Interactive segmentation with incremental watershed cuts

Quentin Lebon[1], Josselin Lefèvre[1,2], Jean Cousty[1], and Benjamin Perret[1]

[1] LIGM, Univ Gustave Eiffel, CNRS, F-77454 Marne-la-Vallée, France
[2] Thermo Fisher Scientific, Bordeaux, France

Abstract. In this article, we propose an incremental method for computing seeded watershed cuts for interactive image segmentation. We propose an algorithm based on the hierarchical image representation called the binary partition tree to compute a seeded watershed cut. We show that this algorithm fits perfectly in an interactive segmentation process by handling user interactions, seed addition or removal, in time linear with respect to the number of affected pixels. Run time comparisons with several state-of-the-art interactive and non-interactive watershed methods show that the proposed method can handle user interactions much faster than previous methods achieving significant speedup from 15 to 90, thus improving the user experience on large images.

Keywords: Interactive segmentation · watershed · binary partition tree · minimum spanning tree

1 Introduction

Image segmentation consists in partitioning an image into meaningful regions. A classical approach to this problem is the watershed (WS) where the image is seen as a topological relief and the regions corresponds to catchment basins: this method constitutes a fundamental stage of many image analysis workflows. The watershed segmentation is still widely used in today's learning era as a pre-processing or a post-processing (see eg [2, 9, 13, 14, 24]), achieving state-of-the-art results. The first approaches for WS on images considered an image as a vertex-weighted graph [3, 23]. Nowadays, state-of-the-art WS methods are defined on edge-weighted graphs [7], allowing to characterize WS cuts as solutions to a global optimization problem related to Minimum Spanning Trees (MST). Further advanced WS are based on hierarchical representations describing how catchments basins are progressively merged into the most significant structures [16, 17]. In this context, the authors of [18] have proposed an algorithm to compute a WS cut from a hierarchical representation called the Binary Partition Tree (BPT) [22].

One of the major drawbacks of WS is over-segmentation as each minimum of the image induces a catchment basin. To overcome this problem, weakly supervised WS segmentation through interactivity is a popular solution. By substituting the minima of the image by user-defined seeds, this procedure makes

it possible on the one hand to avoid over segmentation and on the other hand to introduce semantic information into the output segmentation.

But often, the segmentation has to be corrected by successively refining seeds *i.e.*, adding or removing seeds, until a result close to the desired segmentation is reached. Classical seeded WS algorithms cannot handle such incremental process and each update of the seeds requires to completely recompute the result on the whole image, even if a small number of pixels is affected by the change. This problem can seriously limit the speed of user interactions on large images or 3D volumes. This issue has been addressed in the framework of the Image Foresting Transform (IFT) [11] with the Differential Image Foresting Transform (DIFT) [10] a WS-based and fuzzy-connected segmentation method, whose response time for interactive segmentation is proportional to size of the modified regions of the scene. Interactivity has also been addressed in the context of object segmentation with hierarchical representations, especially for trees based on threshold decomposition *i.e.*, component trees [20] or tree of shapes [5, 19].

In this work, we propose a seeded WS cut algorithm within the framework of BPTs. We show that this algorithm is particularly well suited for interactive segmentation, as it can handle user interactions in time linear with the number of affected pixels. The method is assessed on several images without using any GPU architecture where users were asked to interactively segment an object of interest by adding/removing seeds, we also assessed it on a more extensive dataset where random generated seeds are added. The run-time comparisons with state-of-the-art incremental and non-incremental methods show a significant advantage for the proposed method.

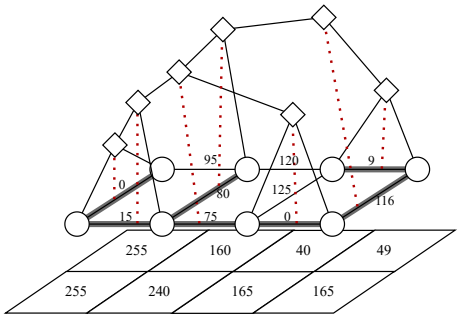

Fig. 1. The three working areas: grayscale image, graph, and BPT. The dashed red lines depict the bijection between non-leaf nodes and edges of the MST (in bold).

