Incremental Watershed Cuts: Interactive Segmentation Algorithm with Parallel Strategy

Quentin Lebon, Josselin Lefèvre, Jean Cousty, Benjamin Perret

Pattern Recognition Letters
journal homepage: www.elsevier.com

Incremental Watershed Cuts: Interactive Segmentation Algorithm with Parallel Strategy

Quentin Lebon[a], Josselin Lefèvre[a,b], Jean Cousty[a], Benjamin Perret[a]

[a]*LIGM, Univ Gustave Eiffel, CNRS, ESIEE Paris, F-77454, Marne-la-Vallée, France*
[b]*Thermo Fisher Scientific, Bordeaux, France*

ABSTRACT

In this article, we design 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. Additionally, we leverage properties of minimum spanning forests to introduce a parallel method for labeling connected component. We show that those algorithms fits perfectly in an interactive segmentation process by handling user interactions, seed addition or removal, in linear time 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 with a significant speedup ranging from 15 to 90 on both 2D and 3D images, thus improving the user experience on large images.

1. Introduction

In image segmentation, the aim is to divide an image into significant regions. The watershed (WS) is a classical approach to image segmentation where an image is seen as a topological relief which is partitioned into catchment basins associated to the minima of the relief. The WS is an important step of many image analysis pipelines and is still widely used in conjunction with modern machine learning methods such as deep-learning, usually as a pre-processing or a post-processing (see e.g. Machairas et al. (2015); Arbeláez et al. (2011); Eschweiler et al. (2019); Wolf et al. (2017); Lux & Matula (2019)). The WS was first defined on images represented as vertex-weighted graphs Vincent & Soille (1991); Beucher & Meyer (1993). It has then been extended to edge-weighted graphs Cousty et al. (2009) through the notion of WS cut, which is a solution to a global combinatorial optimization problem linked to the Minimum Spanning Tree (MST) problem. WS can also be generalized to hierarchical watersheds, i.e. a hierarchical image representation describing how WS regions progressively merge into the large scale components of the image Najman & Schmitt (1996); Meyer (2012). In particular, the authors of Cousty et al. (2013a); Najman et al. (2013) have shown that WS cuts and WS hierarchies can be computed from a prototypical hierarchical representation called the Binary Partition Tree (BPT).

WS is often considered as a low level-segmentation method because it is highly subject to over-segmentation: as each minimum of the image induces a catchment basin, WS segmentations usually contain a lot of small regions. This problem can be solved by a weakly supervised variation of the WS method called seeded WS. In this case, the minima of the image are replaced by seeds, which can be automatically generated or produced by a user: the number of regions in the result is then equal to the number of seeds.

When seeds are generated by a user, an interactive interface is generally used, allowing the user to modify, add or delete seeds until a satisfactory result is obtained. Traditional seeded WS algorithms don't handle this incremental process, and the WS segmentation must be completely recomputed after each user interaction, even if it only affects a few pixels. While this may be acceptable for small images, it can seriously degrade the user experience when processing large images and especially 3D images such as RMI or FIB-SEM images. In this context, the authors of Falcão et al. (2004) have proposed the Differential Image Foresting Transform (DIFT) for incrementally updating minimum spanning forests Falcão et al. (2004), which can correspond to WS segmentation, when seeds are modified in the

e-mail: quentin.lebon@esiee.fr (Quentin Lebon), josselin.lefevre@esiee.fr (Josselin Lefèvre), jean.cousty@esiee.fr (Jean Cousty), benjamin.perret@esiee.fr (Benjamin Perret)

context of interactive 3D images segmentation.

Interactive segmentation has also been studied in the context of hierarchical representations, such as component trees Passat et al. (2011); Carlinet & Geraud (2015); Ngoc et al. (2023) or segmentation hierarchies Salembier & Garrido (2000). In this case, the hierarchical representation of the image is computed once and, after each interaction, the hierarchy is reprocessed to obtain the segmentation.

In this work, we propose an interactive seeded WS cut algorithm within the framework of BPTs. First, we propose an algorithm to compute a seeded WS segmentation using the BPT associated to the input image. This procedure relies on two steps: 1) seeds propagation in the BPT and 2) connected components labeling. Second, we show how the seed propagation and the connected component labeling can be updated incrementally when the seeds are modified. Third, we propose a parallel algorithm to perform the connected component relabeling. The proposed methods are evaluated on several 2D and 3D images with real and simulated user interactions, showing how each contribution helps in reducing processing time and thus improving the user experience. Moreover, we perform run-time comparisons with state-of-the-art incremental and non-incremental watershed segmentation methods, showing a significant advantage for the proposed approach. A c++ implementation of the proposed methods is available at `https://github.com/lebonq/incremental_watershed`.

