EFFECTIVE COMPONENT TREE COMPUTATION WITH APPLICATION TO PATTERN RECOGNITION IN ASTRONOMICAL IMAGING

C. Berger, Th. Géraud, R. Levillain, N. Widynski

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

A. Baillard, E. Bertin

Institut d'Astrophysique de Paris, UMR 7095
98 bis boulevard Arago, F-75014 Paris
France

ABSTRACT

In this paper a new algorithm to compute the component tree is presented. As compared to the state of the art, this algorithm does not use excessive memory and is able to work efficiently on images whose values are highly quantized or even with images having floating values. We also describe how it can be applied to astronomical data to identify relevant objects.

***Index Terms*—** Morphological operations, Algorithms, Nonlinear filters, Pattern recognition, Astronomy.

1. INTRODUCTION

The component tree of an image is a convenient and versatile representation of the image contents. In this representation a tree node denotes a particular connected component of the image level sets and parenthood between nodes maps the relationship of spatial inclusion between components at different levels. Numerous applications rely on component trees: classification, image filtering, segmentation, registration, and compression; see, e.g., [1] for references. However their most prominent use remains to implement many morphological operators [2], for instance algebraic openings/closings and levellings, for which several algorithms have been designed.

The context of our work is the identification of some particular astronomical objects in sky images, namely stars and galaxies, while dealing with optical effects (halos, diffraction spikes and saturated stars) and other defects (satellite trails and cosmic rays). One of the best applications is masking image defects in an automated way. This step is necessary to allow astronomers to compute reliable statistics. Masking is currently largely done by hand. It can take hours for a large field and will become intractable with the next generation of imaging surveys (several terabytes of images per night). Using component trees to represent images in this context is straightforward: the objects to be identified can be described in terms of attributes. Stars are unresolved and therefore show up as realizations of the local Point-Spread Function (PSF), with a given Full-Width at Half-Maximum.

Astronomical images have two main features. To avoid quantization effects after calibration and averaging, they are encoded with floating point values. With current wide-field instruments, mosaiced images weigh several hundreds of millions pixels. We then observed that the algorithms proposed in the literature to compute component trees were not suitable enough. Either execution time is prohibitive or memory usage is too costly. To address those problems we have designed an effective algorithm to compute component trees that realizes a good compromise between both efficiency at run-time and memory usage. As a consequence, this algorithm is also suitable to more common images.

After introducing the specificities of astronomical data in Section 2 the algorithm we propose is described in Section 3. Section 4 compares this algorithm with the state of the art and Section 5 discuss some applications to object identification in astronomical images. Last we conclude in Section 6.

2. THE CASE OF ASTRONOMICAL DATA

Astronomical images contain floating values with high range dynamics. Most pixels correspond to the (noisy) sky background and thus have very low floating values (darker pixels in Figure 1); brighter pixels are objects: stars, galaxies, and some patterns due to optical effects (halos, diffraction spikes, etc.).

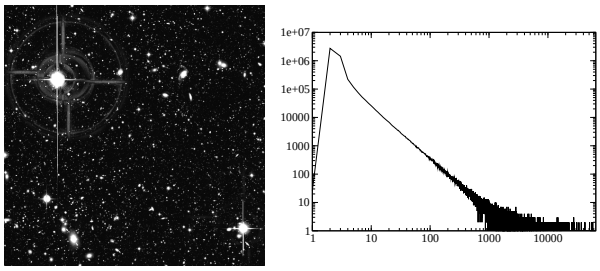

Figure 1: Left: a fragment of 20 million pixels (5% of original image[a]). Right: log-log plot of an image histogram after linear 16-bit quantization.

This work is supported by grant 04-5500 from the French Ministry of Research (ACI "Masses de Données" EFIGI).

[a]Based on observations obtained with MegaPrime/MegaCam, a joint project of CFHT and CEA/DAPNIA, at the Canada-France-Hawaii Telescope (CFHT) which is operated by the National Research Council (NRC) of Canada, the Institut National des Science de l'Univers of the Centre National de la Recherche Scientifique (CNRS) of France, and the University of Hawaii. This work is based in part on data products produced at TERAPIX and the Canadian Astronomy Data Centre as part of the Canada-France-Hawaii Telescope Legacy Survey, a collaborative project of NRC and CNRS.