2 Watershed cuts

In this section, we briefly review the notion of a WS cut, recall its relation to minimum spanning trees and forests and highlight the important BPT datastructure that is used in the following. WS cuts are deeply related to MSTs [7]. In the semi-supervised case where an edge-weighted graph (representing the image) and a set of marked vertices are given, a WS cut can be obtained as the cut induced by a minimum spanning forest where each tree is rooted in one of the seeds [1]. The BPT is a hierarchical datastructure that allows one to represent a MST and to efficiently browse its edges. The BPT is obtained as a by-product of the efficient Kruskal's MST algorithm. Figure 1 illustrates the BPT of an edge-weighted graph and its relation to the MST of this graph (bold edges). In particular, it can be seen that the leaves of the BPT are mapped to the vertices of the MST and that each non-leaf node is a mapped to an edge of the MST (this mapping is represented by the dashed segments in Figure 1). Intuitively, we can say that every non-leaf node of the BPT represents the addition of an edge to the MST during Kruskal's algorithm, the added edge being used to merge the two connected regions that contain the edges extremities. An efficient WS cut algorithm based on BPT is presented in [8,18] to handle the non-supervised case with no seed provided.

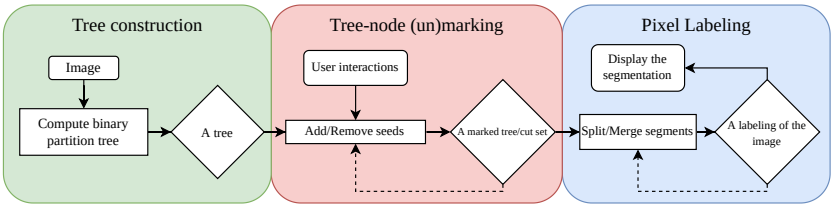

Fig. 2. Overview of our workflow of the interactive incremental watershed (WS) segmentation

3 Semi-supervised watershed cut algorithm with interactions

In this section, we present a novel WS cut algorithm that is able to (i) compute a seeded WS cut from user-provided seeds, and to (ii) efficiently update this WS cut from user's feedback given in the form of successive deletions and additions of seeds. The workflow of the method, presented in Figure 2, comprises three main parts: **1)** Computation of an MST and an associated BPT; **2)** Identification of WS cut edges by browsing the BPT, taking into account the seeds provided by the user. These edges correspond to the edges that must be removed from the

MST to obtain a minimum spanning forest rooted in the given seeds. **3)** Partitioning and labeling of the graph vertices according to the connected components (CC) of the forest resulting from step 2. This labeling is the resulting WS cut segmentation. Step 1 can be performed using the algorithm presented in [18]. Section 3.1 presents an efficient algorithm for step 2, Section 3.2 discusses the labeling involved at step 3, and Section 3.3 shows how to incrementally update the results of the workflow based on user's feedback.

3.1 Tree-node marking

This section is devoted to Algorithm 1 that identifies the WS edges from the BPT of the image for some user provided seeds. The algorithm proceeds seed by seed in an incremental manner. For each seed, taken in any order, one edge of the initial MST must be removed to cut the region of this seed from the regions of the current partition (obtained with the previous seeds). To this end, we search in the BPT for the lowest node that merges a region of the new seed with a region of the already processed seeds. In order to "prevent" this merging and to cut the region of the new seed from the rest of the partition, the MST edge associated to this node is tagged WS and added to the set of edges to be deleted from the MST.

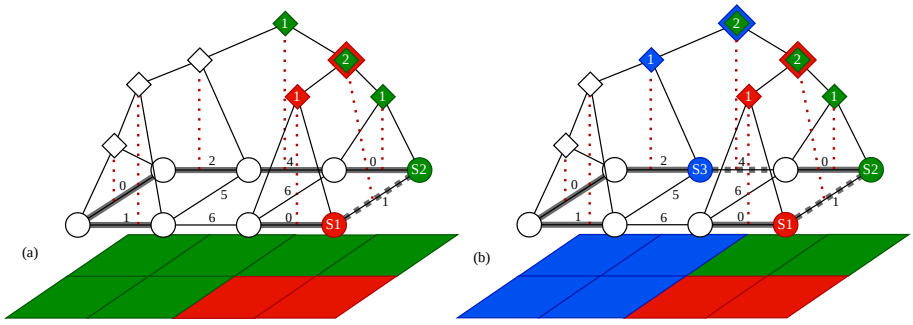

Fig. 3. Integer labels on non-leaf nodes represent the value of `visitCount`. (a) Initialization with S_1 and S_2 as seeds. (b) Update after the addition of S_3 as an additional seed.