This work is an extension of a previous conference paper Lebon et al. (2023). Compared to this preliminary version, we provide: (i) a parallel algorithm for minimum spanning forest labeling, (ii) a more extensive experimental evaluation including 3D images, (iii) a comparison with state-of-the-art methods for images segmentation.

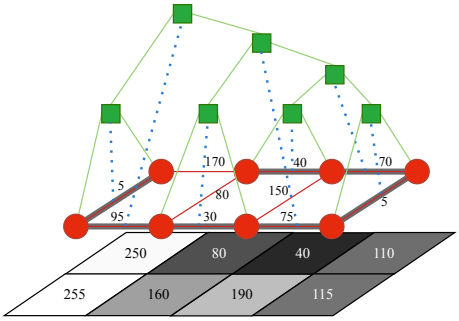

Fig. 1. The three working areas: grayscale image, graph (in red), and BPT (in green). The dotted blue lines depict the bijection between non-leaf nodes and edges of the MST (in bold).

2. Watershed cuts

This section reviews the notion of a WS cut, emphasizing its relation to minimum spanning trees and forests and the associated BPT datastructure that is used in the next section to propose a novel WS cut algorithm. WS cuts are deeply linked to MSTs Cousty et al. (2009): in the semi-supervised case, given an edge-weighted graph (*e.g.* representing the image) and a set of seed vertices (*e.g.*, user-provided scribbles such that each scribble is superimposed to an object of interest to be segmented), a WS cut can be obtained as the cut induced by a minimum spanning forest such that each tree in the forest is rooted in one of the seeds Allène et al. (2010). The BPT is a tree-based hierarchical datastructure used to represent an MST. It allows one to efficiently browse MST's edges. The BPT can be seen as a by-product of the efficient Kruskal's MST algorithm. Figure 1 shows the BPT (in green) of an edge-weighted graph (in red) and its mapping (dotted blue) to the MST of this graph (bold edges). In particular, the leaves of the BPT are associated to the vertices of the MST and each non-leaf node is a associated 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 where the added edge is used to merge the two connected regions that contain the edge extremities. An efficient WS cut algorithm based on BPT is presented in Cousty et al. (2013b); Najman et al. (2013) to handle the unsupervised case with no seed provided.

3. Semi-supervised watershed cut algorithm with interactions

This section introduces a new efficient algorithm to compute a WS cut from user-provided seeds, and to update its result based on user's feedbacks given in the form of successive deletions and additions of seeds.

The overall workflow is presented in Figure 2. It comprises three main parts: **1)** Computation of an MST and of an associated BPT; **2)** Selection of the WS cut edges associated to the seeds provided by the user. These edges, which are found by browsing the BPT, correspond to the edges that must be deleted from the MST to obtain a minimum spanning forest rooted in the provided seeds. **3)** Partitioning and labeling of the graph vertices according to the connected components (CC) of the forest obtained at step 2. This labeling is the resulting WS cut segmentation.

Step 1 is performed using the algorithm presented by Najman et al. (2013). An efficient algorithm for step 2 is given in Section 3.1, Section 3.2 discusses the labeling involved at step 3, and, finally, Section 3.3 shows that the proposed algorithm can be used to incrementally update the results of the workflow based on user's feedback.

3.1. Tree-node marking

In this section, we present Algorithm 1 that returns the WS cut edges from an MST given in the form of a BPT datastructure and from a set of seed vertices. The algorithm works incrementally, seed by seed. For each seed, taken in an arbitrary 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 previously processed seeds). To find this edge, we search in the BPT for the lowest node that is both an ancestor of the new seed and an ancestor of an already processed seed. This node corresponds to the lowest edge of the MST that merges a region containing the new seed with a region containing an already processed seed. In order to "prevent" this merging and to cut the region of the new seed from the rest of