To understand the distribution of floating values in such images, we have quantized an image on 16 bits w.r.t. a linear transform of values. The resulting histogram is depicted in Figure 1 (right) with a $log - log$ plot. We can observe that most of the pixels are quantized over the range 0 to 255 (the left half part of the diagram) whereas the remaining pixels are quantized over 256 to 65535. The slope that appears on this plot is due to the presence of blur. Precisely the observed image is convolved by a point-spread function so intermediate values correspond to the object's contours. As a consequence, to avoid loosing precision in identifying objects, we have no other choice but to perform an *optimal* image quantization on 16 bits or, better, to handle pixel floating values as-is.

In the case of an optimal quantization on 16 bits we end up with an image having a flat histogram and the relative quantization error is quite low for each quantized value. We can then expect to rather correctly identify objects, whatever their range of values (low, intermediate, or high). Yet for a better accuracy in delineating objects we have to stick to the original floating values.

3. PROPOSED ALGORITHM

3.1. Canonical Tree Computation

The algorithm we propose to compute the component tree is based upon the union-find algorithm [3]. It is widely used to implement connected operators [2]. It is also the cornerstone of a recent algorithm [1] designed to compute the component tree (but with too expensive a memory usage as explained in Section 4).

Let us consider the sample image f given in Figure 2 (top) associated with the 4-connectivity. Let us consider the total ordering relationship $\mathcal{R}$ between pixels of f based on decreasing gray levels and, for pixels having the same level, with the classical video scan order. $\mathcal{R}$ is symbolized by the lexicographical naming of points in Figure 2 (bottom): following $\mathcal{R}$ points are sorted from A to J.

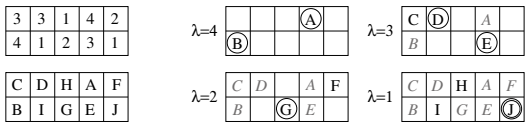

Figure 2: Left: sample image (top) and the ordering $\mathcal{R}$ between pixels (bottom). Right: level sets for different values of λ; canonical points are circled.

The level sets defined by $L_\lambda(f) = \{ p \mid f(p) \geq \lambda \}$ for decreasing values of λ are depicted in the right part of Figure 2. Elements of $L_\lambda(f)$ are printed in slanted gray if they also belong to $L_{\lambda+1}(f)$. The image level sets can be represented by the component tree shown in Figure 3 (left) where node parenthood maps component inclusion. A more compact representation of a component tree is the max-tree shown in Figure 3 (right). In the max-tree, nodes only store the points of $L_\lambda(f) - L_{\lambda+1}(f)$, so data redundancy is avoided.

To construct the max-tree corresponding to a given image, we use the most compact tree representation of images: a rooted tree defined by a parenthood function, named $parent$ and encoded as a 2D image. This function is such as $parent(p)$ is a 2D point. When a node of the max-tree contains several points, we choose its last point (w.r.t. $\mathcal{R}$) as the representative for this node. This canonical element is also called "level root" in the literature. Such elements are depicted within circles in Figures 2 and 4. The canonical element corresponding to the root node of the max-tree is called the "root element". We display this element, J in our example, within a double circle.

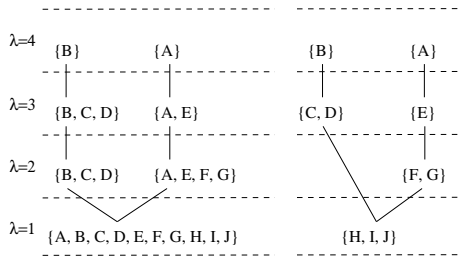

Figure 3: Component tree (left) and max-tree (right).

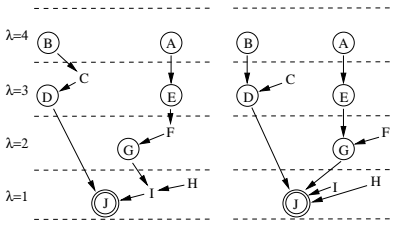

Figure 4: Correct tree (left) and its canonical form (right). The parenthood between points is denoted by arrows; for instance, $parent(C) = D$.

