost. Since the number of insertions is at least C times greater than the number of retrievals, the original choice appears to be the best.

Apart from computational complexity, memory use must be taken into account, not just to estimate whether enough resources are available for a given image size, but also because memory bandwidth may actually limit the processing speed if large (volume) data sets are used. Apart from the original image, the priority-queue method requires a queue for detection of the regional maxima, which in our implementation requires N integers (with N the image size), a priority queue of either 2λ or N integers, depending on which value is the smaller, a label image of N integers, a fill list of λ (worst case N) integers to store the pixels belonging to the current maximum, and optionally an output image (the standard implementation overwrites the original image, which may not be desirable). The union-find method uses only the parent array (which becomes the output image) of N integers and the sorted pixel array (also N integers), apart from the input image. Therefore, it yields a memory saving of at least $2N$ integers with respect to the priority-queue method and possibly an image of N pixels, as well, if a separate output image is used in the priority-queue approach. The Max-tree approach is the most costly, memory-wise. The hierarchical queue and status arrays each require N integers. The Max-tree itself may have N nodes (worst case), each of which contains its area, pointer to its parent, and the output gray level, or $2N$ integers and N pixels. An output image may also be required, if the input image may not be overwritten. The savings obtained by using the union-find approach compared to the Max-tree method is $2N$ integers and N pixels (or $2N$ pixels if a separate output image is used).

These differences in memory use are particularly important for 3D analysis. In the case of the $128 \times 128 \times 62$ volume, the total memory use of the Max-tree method was 57.5 MB, priority-queue methods needed 45 MB, whereas the union-find algorithm required only 25 MB.

6 Extension to Attribute Openings

All three algorithms can be extended to compute other attribute openings and closings. The priority-queue approach has been extended by Breen and Jones [5], and a union-find algorithm based on the area opening version described here has also been published [24]. The original Max-tree algorithm can be used for not just attribute openings and closings, but also for thinnings and thickenings [16].

Unlike the area of a set, many attributes cannot be computed "on the fly." Therefore, the versions for attribute openings of all three algorithms rely on auxiliary data sets per peak component which has been processed partially. The auxiliary data are chosen in such a way that they can be updated pixel by pixel easily and that the desired attribute can be computed efficiently. In the case of the union-find and Max-tree approaches, easy merging of auxiliary data sets of different connected sets of pixels must also be possible. For example, consider the opening using the moment of inertia $I(C)$ of connected set C, which is defined in 2D as

$$I(C) = \sum_{(x,y)\in C} (x - \bar{x})^2 + (y - \bar{y})^2 \tag{6}$$

with x and y the pixel coordinates and $\bar{x}$ and $\bar{y}$ their mean values. In this case, it can be shown that the area $A(C)$, $\sum x$, $\sum y$, $\sum x^2$, and $\sum y^2$ as auxiliary data are sufficient. As pixels are added to the set, the sums and area are updated in constant time. Merging two data sets reduces to adding the area and sums.

A number of adaptations to data structures and algorithms are needed in both the priority-queue and union-find approaches. First of all, in both cases the attributes are only computed when a peak component has completely been processed, i.e., when a gray-level boundary is crossed. In the pixel-queue approach, this occurs when a pixel with lower gray level is retrieved from the priority queue. The attribute is computed, compared to λ and if it is smaller the flooding proceeds, otherwise, the component is filled with the current gray level. Because the priority queue algorithm processes one peak component at a time, only one auxiliary data set is needed. Memory-wise this is the optimal solution.

Breen and Jones [5] use a single "driver" routine with function pointers as arguments to be able to compute any attribute opening, without code duplication. Their method requires functions to create, destroy, add a pixel to an auxiliary data set, and computing the attribute value. The union-find version uses a similar approach, but a merge function is also needed.

In the union find case, we need some more adaptations. First of all, it is no longer possible to store the attribute value as a negative number in the root node parent array itself. Instead, negative values (`ACTIVE` and `INACTIVE`) are used to flag active and inactive root nodes of sets. A second array `auxdata` is used to store pointers to auxiliary data sets. Only active root nodes have valid auxiliary data sets assigned to them. Because multiple peak components are processed simultaneously, $N/2$ auxiliary data sets are needed in the worst case. Computing attributes is done when the next pixel in the sorted pixel array has a lower gray level than the current. In that case, all attributes of all active peak components at the current gray level are computed and compared to λ. All peak components which have attribute larger than lambda are set to inactive. For details, see [24].