To make this idea practicable, we consider an array `visitCount` that marks each BPT node with the number of times that this node was visited during the successive searches. At the beginning, `visitCount` is initialized to 0 for each node. Then, for each leaf node of the tree corresponding to a seed to add, we browse its ancestors and update `visitCount` accordingly. Let `p` be a parent of the considered seeded node, if `visitCount[p]` $= 0$ then this value has to be incremented and the traversal continues by considering the parent of `p`. However,

if `visitCount[p]` $= 1$, then node `p` has already been visited by a previous seed: it must then become a WS node separating 2 different seeds. We thus increment the value of `visitCount[p]` to 2 and the MST edge associated to `p` (denoted by `H.mstEdge[p]`) is added to the edge set WS. Note that the traversal stops when `visitCount[p]` $= 2$ as the separation induced by the addition of the new seed has been found. When a seed is added, only the nodes in the path from this seed to its closest WS node are visited. We can see this on Figure 3(b), where adding seed S_3 results in browsing only three nodes. During a call or a succession of calls to Algorithm 1, each node is visited at most twice, leading to an overall linear-time complexity with respect to the number of vertices.

Algorithm 1: ADD SEED

Data: $\mathcal{H}$: a BPT, `seeds` : a set of seeds and `visitCount`;
Result: `ws` a set of edges to be removed and `visitCount` updated.

```
1 ws ← ∅
2 foreach leaf n of seeds do
3     while n ≠ H.root and visitCount[n] ≠ 2 do
4         n := H.parent[n]
5         visitCount[n] := visitCount[n] + 1
6         if visitCount[n] = 2 then
7             ws := ws ∪ {H.mstEdge[p]}
```

3.2 Pixel labeling

Once the WS nodes set is computed, we return to the image domain to compute the segmentation. First, we compute the minimum spanning forest representing the WS cut: for each added WS node, the corresponding WS edge is removed from the MST. Then we perform a labeling of the CCs of the forest with a simple Breadth First Search (BFS) algorithm.

3.3 Incremental workflow

The workflow presented in the previous sections can be adapted to work in an incremental way, by considering the addition or removal of seeds and the differential update of the resulting WS cut and all intermediary structures.

Firstly, we adapt Algorithm 1 to obtain Algorithm 2 which accounts for seed removal. Such removal induces the merging of two regions and the disappearance of a WS node. The traversal procedure is the same as for adding seeds except that for each parent `p`, `visitCount[p]` is decremented and the parent browsing stops if `visitCount[p]` $= 1$ after decrementation. Indeed, if the value was decremented to 1, the current node is no longer be a WS node.

Regarding the labeling, Algorithm 2 can now restore WS edges which induces the merging of two CCs. That can be efficiently performed by constraining BFS to explore only the smallest CC associated with one extremity of a cut edge. The label of the larger CC is then spread on the smaller one. This can be done by keeping track of the size of each CC resulting in a linear time merging *w.r.t.* the number of vertices of the smallest CC. Note that when a WS edge is removed, we also only need to relabel the components at the two extremities of this edge. The split of a component thus also runs in time linear with the number of pixels in the affected component.

As a result, this incremental workflow enables updating of the segmentation in a time proportional to the number of pixels in the region affected by the seed refinement.

Algorithm 2: REMOVE SEED

Data: $\mathcal{H}$: a BPT, `seeds` : a set of seeds and `visitCount`;
Result: `ws` a set of edges to be added and `visitCount` updated.

```
1 ws ← ∅
2 foreach leaf n of seeds do
3     while n ≠ H.root and visitCount[n] ≠ 1 do
4         n := H.parent[n]
5         visitCount[n] := visitCount[n] − 1
6         if visitCount[n] = 1 then
7             ws := ws ∪ {H.mstEdge[p]}
```

4 Experiments

We assessed the method with two different experiments. The first one confronts different methods with real user interactions while the second one relies on a publicly available dataset with a larger number of images and associated ground-truth segmentations, for which the interactions are simulated by randomly selecting markers from the ground truth.