Let Γ denote a component corresponding to a node of the max-tree, p_Γ its canonical element, and p_r the root element. The parent function that we want to construct should verify the following four properties:

1. $parent(p_r) = p_r$
2. $\forall\, p \neq p_r,\ p\ \mathcal{R}\ parent(p)$
   therefore $\forall\, p \neq p_r,\ p\ \mathcal{R}\ p_r$
   and $\forall\, p,\ f(p_r) \leq f(parent(p)) \leq f(p)$
3. p is canonical iff $p = p_r\ \vee\ f(parent(p)) < f(p)$
4. $\forall\, p,\ p \in \Gamma\ \Leftrightarrow\ f(p) = f(p_\Gamma)\ \wedge\ \exists\, i, parent^i(p) = p_\Gamma$
   (where $parent^i$ is the i$^{\text{th}}$ application of $parent$)
   therefore $\forall\, p \in \Gamma,\ p = p_\Gamma\ \vee\ p\ \mathcal{R}\ p_\Gamma$.

The algorithm COMPUTE-TREE given in Figure 5 computes a "correct" $parent$ function, that is, a function that fulfills those properties. Both trees of this figure depict a correct $parent$ function but the one on the right verifies an extra property:

5. $\forall p,\ parent(p)$ is a canonical element.

The algorithm CANONIZE-TREE in Figure 5 transforms any correct $parent$ function so that the last property is verified. The resulting tree has now the simplest form that we can expect. Furthermore we have an isomorphism between images and their canonical representations.

The key feature of the algorithm we propose is to rely on an *un*compressed version of the union-find algorithm to

```
FIND-ROOT(x)
1  if zpar(x) = x then return x
2     else { zpar(x) ← FIND-ROOT(zpar(x)) ; return zpar(x) }

COMPUTE-TREE(f)
1  for each p, zpar(p) ← undef
2  R ← REVERSE-SORT(f)   // maps R into an array
3  for each p ∈ R in direct order
4     parent(p) ← p ; zpar(p) ← p
5     for each n ∈ N(p) such as zpar(n) ≠ undef
6        r ← FIND-ROOT(n)
7        if r ≠ p then { parent(r) ← p ; zpar(r) ← p }
8  DEALLOCATE(zpar)
9  return pair(R, parent)   // a ''correct'' function

CANONIZE-TREE(parent, f)
1  for each p ∈ R in reverse order
2     q ← parent(p)
3     if f(parent(q)) = f(q) then parent(p) ← parent(q)
4  return parent   // a ''canonized'' function
```

Figure 5: Proposed algorithm.

compute the *parent* function. However the function resulting of COMPUTE-TREE, as it is correct, somehow "includes" the max-tree, such as it is observable in the left tree of Figure 4. Getting a simple and compressed representation of the max-tree is postponed to the routine CANONIZE-TREE. For the *parent* computation to be efficient, we need a fast access to temporary root elements (canonical elements of the connected components of the set of already processed points). This fast access is provided thanks to an auxiliary compressed union-find structure, namely *zpar*.

3.2. Attributes Computation and Node Labeling

The result of the algorithm is the array *R* of sorted points (following $\mathcal{R}$) and the *parent* image. As compared to other implementations of component trees, we do not store the childhood relationship. Yet one can compute attributes. To that aim, the mechanism is to push information from children to their parent. Figure 6 displays it with the classical "area" attribute.

```
COMPUTE-AREA(f, R, parent)
1  for each p ∈ R, area(p) ← 1  // initialization
2  for each p ∈ R in direct order
3     area(parent(p)) ← area(parent(p)) + area(p)  // update
```

Figure 6: Sample attribute computation.

At the end each canonical element p_Γ stores the correct attribute value. Our strategy is thus a three-step process: 1. compute the component tree; 2. compute attributes for each node of the tree; 3. label the nodes w.r.t. some criteria computed over attributes. Dissociating these three steps enables an interactive processing: separating the construction of the component tree from the computation of attributes allows the user to select the set of attributes to be computed, in accordance with the form of the expected result, at a very little cost. Likewise, a separate labeling step allows the modification of the parameters used for criteria (e.g., a maximum area when searching for tiny objects) with immediate visualization.