In the Max-tree case, similar adaptations are needed with respect to the area opening algorithm presented here. In this case, though only a single peak component is flooded at a time, it may have collected data at multiple gray levels. In the worst case, G sets (with G the number of gray levels) are needed. Though still worse than the priority-queue method, this is considerably less than the worst case in the union-find approach. In this scheme, a Max-tree node stores a pointer to its parent, the attribute value, and gray level in the output image as before. It does not need a pointer to its auxiliary data, because we can store these in an array of length G, and use the current gray level of the node to reference the correct data set.

We have performed timings of the three algorithms for the moment-of-inertia opening, and the relative results are very similar to those obtained with the area opening. On 256×256 natural images, the union-find method took some 155 ms, compared to 33 ms for the area opening. The Max-tree approach took some 200 ms, and the priority-queue approach 540 ms at $\lambda = 32,768$.

7 Extension to Pattern Spectra

All three algorithms can readily be adapted to computing area opening and closing pattern spectra efficiently. For the priority-queue method, this has been done by Breen and Jones [5] and a similar extension has been done for the union-find case [15].

Let f and g be gray-scale images. A size distribution or granulometry is a set of operators $\{\alpha_r\}$ with r from some totally ordered set Λ (usually $\Lambda \subset \mathbb{R}$ or $\mathbb{Z}$), with the following three properties [13], [23]:

$$\alpha_r(f) \leq f \tag{7}$$

$$f \leq g \Rightarrow \alpha_r(f) \leq \alpha_r(g) \tag{8}$$

$$\alpha_r(\alpha_s(f)) = \alpha_{\max(r,s)}(f) \tag{9}$$

for all $r, s \in \Lambda$. Since (9) implies idempotence, it can be seen that size distributions are openings. The pattern spectrum S_f^α obtained by applying a size distribution α_r to gray-scale image f can be defined as the integral of the gray level of $\alpha_r(f)$ over the image domain. In the discrete case, we have:

$$S_f^\alpha(r) = \sum_{x \in \mathbf{M}} (\alpha_r(f))(x). \tag{10}$$

A trivial algorithm for computation of area pattern spectra consists of performing area openings at $N_s \leq N$ scales, ranging from $\lambda = 1$ to $\lambda = N$, and computing the sum of gray levels of the resulting image at each scale and storing the results in an array of N_s bins. This method yields a worst-case computational complexity of $O(N_s N^2 \log N)$ for the priority-queue algorithm and $O(N_s N \log N)$ for the union-find approach. In both cases, it is possible to reduce the computational cost by a factor of N_s, by effectively performing an area opening with $\lambda = N$, which will merge all flat-zones of the image. During the merging process, the spectrum is updated as each peak component is merged with flat zones at a lower gray level (for details, see [5], [15]). If a peak component of area A is merged with a set at a lower gray level, a function `Bin` is used to map the area to the range of bin numbers in the spectrum $(0, 1, \ldots, N_s)$. These adaptations can be seen in Fig. 10 for the union-find case. It has been demonstrated that it is indeed possible to compute an area pattern spectrum in the time needed for a single area opening. A pattern spectrum version of the Max-tree algorithm is also trivial to implement. In this case, the tree-traversal used for computing the gray levels of the output image is used to update the pattern spectrum as above.

The three algorithms have been implemented and yield CPU times indistinguishable from performing a single area opening with the same method with $\lambda = N$ in all cases tested (data not shown).

8 Conclusions

The Max-tree and union-find algorithms for computation of area openings and closings have clear advantages over the priority-queue algorithm in terms of computing time, especially when using large values of λ. Furthermore, the strong dependence of computing time on image content seen in the priority-queue algorithm is lacking in both. Though theoretically less computationally efficient, the union-find approach outperforms the Max-tree method, especially in large (3D) data sets. This may be caused by the much larger memory requirements of the Max-tree approach, and its less regular access of memory due to the flooding process.

In the case of the area opening, the union-find is also the most efficient in terms of memory usage, especially if the original image may not be overwritten. In that case, it uses between 2.3 and 3 times less memory than the Max-tree approach, and about 40-50 percent less than the priority-queue method (depending slightly on the size of a pixel). However, in the more general case of attribute operators, the union-find approach requires more space for auxiliary data sets in the worst case. If auxiliary data sets required to compute the attributes are large, the priority-queue method in particular may be more efficient in terms of memory usage.