4.1 Experiment with user generated seeds

For assessments, we confronted a user with a binary segmentation problem on three images from the INRIA Holidays dataset [12] and on one provided by ourselves. The images are all the same size: 2048 by 1536 pixels. To retrieve seeds, we ask the user to segment the images with an interactive tool: he could draw green seeds for the object of interest and red seeds for the background and if a mistake was made, he could remove seeds with an eraser tool. Each

user interaction was recorded in batches of added/removed green/red seeds. The same sets of seeds were used for each tested method.

In this study, we consider two versions of the proposed method: (i) a non incremental version (denoted NIWS) where we first compute the BPT and then, at each user interaction, we completely recompute the WS edges (using only Algorithm 1) and the induced labeling; and (ii) an incremental version (denoted IWS) where at each interaction we update `visitCount` (Algorithms 1 and 2) and the labeling by considering only the added/removed seeds. We also consider three state-of-the-art implementations of seeded WS, namely OpenCV [4, 15] (highly optimized library for image processing), Higra [21] (generic library for hierarchical graph analysis) and another incremental method: the Differential Image Foresting Transform (DIFT) [10]. IWS and NIWS have been tested with a C++ implementation available at `https://github.com/lebonq/incremental_watershed`. We used a C implementation of DIFT available at `https://github.com/tvspina/ift-demo`.

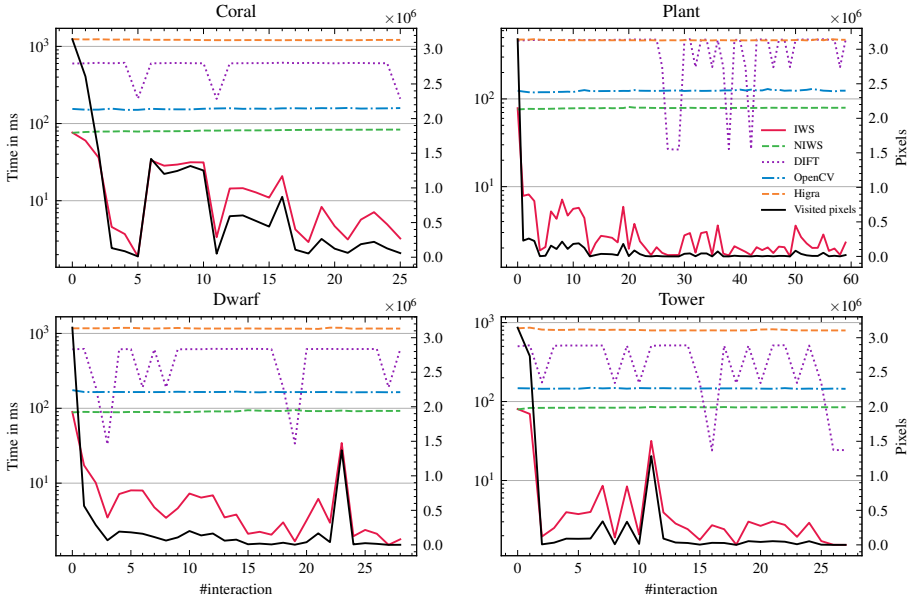

Fig. 4. Computation time of OpenCV (dot-dash blue), DIFT (dotted purple), Higra (dashed orange), NIWS (dashed green) and IWS (plain red) along an interactive segmentation session compared to the number of pixels to be updated (plain black).

The results are presented in Figure 4 and in Table 1. We see that the initialization cost (BPT creation) of both NIWS and IWS version of our method is quickly amortized during the segmentation process: IWS and NIWS have a much lower average execution time than other methods. In addition, we can see that the execution time of IWS is proportional to the number of pixels af-

fected by seed updates as expected from the theoretical study. The upper bound is given by the first interaction, which labels all pixels (the first step is therefore equivalent to NIWS), and spikes in computation time occurs during mid- or end-interactions if the user updates seeds in a large CC, resulting in a significant number of pixel changes. Our results also indicate that there is no significant difference in computation time between adding or removing seeds.

Method	Init	Average	Max	Accumulated
IWS	210.0	9.2	89.3	514.1
NIWS	210.0	82.2	94.8	3164.4
DIFT	14.0	478.4	622.8	15886.3
OpenCV	∅	141.7	174.8	5089.4
Higra	∅	829.1	1239.4	29012.0

Table 1. Computation time (ms) of the methods. The second column corresponds to the initialization time. The third and fourth columns give the average and maximum computation time over all user interactions. The last column gives the total computation time (initialization plus all user interactions).