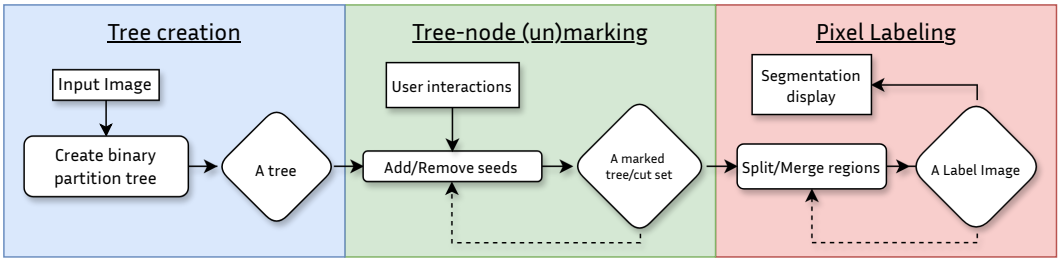

Fig. 2. Overview of our workflow of the interactive incremental watershed (WS) segmentation.

the partition, the MST edge associated to this node is tagged as a "WS edge" and added to the set of edges to be deleted from the MST.

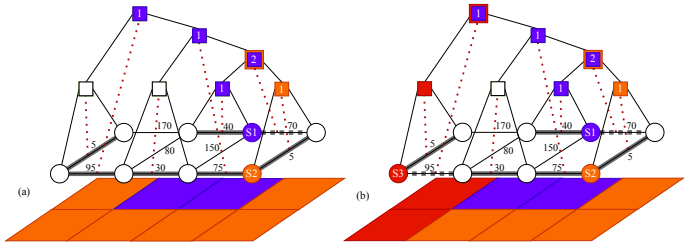

Fig. 3. Integer labels on non-leaf nodes represent the value of `visitCount`. (a) Initialization with S_1 and S_2 as seeds. (b) Update after the addition of S_3 as an additional seed.

In order to implement this scheme, we consider an array `visitCount` which counts for every BPT node the number of times that this node has been visited during the successive searches. Initially, `visitCount` is set to 0 for every node. Then, for each leaf node of the tree corresponding to a seed, we visit its ancestors by browsing the tree from the leaf to the root, and we update `visitCount` accordingly. Let `p` be an ancestor of the considered seed node found during the traversal. If `visitCount[p]` $= 0$ then it must 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 we add the MST edge associated to `p` (denoted by `H.mstEdge[p]`) to the edge set WS (line 7 of Algorithm 1). It can be seen 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.

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 corresponding to the desired WS cut.

Algorithm 1: ADD SEEDS

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 $ws \leftarrow \emptyset$
2 **foreach** *leaf n of* `seeds` **do**
3 **while** $n \neq \mathcal{H}.root$ ***and*** $visitCount[n] \neq 2$ **do**
4 $n := \mathcal{H}.parent[n]$
5 $visitCount[n] := visitCount[n] + 1$
6 **if** $visitCount[n] = 2$ **then**
7 $ws := ws \cup \{\mathcal{H}.mstEdge[p]\}$

To this end, we remove from the MST each WS edge returned by Algorithm 1. Then, we perform a labeling of the CCs of the forest with a Breadth First Search (BFS) algorithm.

3.3. Incremental workflow

The workflow presented in the previous sections is suited to work in an incremental way, where one considers successive addition or removal of seeds and the differential update of the resulting WS cut and intermediary structures. Let us detail the two cases corresponding to the addition and removal of a seed.

Seed addition: When new seeds are added, new WS edges must be produced (one for each new seed). These new watershed edges can be simply obtained, in a differential way, by a call to Algorithm 1, provided that the array `visitCount` has been memorized after the previous call and that the input data only contains the new seeds. Then, we remove the returned new WS cut edges from the current MSF and the connected components split by the removal of these edges are relabeled using breadth first search started at their extremity vertices. It can be seen that the split of a component runs in time linear with respect to the the number of pixels in the split component.

Seed removal: When seeds are removed, WS edges must be removed (one for each seed). We adapt Algorithm 1 to obtain Algorithm 2 which accounts for seed removal in a differential way. Such removal induces the merging of two regions and the disappearance of a WS edge. 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 a WS node and the associated edge must be removed from the

set of WS cut edges. Regarding the associated minimum spanning forest and labeling, Algorithm 2 returns a set of WS cut edges that must be restored within the minimum spanning forest, inducing the merging of a CC for each of them, . This is efficiently performed by constraining BFS to search and relabel only the smallest CC associated with one extremity of the edge. In this process 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.