The union-find algorithm is suitable for parallel implementation on shared-memory systems, since it processes extrema concurrently, while the other algorithms process extrema sequentially. Besides, the algorithm does not use a

```
void Union ( int n, int p )
{ int r=FindRoot(n);
  if ( r != p )
    { if ( I[p]!=I[r]
        { spec[Bin(-parent[r])] -=
            (I[r]-I[p]) * parent[r];
        }
      parent[p] = parent[p] + parent[r];
      parent[r] = p;
    }
}
```

```
  /* array S contains sorted pixel list */
  for (i=0;i<Ns;i++)
    spec[i]=0;
  greysum=0;
  for (p=0; p<Length(S); p++)
    {
      pix = S[p];
      greysum += I[p];
      MakeSet(pix);
      for all neighbors nb of pix do
        if ((I[pix] < I[nb]) ||
            ((I[pix] == I[nb]) && (nb<pix)))
          Union(nb,pix);
    }
  for (i=0;i<Ns;i++)
    { spec[i] = greysum - spec[i];
      greysum = spec[i];
    }
```

Fig. 10. Pseudocode showing how to compute an area opening pattern spectrum: (top) modified `Union` procedure, in with the function `Bin` maps the range of areas to the range of bins; (bottom) code of the area pattern spectrum algorithm itself. The `FindRoot` procedure is left unchanged (see Fig. 5).

queue based flooding process, which is particularly difficult to parallelize. It is not hard to augment the union-find algorithm with locking-primitives to avoid concurrent accesses to shared memory locations, as in the case of connected component labeling [11].

Unlike the current implementations of the priority-queue and union-find algorithms, the Max-tree method can be used for attribute thinnings as well. The separation of Max-tree construction, Max-tree filtering and image restitution is elegant and very versatile. However, there is no particular reason to use a flooding approach to construct the Max-tree. Indeed, it is slightly impractical, because merging two sets of pixels belonging two different nodes requires reflooding one of the sets and setting all gray levels and `STATUS` values correctly. Therefore, we are working on a union-find approach to building and filtering Max-trees, in which merging nodes can be done in constant time. This algorithm will also be designed for parallel execution on shared-memory systems.

Finally, the disjoint set approach discussed here could be generalized to other and possibly all connected filters. The reason for this is that all connected filters operate on the flat zones of an image, merging them, or changing their intensities, but never splitting them. Tarjan's union-find method provides an efficient means to perform such operations.

Acknowledgments

This work was partly funded by the European MAST programme, contract MAS3-CT97-0122.

References

[1] J.A. Bangham, P. Chardaire, C.J. Pye, and P.D. Ling, "Multiscale Nonlinear Decomposition: The Sieve Decomposition Theorem," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 18, pp. 529-538, 1996.

[2] J.A. Bangham, R. Harvey, P.D. Ling, and R.V. Aldridge, "Morphological Scale-Space Preserving Transforms in Many Dimensions," *J. Electronic Imaging,* vol. 5, pp. 283-299, 1996.

[3] J.A. Bangham, R. Harvey, P.D. Ling, and R.V. Aldridge, "Non-Linear Scale-Space from N-Dimensional Sieves," *Proc. IEEE European Conf. Computer Vision '96,* pp. 189-198, 1996.

[4] J.A. Bangham, P.D. Ling, and R. Harvey, "Scale-Space from Nonlinear Filters," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 18, pp. 520-528, 1996.

[5] E.J. Breen and R. Jones, "Attribute Openings, Thinnings, and Granulometries," *Computer Vision and Image Understanding,* vol. 64, no. 3, pp. 377-389, 1996.

[6] F. Cheng and A.N. Venetsanopoulos, "An Adaptive Morphological Filter for Image Processing," *IEEE Trans. Image Processing,* vol. 1, pp. 533-539, 1992.

[7] J. Crespo, R.W. Schafer, J. Serra, C. Gratin, and F. Meyer, "The Flat Zone Approach: A General Low-Level Region Merging Segmentation Method," *Signal Processing,* vol. 62, pp. 37-60, 1997.

[8] M.B. Dillencourt, H. Samet, and M. Tamminen, "A General Approach to Connected-Component Labeling for Arbitrary Image Representations," *J. ACM,* vol. 39, pp. 253-280, 1992.