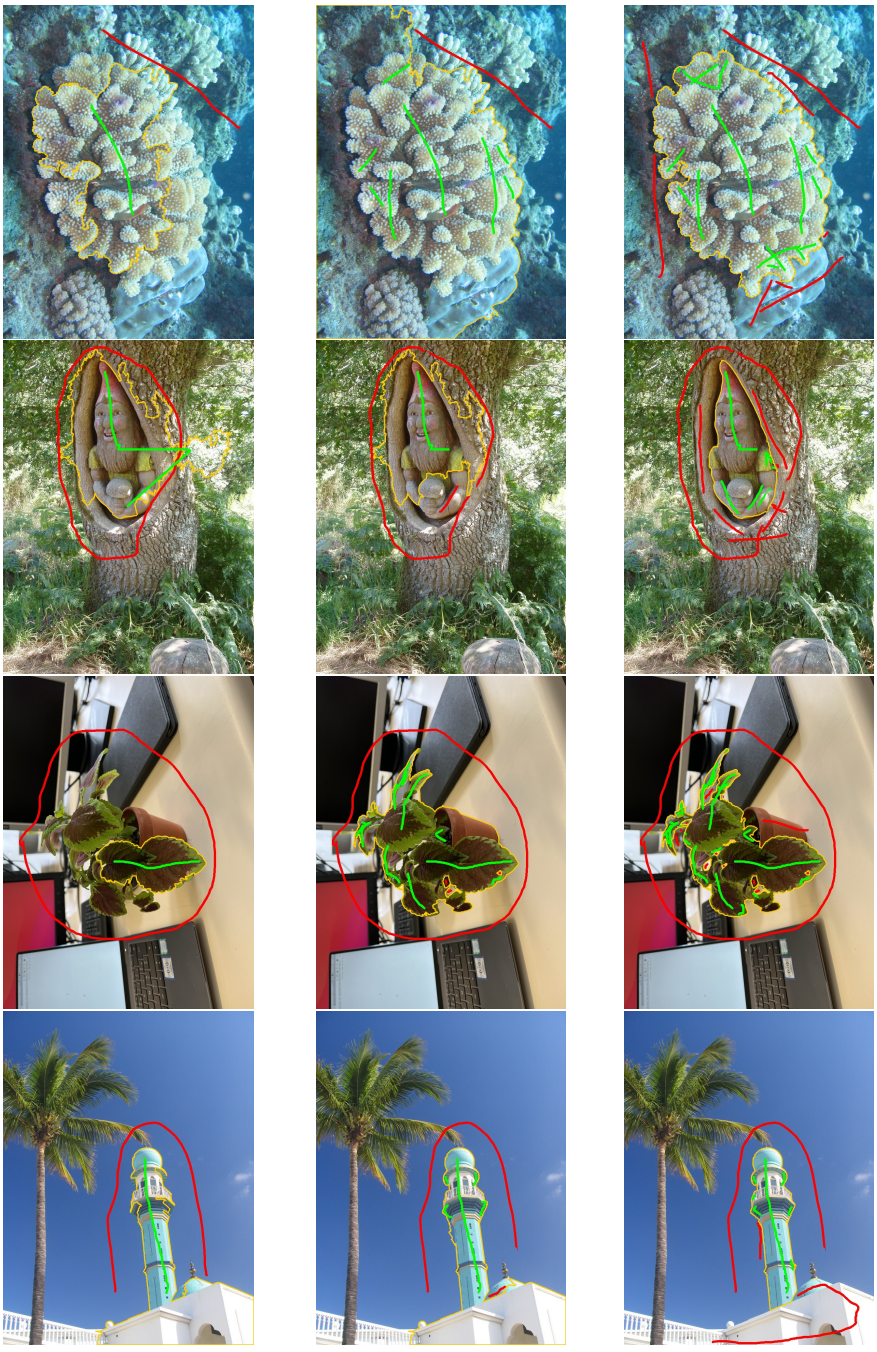

Fig. 5. Three stages in an interactive segmentation session. The red lines represent background seeds, the green lines represent object seeds, and the yellow lines represent the segmentation produced by the current set of seeds.

4.2 Experiment with randomly generated seeds

To assess the methods on a standard dataset with more samples, we chose the BIG dataset [6] which is composed of large natural images (from 2048 by 1600 to 5000 by 3600 pixels). We use all 150 images and pair each image with a series of 70 seeds. Seeds are divided into two balanced classes: object and background. Each seed is a ball of radius 11 centered on a randomly chosen pixel of the object (resp. background) mask eroded by a ball of radius 12 ensuring that the seed lies in the object (resp. background).