3.3. Handling Floating Values

When values are not quantized but floating values such as in original astronomical images, almost every points is a canonical element. Put differently there are no flat zones so most of the nodes of the max-tree only contain a single point. An interesting property of the algorithm given in the previous section is that it just applies *as-is*. However proceeding to the tree canonization step then becomes useless.

4. RELATED WORK AND COMPARISON

Several approaches have been used to compute the component tree of an image, divided in two main categories: priority queue-based algorithms, building the tree by accumulation, and union-find-based algorithms, generating disjoint set forests.

Jones [4] proposed an algorithm of the first category, as a generalization of Vincent's algorithm [5]. Another fast queue-based method was proposed by Salembier *et al.* [6]. Both methods are penalized by images with a high quantization, because of the cost of the updates (insertion and removal of pixels) in the queue data structure. The union-find approaches are based on Tarjan's disjoint set forest computation [3]. Meijster [7] exposed the first method accelerating the FIND-ROOT procedure using a FIND-LEVEL-ROOT routine (per-level path compression). However, this approach is intractable with highly quantized images such as astronomical pictures, as they contain too many different levels for the compression to be useful. Najman and Couprie [1] also defined an "emergence process" using two disjoint set forests (node and tree) to prevent the creation of degenerated (unbalanced) trees. While having execution time comparable with our proposition, this method requires too much memory on today's computers (about five times the size of the input) to be used with high resolution images.

The following results were obtained on a 3 Ghz Bi-Xeon-based computer with 2×1 MB cache memory and 4 GB RAM running GNU/Linux. The implementations of the algorithms don't make use of parallel capabilities of the processor. The first series of tests was made on 16-bit images, downsampled and equalized from float images (see Figure 7). Both Najman and Couprie's algorithm and our algorithm outperform Salembier *et al.* algorithm, whose curve grows faster than the two other algorithms. Our algorithm obtains the best results in the 16-bit case.

The floating point value case shows new issues that were not yet observed in component tree computations. In particular, the fact that a level is rarely seen twice in a (natural) float image affects Salembier *et al.* algorithm insomuch that its execution time is beyond comparison with the other algorithms on images of floating point values, e.g. about two days for a 5 millions pixels image (these results were not represented on Figure 7). The cost of the updates within the hierarchical queue is presumably too high when it has to store many more levels than in an image with a much lower quantization. Najman and Couprie's algorithm performs better than ours, showing that the ranking technique is relevant on float images.

The structure used in Najman and Couprie's algorithm

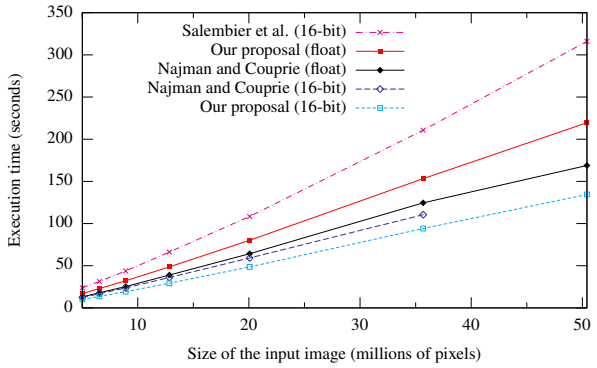

Figure 7: Execution time on 16-bit and float images.

and ours encodes only *points*, so the memory used depends only on the number of points of the input image (and the way these points are represented), not the size of the value carried by each pixel – in the present cases, 2 bytes (16-bit images) or 4 bytes (float images). For this space complexity analysis, we consider that each point is encoded as a 32-bit index (i.e., an offset from the beginning of the image). As for space requirements, Najman and Couprie's algorithm is the one using the most memory. A rough estimation for an image of n points is that it uses n points for the $\mathcal{R}$ relationship, $2n$ points to store the $\mathcal{Q}_{\text{tree}}$ relationship (for the tree itself and the rank structure), $2n$ points for the $\mathcal{Q}_{\text{tree}}$ as well, and n pixels or more to store the actual component tree as the $children$ structure, which ends in 6 times the size of the input in the float case. (We do not take the *lowestNode* array into account, since the algorithm can be adjusted to get rid of it, according to the authors.) Our proposal requires 3 times the size of the input in the float case (for $\mathcal{R}$, the $parent$ and $zpar$ images), i.e. twice less than the previous algorithm. The memory used by Salembier *et al.* algorithm is more difficult to evaluate, since the maximum size of the hierarchical FIFO queue depends on the nature of the data. However, the time complexity of the algorithm images goes hand in hand with a space complexity, since the algorithm ran out of memory on processing a 8.9 million pixels float image.