[9] C. Fiorio and J. Gustedt, "Two Linear Time Union-Find Strategies for Image Processing," *Theoretical Computer Science A,* vol. 154, pp. 165-181, 1996.

[10] H.J.A.M. Heijmans, "Connected Morphological Operators for Binary Images," *Computer Vision and Image Understanding,* vol. 73, pp. 99-120, 1999.

[11] W.H. Hesselink, A. Meijster, and C. Bron, "Concurrent Determination of Connected Components," *Science of Computer Programming,* vol. 41, pp. 173-194, 2001.

[12] P.T. Jackway and M. Deriche, "Scale-Space Properties of the Multiscale Morphological Dilation-Erosion," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 18, pp. 38-51, 1996.

[13] G. Matheron, *Random Sets and Integral Geometry.* John Wiley, 1975.

[14] A. Meijster and J.B.T.M. Roerdink, "A Disjoint Set Algorithm for the Watershed Transform," *Proc. EUSIPCO '98, IX European Signal Processing Conf.,* pp. 1665-1668, 1998.
[15] A. Meijster and M.H.F. Wilkinson, "Fast Computation of Morphological Area Pattern Spectra," *Proc. Int'l Conf. Image Processing 2001,* pp. 668-671, 2001.
[16] P. Salembier, A. Oliveras, and L. Garrido, "Anti-Extensive Connected Operators for Image and Sequence Processing," *IEEE Trans. Image Processing,* vol. 7, pp. 555-570, 1998.
[17] P. Salembier and J. Serra, "Flat Zones Filtering, Connected Operators, and Filters by Reconstruction," *IEEE Trans. Image Processing,* vol. 4, pp. 1153-1160, 1995.
[18] R.E. Tarjan, "Efficiency of a Good but Not Linear Set Union Algorithm," *J. ACM,* vol. 22, pp. 215-225, 1975.
[19] E.R. Urbach and M.H.F. Wilkinson, "Shape Distributions and Decomposition of Gray Scale Images," IWI-report 2000-9-15, Inst. Math. and Computing Science, Univ. of Groningen, 2001.
[20] L. Vincent, "Grayscale Area Openings and Closings, Their Efficient Implementation and Applications," *Proc. EURASIP Workshop Math. Morphology and Its Application to Signal Processing,* pp. 22-27, 1993.
[21] L. Vincent, "Morphological Area Openings and Closings for Gray-Scale Images," *Shape in Picture: Mathematical Description of Shape in Grey-Level Images,* Y.-L. O, A. Toet, D. Foster, H.J.A.M. Heijmans, and P. Meer, eds., pp. 197-208, 1993.
[22] L. Vincent, "Morphological Grayscale Reconstruction in Image Analysis: Application and Efficient Algorithm," *IEEE Trans. Image Processing,* vol. 2, pp. 176-201, 1993.
[23] L. Vincent, "Granulometries and Opening Trees," *Fundamenta Informaticae,* vol. 41, pp. 57-90, 2000.
[24] M.H.F. Wilkinson and J.B.T.M. Roerdink, "Fast Morphological Attribute Operations Using Tarjan's Union-Find Algorithm," *Proc. Int'l Symp. Memory Management 2000,* pp. 311-320, June 2000.
[25] M.H.F. Wilkinson and M.A. Westenberg, "Shape Preserving Filament Enhancement Filtering," *Medical Image Computing and Computer-Assisted Intervention,* W.J. Niessen and M.A. Viergever, eds., pp. 770-777, 2001.

Arnold Meijster obtained the MSc degree in computing science from the Institute of Mathematics and Computing Science, University of Groningen, The Netherlands. He is now working at the Centre for High Performance Computing and Visualization of the University of Groningen. Besides this, he is currently finalizing the PhD degree on parallel implementation of various operators in mathematical morphology.

Michael H.F. Wilkinson obtained the MSc degree in astronomy from the Kapteyn Laboratory, University of Groningen (RuG) in 1993, after which he worked on image analysis of intestinal bacteria at the Department of Medical Microbiology, RuG. This work forms the basis of his PhD at the Institute of Mathematics and Computing Science (IWI), RuG, in 1995. He was appointed as researcher at the Centre for High Performance Computing (also RuG) working on simulating the intestinal microbial ecosystem on parallel computers. After this, he worked as a researcher at the IWI on image analysis of diatoms, funded by European Commission MAST-contract MAS3-CT97-0122. He is currently assistant professor at the IWI. He is a member of the IEEE and IEEE Computer Society.