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

Algorithm 2: REMOVE SEEDS

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 $ws \leftarrow \emptyset$
2 **foreach** *leaf n of* `seeds` **do**
3 **while** $n \neq \mathcal{H}.root$ ***and*** $visitCount[n] \neq 1$ **do**
4 $n := \mathcal{H}.parent[n]$
5 $visitCount[n] := visitCount[n] - 1$
6 **if** $visitCount[n] = 1$ **then**
7 $ws := ws \cup \{\mathcal{H}.mstEdge[p]\}$

4. Parallelization

As shown in the experimental Section 5.3, most of the time spent to update a watershed cut segmentation in this differential framework is dedicated to CCs labeling (around 99.4% for 3D images). Therefore, to further improve the quality and speed of the interactions when dealing with large images, we investigate a parallel algorithm to reduce the time of this labeling by taking advantage of multicore CPUs.

The proposed method, described in Algorithms 3 and 4, is an improved version of the strategy proposed by Youkana et al. (2017) which is able to take into account that the considered CCs are trees which do not contain any cycle and that we can propagate the labels of the initial root vertices instead of the distances to the roots while performing the breadth-first search. The idea is to traverse iteratively the successive level sets found from the root vertices, each level set being traversed in parallel. Before this parallel traversal (Algorithm 3, Line 6), the current level set is partitioned (Line 5) in order to be distributed over the available processors in a balanced way. Since connected components are trees, the same vertex cannot be discovered from several distinct neighbors, thus from several distinct processors. Therefore, to check if a vertex is already labeled, we simply use the regular array `label` without any synchronization.

Moreover, we also introduce two parameters in the method to control the degrees of parallelization and synchronization. The first one, denoted by `MIN_BREADTH`, allows the algorithm to dynamically switch from sequential to parallel search when the current level set (*i.e.* the propagation front) is large enough. The second parameter, denoted by `PAR_DEPTH` in Algorithm 3, allows us to control the number of parallel propagation steps performed between two synchronizations of the processors (union step performed at line 10), knowing that each synchronization between processors is time consuming, but that completely removing those synchronizations could lead to an unfair distribution of the data and an unbalanced workload between processors, resulting in a poor parallelization rate.

Algorithm 3: PARALLEL BREADTH-FIRST SEARCH

Data: $\mathcal{F}$: a forest, *roots*: a set of root vertices to search from, each of which being in a distinct tree of $\mathcal{F}$, p: the number of processor;
Result: `label`: the updated segmentation.

1 $E \leftarrow roots$
2 label[r]:= new unique label
3 **while** $E \neq \emptyset$ **do**
4 **if** $|E| >$ `MIN_BREADTH` **then**
5 $(E_1, \cdots, E_p) := Partition(E, p)$
6 **foreach** *processor i in* $\{1, \cdots, p\}$ **do in parallel**
7 S_i:= SuccessorLabeling($\mathcal{F}, E_i$)
8 **foreach** ℓ *in* $\{1, \ldots,$`PAR_DEPTH`$\}$ **do**
9 S_i:= SuccessorLabeling($\mathcal{F}, S_i$)
10 $E \leftarrow \bigcup_{i=1}^{i=p} \{S_i\}$
11 **else** E:= SuccessorLabeling($\mathcal{F}, E$)

Algorithm 4: SUCCESSORLABELING

Data: $\mathcal{F}$: a forest, E: a set of vertices to explore;
Result: S: the set of explored vertices

1 $S \leftarrow \emptyset$
2 **foreach** *vertex v in E* **do**
3 **foreach** *vertex w adjacent to v in F* **do**
4 **if** *label[w]* $\neq$ *label[v]* **then**
5 $S := S \cup w$
6 $label[w] := label[v]$

5. Experiments

We evaluate the method with three different experiments. The first one compares different methods with real user interactions. The second one relies on a publicly available dataset with a larger number of images and associated ground truth segmentation, for which the interactions are simulated by randomly selecting markers from the ground truth. The third is based on a 3D images dataset of CT scans. The parallel approach was only evaluated on 3D data. Other datasets were too small to demonstrate a speedup.

5.1. Experiment with user generated seeds

In this first experiment, we asked users to segment four images of size 2048×1536 pixels: three images come from the INRIA Holidays dataset Jegou et al. (2008) and one image is provided by ourselves. The users had to segment the images with a graphical interface to interactively edit seeds: they could draw green seeds for the object of interest and red seeds for

Table 1. Computation time (in milliseconds) of the different methods with user generated seeds. The second column indicates the initialization time, while the third and fourth columns show the average and maximum computation time for all user interactions. The last column represents the total computation time, which includes both the initialization and the sum of all user interactions.