In this experiment, we consider the same methodology as in Section 4.1. During the iterative process, seeds are alternatively picked within the object and within the background. In this experiment, only addition of seeds is considered, never removal.

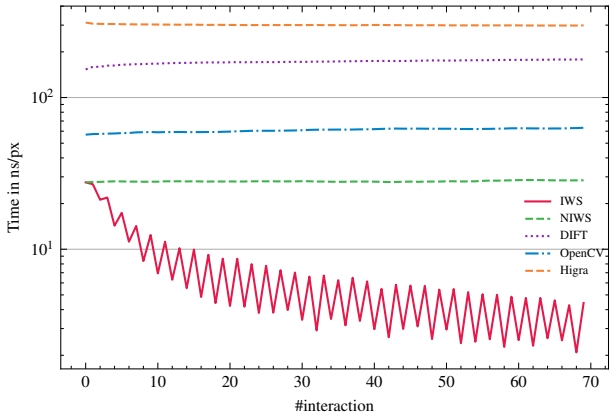

Fig. 6. Computation time per pixel of OpenCV (dot-dash blue), DIFT (dotted purple), NIWS (dashed green), Higra (dashed orange) and IWS (plain red). The time per pixel is the average for all 150 images.

The results are presented in Figure 6 and Table 2. For each interaction, the computation time is the average over all 150 images. With this experiment, the tendency observed in Section 4.1 is confirmed: the execution times of other methods are constant over the iterations whereas our incremental method shows decreasing execution times over the iterations, as the number of pixels affected by the interaction decreases. Furthermore, we observe that our proposed incremental methods significantly improves the answer-time to user interactions compared to all other methods. We can see that the curve for the IWS has a "sawthooth" look. It can be easily explained as we alternately add seeds on the objects and on the background, and the size of the objects is often much smaller than that of the background. Up spikes correspond to seeds placed on the back-

ground, while down spikes correspond to seeds placed on the objects. This is due to the algorithm's dependence on the size of the regions to be updated.

Method	Init	Average	Max	Accumulated
IWS	88.1	6.8	27.6	563.6
NIWS	88.1	28.0	28.6	2051.3
DIFT	3.8	172.0	176.3	12044.9
OpenCV	∅	60.9	61.2	4262.0
Higra	∅	300.7	311.9	21045.3

Table 2. Computation times expressed in nanoseconds per pixels. The second column gives the initialization time of each method. The third and fourth columns give the average and maximum computation time over all interactions. The last column gives the total computation time, considering the sum over the 70 interactions and the initialization time.

5 Conclusion

We proposed an interactive seeded WS segmentation method that is compliant with an incremental process exploiting the causality within the interactive sessions to achieve remarkable performances, significantly improving responsiveness. In future works, we plan to test it on larger images or 3D volumes and to better optimize the splitting algorithm by inferring region size to ensure that the region to label is the smallest.