5. APPLICATION TO ASTRONOMICAL DATA

We only sketch some criteria used to recognize objects in this section. The component tree representation is versatile enough so that objects and defects can be identified. First, to restrict this identification to some parts of the tree, we discard the branches that are not significant; practically, flat regions of the sky background are spotted as not having their isomagnitude at $\frac{f_{\max}}{2}$ twice greater than the one of the surrounding region $\{p|f(p) \in [\frac{f_{\max}}{4}; \frac{f_{\max}}{4}]\}$. For instance in the identification process, stars are recognized as tiny components (i.e. having a small area) that follow the PSF.

As an example, satellite trails are long and thin objects without uniform brightness (unlike "centered" objects). They are identified as large enough components (width (w) + height (h) greater than threshold), so that $\operatorname{var} x + \operatorname{var} y > \frac{(w+h)^2}{14}$, where $\operatorname{var} x$ and $\operatorname{var} y$ are respectively the spatial variance on the component along the X- and Y-axis.

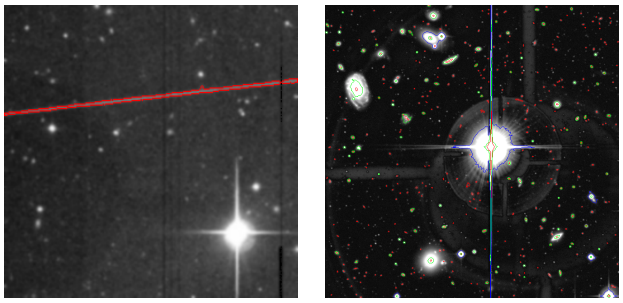

Figure 8: Left: Satellite trail.
Right: Contours of components at $\frac{f_{\max}}{2}$, $\frac{f_{\max}}{4}$ and $\frac{f_{\max}}{10}$.

6. CONCLUSION

In this paper we have presented a new algorithm to compute the component tree. Its advantages are manifold. First it is effective for images with high quantization and with no quantization. It is about as efficient as the fastest known algorithm. Last it saves half of memory usage as compared with its competitor algorithm, which is a crucial point when dealing with large data. We have also presented how to apply this algorithm in order to identify objects of interest in astronomical images.

7. REFERENCES

[1] L. Najman and M. Couprie, "Building the component tree in quasi-linear time," *IEEE Trans. on Image Processing*, vol. 15, no. 11, pp. 3531–3539, Nov. 2006.

[2] Th. Géraud, "Ruminations on Tarjan's union-find algorithm and connected operators," in *Mathematical Morphology: 40 Years On (Proc. of ISMM)*. 2005, Computational Imaging and Vision Series, pp. 105–116, Springer.

[3] R.E. Tarjan, "Efficiency of a good but not linear set union algorithm," *Journal of the ACM*, vol. 22, no. 2, pp. 215–225, 1975.

[4] R. Jones, "Component trees for image filtering and segmentation," in *IEEE Workshop on Nonlinear Signal and Image Processing*, E. Coyle, Ed., Mackinac Island, September 1997.

[5] L. Vincent, "Grayscale area openings and closings: their applications and efficient implementation," in *Internal Symposium on Mathematical Morphology*, 1993, pp. 22–27.

[6] P. Salembier, A. Oliveras, and L. Garrido, "Antiextensive connected operators for image and sequence processing," *IEEE Trans. on Image Processing*, vol. 7, no. 4, pp. 555–570, 1998.

[7] A. Meijster, *Efficient Sequential and Parallel Algorithms for Morphological Image Processing*, Ph.D. thesis, Univ. of Groningen, the Netherlands, March 2004.