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

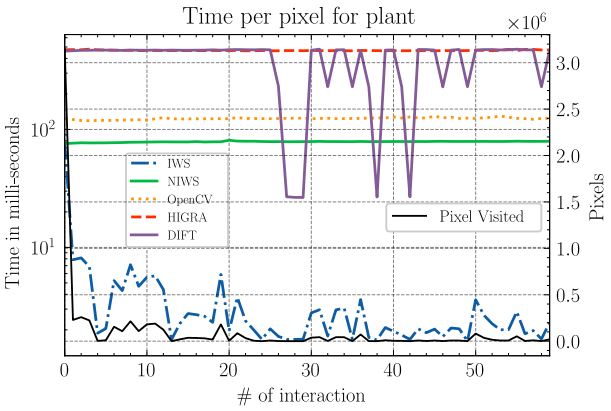

Fig. 4. Computation time of OpenCV (dotted orange), DIFT (plain purple), Higra (dashed red), NIWS (plain green) and IWS (dot-dash blue) during an interactive session (user generated seeds). The plain black curves show the number of pixels updated at each user interaction (right axis).

the background and if a mistake was made, they could remove seeds with an eraser tool. Each user interaction was recorded in batches of added/removed green/red seeds.

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 Meyer (1992); Bradski (2000) (highly optimized library for image processing), Higra Perret et al. (2019) (generic library for hierarchical graph analysis) and another incremental method: the Differential Image Foresting Transform (DIFT) Falcao & Bergo (2004) (we used the authors' implementation available at `https://github.com/tvspina/ift-demo`). The same sets of seeds were used for each tested method. The reported execution times are obtained using an Intel I7 13700H with 14 cores and 20 threads at 5.0GHz running GNU Linux.

The execution times 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

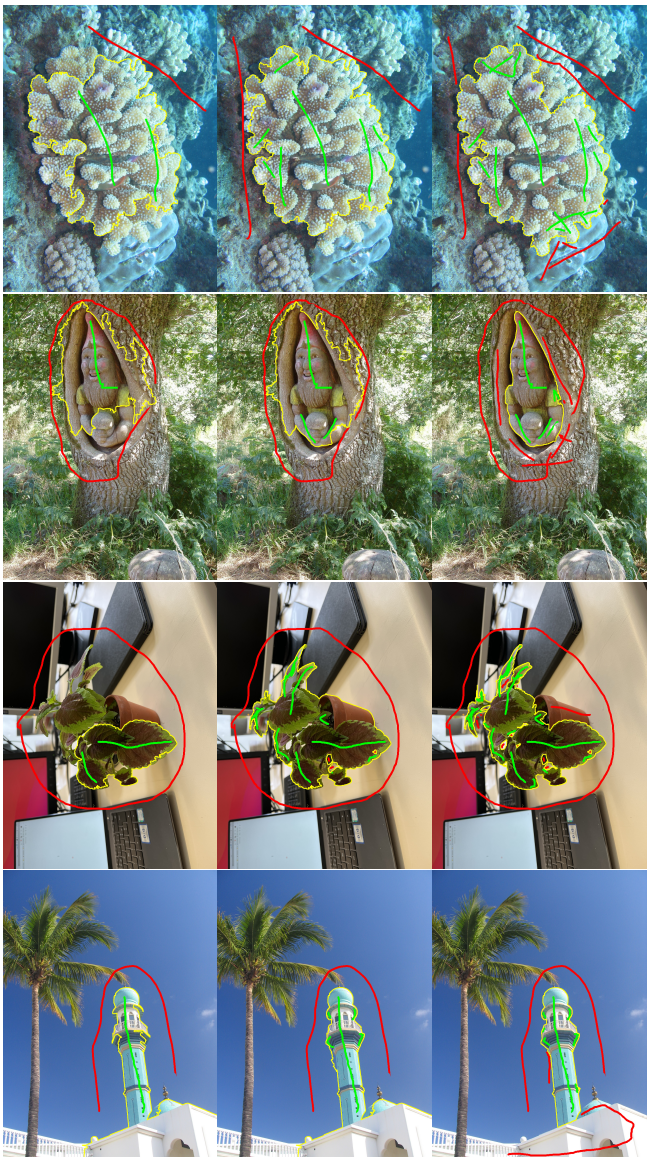

Fig. 5. Evolution of the segmentations thought user interactions. First, second and third show respectively the segmentation (yellow lines) after 2, half and all the user interactions. Red and green lines represent respectively the background and object seeds.

that the execution time of IWS is proportional to the number of pixels affected 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.