References

1. Allène, C., Audibert, J.Y., Couprie, M., Keriven, R.: Some links between extremum spanning forests, watersheds and min-cuts. Image and Vision Computing **28**(10), 1460–1471 (2010)
2. Arbeláez, P., Maire, M., Fowlkes, C., Malik, J.: Contour Detection and Hierarchical Image Segmentation. IEEE Transactions on Pattern Analysis and Machine Intelligence **33**(5), 898–916 (May 2011). https://doi.org/10.1109/TPAMI.2010.161, conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence
3. Beucher, S., Meyer, F.: The morphological approach to segmentation: The watershed transformation. Mathematical Morphology in Image Processing **Vol. 34**, 433–481 (01 1993). https://doi.org/10.1201/9781482277234-12
4. Bradski, G.: The OpenCV Library. Dr. Dobb's Journal of Software Tools (2000)
5. Carlinet, E., Geraud, T.: Morphological object picking based on the color tree of shapes. In: 2015 IPTA. pp. 125–130. IEEE, Orleans, France (Nov 2015). https://doi.org/10.1109/IPTA.2015.7367111
6. Cheng, H.K., Chung, J., Tai, Y.W., Tang, C.K.: CascadePSP: Toward class-agnostic and very high-resolution segmentation via global and local refinement. In: CVPR (2020)
7. Cousty, J., Bertrand, G., Najman, L., Couprie, M.: Watershed Cuts: Minimum Spanning Forests and the Drop of Water Principle. IEEE TPAMI **31**(8), 1362–1374 (Aug 2009)
8. Cousty, J., Najman, L., Perret, B.: Constructive links between some morphological hierarchies on edge-weighted graphs. In: ISMM. pp. 86–97 (2013)
9. Eschweiler, D., Spina, T.V., Choudhury, R.C., Meyerowitz, E., Cunha, A., Stegmaier, J.: CNN-Based Preprocessing to Optimize Watershed-Based Cell Segmentation in 3D Confocal Microscopy Images. pp. 223–227. IEEE, Piscataway, NJ (Apr 2019), conference Name: 2019 IEEE 16th International Symposium on Biomedical Imaging (ISBI 2019)
10. Falcao, A., Bergo, F.: Interactive Volume Segmentation With Differential Image Foresting Transforms. IEEE Transactions on Medical Imaging **23**(9), 1100–1108 (Sep 2004). https://doi.org/10.1109/TMI.2004.829335
11. Falcão, A., Stolfi, J., de Alencar Lotufo, R.: The image foresting transform: theory, algorithms, and applications. IEEE TPAMI **26**(1), 19–29 (2004). https://doi.org/10.1109/TPAMI.2004.1261076
12. Jegou, H., Douze, M., Schmid, C.: Hamming embedding and weak geometric consistency for large scale image search. In: Forsyth, D., Torr, P., Zisserman, A. (eds.) Computer Vision – ECCV 2008. pp. 304–317. Springer Berlin Heidelberg, Berlin, Heidelberg (2008)
13. Lux, F., Matula, P.: Dic image segmentation of dense cell populations by combining deep learning and watershed. In: 16th IEEE ISBI. pp. 236–239 (2019), `https://doi.org/10.1109/ISBI.2019.8759594`
14. Machairas, V., Faessel, M., Cardenas, D., Chabardes, T., Walter, T., Decencière, E.: Waterpixels. IEEE TIP **24**, 3707–3716 (Nov 2015). https://doi.org/10.1109/TIP.2015.2451011
15. Meyer, F.: Color image segmentation. In: 1992 International Conference on Image Processing and its Applications. pp. 303–306 (1992)
16. Meyer, F.: The watershed concept and its use in segmentation : a brief history (Feb 2012), `http://arxiv.org/abs/1202.0216`, arXiv:1202.0216 [cs]

17. Najman, L., Schmitt, M.: Geodesic saliency of watershed contours and hierarchical segmentation. IEEE TPAMI **18**(12), 1163–1173 (1996). https://doi.org/10.1109/34.546254
18. Najman, L., Cousty, J., Perret, B.: Playing with Kruskal: algorithms for morphological trees in edge-weighted graphs. In: ISMM. pp. 135–146 (2013)
19. Ngoc, M.Ô.V., Carlinet, E., Fabrizio, J., Géraud, T.: The dahu graph-cut for interactive segmentation on 2d/3d images. Pattern Recognition **136**, 109–207 (2023)
20. Passat, N., Naegel, B., Rousseau, F., Koob, M., Dietemann, J.L.: Interactive segmentation based on component-trees. Pattern Recognition **44**(10), 2539–2554 (2011). https://doi.org/10.1016/j.patcog.2011.03.025, semi-Supervised Learning for Visual Content Analysis and Understanding
21. Perret, B., Chierchia, G., Cousty, J., F. Guimarães, S., Kenmochi, Y., Najman, L.: Higra: Hierarchical Graph Analysis. SoftwareX **10**, 100335 (Jul 2019). https://doi.org/10.1016/j.softx.2019.100335
22. Salembier, P., Garrido, L.: Binary partition tree as an efficient representation for image processing, segmentation, and information retrieval. TIP **9**(4), 561–576 (2000)
23. Vincent, L., Soille, P.: Watersheds in digital spaces: an efficient algorithm based on immersion simulations. IEEE TPAMI **13**(6), 583–598 (1991). https://doi.org/10.1109/34.87344
24. Wolf, S., Schott, L., Köthe, U., Hamprecht, F.: Learned Watershed: End-to-End Learning of Seeded Segmentation (Sep 2017). https://doi.org/10.48550/arXiv.1704.02249, arXiv:1704.02249 [cs]