5.2. Experiment with randomly generated seeds

To assess the methods on a standard dataset with more samples, we chose the BIG dataset Cheng et al. (2020) 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

Table 2. Computation times in nanoseconds per pixel on the BIG dataset with simulated interactions. The second column displays the initialization time for each method. The third and fourth columns show the average and maximum computation time for all interactions. The last column displays the total computation time, which includes the sum of the 70 interactions and the initialization time.

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

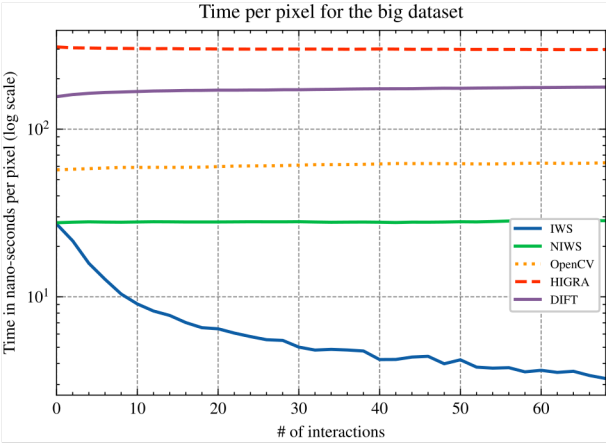

Fig. 6. Computation time per pixel on the BIG dataset with simulated interactions of OpenCV (dotted orange), DIFT (plain purple), NIWS (plain green), Higra (dashed red) and IWS (plain blue). The time per pixel is the average for all 150 images. In order to avoid noisy visualizatin, the curve is smoothed by averaging values in a window of size 2.

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 5.1. 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.

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 5.1 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 response-time to user interactions compared to all other methods.

5.3. Experiment on 3D images

To assess the method on larger data, we considered the liver segmentation dataset Soler et al. (2010), which consists of 3D images acquired by CT scan. Out of them, we selected five 3D images of sizes from $512 \times 512 \times 124$ to $512 \times 512 \times 260$ voxels. We use the same methodology as in Section 5.2, using the liver masks as ground-truths. We generated 60 sets of seeds, each set containing 30 randomly positioned seeds of 100 pixels. Note that since our algorithm works on arbitrary graphs, no modifications are required to process the 3D images. For this experiment we tested the following four methods:

Table 3. Computation times in nanoseconds per pixel on 3D volumes with simulated interactions. The second column displays the initialization time for each method. The third and fourth columns show the average and maximum computation time for all interactions. The last column displays the total computation time, which includes the sum of the 60 interactions and the initialization time.

Method	Init	Average	Max	Accumulated
IWS	85.6	18.4	36.0	1189.6
IWS PAR	92.6	8.0	15.0	572.6
NIWS	85.6	37.7	38.5	2350.2
ITK	∅	162.6	165.1	9754.6

1. ITK: for each iteration we call the 3D watershed implementation available in Insight Toolkit (ITK) state-of-the-art library for medical imaging Beare & Lehmann (2006);
2. NIWS: the watershed-cut algorithm proposed in Section 3.1, thus without using the differential updates;
3. IWS: the watershed-cut algorithm proposed in Section 3.3, thus using differential updates but not using the parallel BFS;
4. IWS PAR: the watershed cut method proposed in Section 3.3 using the parallel BFS presented in Section 4. For this method, we experimentally chose to set the parameters `MIN_BREADTH` and `PAR_DEPTH` to 4000 and 3, respectively and 10 processors/threads were used.

The execution times of these methods are presented in Figure 7 and Table 3. The tendency observed in Section 5.2 and Section 5.1 is confirmed: the execution times for ITK and NIWS remain constant across iterations, whereas for the two differential methods, namely IWS and IWS PAR, the response time to user interactions is significantly reduced and furthermore decreases after few iterations, when the addition of new seeds starts to induce less changes in the segmentation results corresponding to small refinement on the results. On average, over all considered 3D images, ITK (resp. NIWS, IWS, and IWS PAR) requires 8 (resp. 1.8, 0.9, and 0.4) seconds per interaction, leading to a speedup of 20 when we compare the fastest IWS PAR method to the reference ITK. We also note that the use of parallelization in IWS PAR leads to a speedup of 2.3 over the IWS method.

In order to better analyze the proposed methods, we provide the time taken for the different steps of each method. Over all iterations, for NIWS, 95.7% (resp 0.66%, 3.64%) of the time is spend on pixel labeling (resp. node marking/unmarking, and building BPT), for IWS 92.3% (resp 0.5%, 7.2%) of the time is spend on pixel labeling (resp. node marking/unmarking, and building BPT), and for IWS PAR 82.8% (resp 1.03%, 16.17%) of the time is spend on pixel labeling (resp. node marking/unmarking, and building BPT). Moreover, if we consider only one interaction step, in average, for NIWS the time taken by the different steps remain the same whereas for IWS

and PAR IWS the time spent for building the hierarchy completely vanish and, for both of them, about 99.5% of the time is spent on labeling CC, hence justifying our choice of focusing on parallel strategies for this particular step.

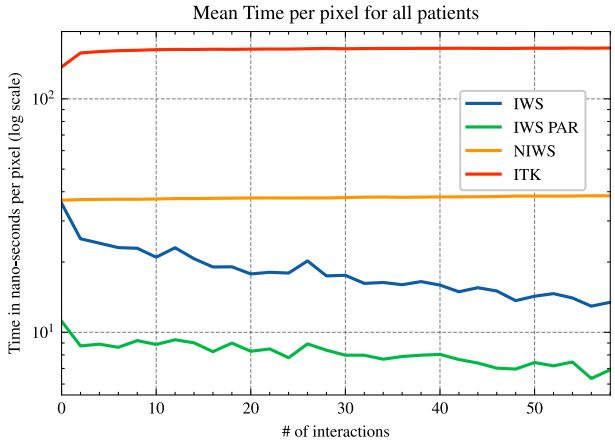

Fig. 7. Computation time in nano second per pixel on 3D images with simulated interactions. Our (IWS) is in blue, our parallelized version (IWS PAR) is in green, the non incremental version (NIWS) is in orange, ITK is in red. In order to avoid noisy visualization, the curve is smoothed by averaging values in a window of size 2.

6. Conclusion

We proposed an interactive seeded WS segmentation method that complies with an incremental process, exploiting causality within interactive sessions to achieve remarkable performance improvements and significantly enhance responsiveness. Additionally, we introduced a parallel method that leverages the properties of minimum spanning forest to further improve the responsiveness, especially for marker addition.

In future studies, we aim to investigate the optimal values for the parameters for parallel labeling to achieve a more significant speedup, particularly for a high number of processors/threads.

References

Allène, C., Audibert, J.-Y., Couprie, M., & Keriven, R. (2010). Some links between extremum spanning forests, watersheds and min-cuts. *Image and Vision Computing*, *28*, 1460–1471.

Arbeláez, P., Maire, M., Fowlkes, C., & Malik, J. (2011). Contour Detection and Hierarchical Image Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *33*, 898–916. doi:10.1109/TPAMI.2010.161. Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence.

Beare, R., & Lehmann, G. (2006). The watershed transform in itk - discussion and new developments, . doi:10.54294/lf8u75.

Beucher, S., & Meyer, F. (1993). The morphological approach to segmentation: The watershed transformation. *Mathematical Morphology in Image Processing*, *Vol. 34*, 433–481. doi:10.1201/9781482277234-12.

Bradski, G. (2000). The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, .

Carlinet, E., & Geraud, T. (2015). Morphological object picking based on the color tree of shapes. In *2015 IPTA* (pp. 125–130). Orleans, France: IEEE. doi:10.1109/IPTA.2015.7367111.

Cheng, H. K., Chung, J., Tai, Y.-W., & Tang, C.-K. (2020). CascadePSP: Toward class-agnostic and very high-resolution segmentation via global and local refinement. In *CVPR*.

Cousty, J., Bertrand, G., Najman, L., & Couprie, M. (2009). Watershed Cuts: Minimum Spanning Forests and the Drop of Water Principle. *IEEE TPAMI*, *31*, 1362–1374.

Cousty, J., Najman, L., & Perret, B. (2013a). Constructive links between some morphological hierarchies on edge-weighted graphs. In *Mathematical Morphology and Its Applications to Signal and Image Processing: 11th International Symposium, ISMM 2013, Uppsala, Sweden, May 27-29, 2013. Proceedings 11* (pp. 86–97). Springer.

Cousty, J., Najman, L., & Perret, B. (2013b). Constructive links between some morphological hierarchies on edge-weighted graphs. In *ISMM* (pp. 86–97).

Eschweiler, D., Spina, T. V., Choudhury, R. C., Meyerowitz, E., Cunha, A., & Stegmaier, J. (2019). CNN-Based Preprocessing to Optimize Watershed-Based Cell Segmentation in 3D Confocal Microscopy Images. (pp. 223–227). Piscataway, NJ: IEEE. Conference Name: 2019 IEEE 16th International Symposium on Biomedical Imaging (ISBI 2019).

Falcao, A., & Bergo, F. (2004). Interactive Volume Segmentation With Differential Image Foresting Transforms. *IEEE Transactions on Medical Imaging*, *23*, 1100–1108. doi:10.1109/TMI.2004.829335.

Falcão, A., Stolfi, J., & de Alencar Lotufo, R. (2004). The image foresting transform: theory, algorithms, and applications. *IEEE TPAMI*, *26*, 19–29. doi:10.1109/TPAMI.2004.1261076.

Jegou, H., Douze, M., & Schmid, C. (2008). Hamming embedding and weak geometric consistency for large scale image search. In D. Forsyth, P. Torr, & A. Zisserman (Eds.), *Computer Vision – ECCV 2008* (pp. 304–317). Berlin, Heidelberg: Springer Berlin Heidelberg.

Lebon, Q., Lefèvre, J., Cousty, J., & Perret, B. (2023). Interactive segmentation with incremental watershed cuts. In V. Vasconcelos, I. Domingues, & S. Paredes (Eds.), *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications* (pp. 189–200). Cham: Springer Nature Switzerland.

Lux, F., & Matula, P. (2019). Dic image segmentation of dense cell populations by combining deep learning and watershed. In *16th IEEE ISBI* (pp. 236–239). URL: https://doi.org/10.1109/ISBI.2019.8759594.

Machairas, V., Faessel, M., Cardenas, D., Chabardes, T., Walter, T., & Decencière, E. (2015). Waterpixels. *IEEE TIP*, *24*, 3707–3716. doi:10.1109/TIP.2015.2451011.

Meyer, F. (1992). Color image segmentation. In *1992 International Conference on Image Processing and its Applications* (pp. 303–306).

Meyer, F. (2012). The watershed concept and its use in segmentation : a brief history. URL: http://arxiv.org/abs/1202.0216.

Najman, L., Cousty, J., & Perret, B. (2013). Playing with Kruskal: algorithms for morphological trees in edge-weighted graphs. In *ISMM* (pp. 135–146).

Najman, L., & Schmitt, M. (1996). Geodesic saliency of watershed contours and hierarchical segmentation. *IEEE TPAMI*, *18*, 1163–1173. doi:10.1109/34.546254.

Ngoc, M. Ô. V., Carlinet, E., Fabrizio, J., & Géraud, T. (2023). The dahu graph-cut for interactive segmentation on 2d/3d images. *Pattern Recognition*, *136*, 109–207.

Passat, N., Naegel, B., Rousseau, F., Koob, M., & Dietemann, J.-L. (2011). Interactive segmentation based on component-trees. *Pattern Recognition*, *44*, 2539–2554. doi:10.1016/j.patcog.2011.03.025. Semi-Supervised Learning for Visual Content Analysis and Understanding.

Perret, B., Chierchia, G., Cousty, J., F. Guimarães, S., Kenmochi, Y., & Najman, L. (2019). Higra: Hierarchical Graph Analysis. *SoftwareX*, *10*, 100335. doi:10.1016/j.softx.2019.100335.

Salembier, P., & Garrido, L. (2000). Binary partition tree as an efficient representation for image processing, segmentation, and information retrieval. *TIP*, *9*, 561–576.

Soler, L., Hostettler, A., Agnus, V., Charnoz, A., Fasquel, J., Moreau, J., Osswald, A., Bouhadjar, M., & Marescaux, J. (2010). 3d image reconstruction for comparison of algorithm database: A patient specific anatomical and medical image database. *IRCAD, Strasbourg, France, Tech. Rep*, *1*.

Vincent, L., & Soille, P. (1991). Watersheds in digital spaces: an efficient algorithm based on immersion simulations. *IEEE TPAMI*, *13*, 583–598. doi:10.1109/34.87344.

Wolf, S., Schott, L., Köthe, U., & Hamprecht, F. (2017). Learned Watershed: End-to-End Learning of Seeded Segmentation. doi:10.48550/arXiv.1704.02249 arXiv:1704.02249 [cs].

Youkana, I., Cousty, J., Saouli, R., & Akil, M. (2017). Parallelization strategy for elementary morphological operators on graphs: distance-based algorithms and implementation on multicore shared-memory architecture. *JMIV*, *59*, 136–160.