Higra: Hierarchical Graph Analysis

Benjamin Perret, Giovanni Chierchia, Jean Cousty, Silvio Jamil F. Guimarães, Yukiko Kenmochi, Laurent Najman

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

Higra: Hierarchical Graph Analysis

B. Perret [a,*], G. Chierchia [a], J. Cousty [a], S.J. F. Guimarães [b,a], Y. Kenmochi [a], L. Najman [a]

[a] Université Paris-Est, Laboratoire d'Informatique Gaspard-Monge UMR 8049, UPEMLV, ESIEE Paris, ENPC, CNRS, F-93162 Noisy-le-Grand, France
[b] PUC Minas - ICEI - DCC - VIPLAB, Brazil

ARTICLE INFO

Article history:
Received 19 July 2019
Received in revised form 10 September 2019
Accepted 26 September 2019

Keywords:
Graph
Hierarchical clustering
Component tree

ABSTRACT

Higra — Hierarchical Graph Analysis is a C++/Python library for efficient sparse graph analysis with a special focus on hierarchical methods capable of handling large amount of data. The main aspects of hierarchical graph analysis addressed in Higra are the construction of hierarchical representations (agglomerative clustering, mathematical morphology hierarchies, etc.), the analysis and processing of such representations (filtering, clustering, characterization, etc.), and their assessment. Higra targets a large audience, from students and practitioners wanting an accessible library for quickly experimenting, to researchers developing new methods for hierarchical analysis of graph data. Higra is a generic toolbox for graph analysis and can be utilized in a large variety of application fields like machine learning, data science, pattern analysis and computer vision. Moreover, it contains an *image analysis* module easing the handling of pixel grid graphs by providing efficient algorithms dedicated to this field.

Code metadata

Version	0.4.1
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX_2019_243
Code Ocean compute capsule	NA
Legal Code License	Cecill-B
Code versioning system used	git
Software code languages, tools, and services used	C++, python
Compilation requirements, operating environments & dependencies	C++ 14 compiler, Python, Numpy
If available Link to developer documentation/manual	https://higra.readthedocs.io
Support email for questions	benjamin.perret@esiee.fr

1. Motivation and significance

Graphs are getting increasingly popular in machine learning, data science, pattern analysis and computer vision, as they provide a natural representation for structured or network data. Among graph analysis methods, hierarchical approaches try to capture the structure of a graph at various scales in a consistent manner by sequentially merging graph vertices into increasingly larger clusters until a single cluster remains. Such methods have proven to be useful in a wide variety of application fields in which data can be modelled as graphs such as image analysis, community detection, mesh analysis, philogenetic tree construction, and so on.

Higra — Hierarchical Graph Analysis is a C++/Python library for efficient sparse graph analysis with a special focus on hierarchical methods. It aims at providing standard and state-of-the-art algorithms for hierarchical graph analysis capable of handling large amount of data (currently up to dozen of millions of vertices on a classical desktop computer). It can handle both hierarchical clustering and component trees, where each level of a hierarchy represents a partial clustering of the space. Furthermore, the design of Higra enables the user to easily switch between the dual representations of hierarchical clustering: trees (dendrograms) and saliency maps (ultrametric distances). Higra is a generic toolbox for hierarchical graph representation construction, processing and assessment: it is thus not focused toward a particular application or domain and its fundamental functions can be used in a variety of situations.

* Corresponding author.
E-mail address: benjamin.perret@esiee.fr (B. Perret).

https://doi.org/10.1016/j.softx.2019.100335

Higra provides a Python API to ease its usage and to benefit from the synergies created by the large amount of scientific libraries available in the Python ecosystem. This API is thought to be useable by students and expert researchers willing to quickly develop new applications, experiment new methods, or develop proofs of concepts. Vectorized operations on hierarchical representations enable writing various algorithms working on hierarchical representations efficiently in Python. It is also thought for seamless integration with classical Python data analysis pipelines such as *Scikit-Learn* and modern optimization framework such as *PyTorch* and *TensorFlow*. The Python interface is backed-up by a C++ module where core algorithms are implemented to ensure high performances. The C++ module is also useable as a standalone library since it does not have any dependency to the Python runtime.

Graphs are a common representation for data and many libraries exist to analyse them. Regarding generic toolbox for graph processing with a Python interface, the main publicly available libraries are *graph-tool* [1], *NetworkX* [2], and *igraph* [3]. While they propose numerous algorithms for graph analysis, they offer little or no support for hierarchical analysis. Among major Python data analysis libraries, *Scipy* [4] has a module dedicated to hierarchical clustering but it is limited to complete graphs. On the contrary, *Scikit-Learn* [5] has a module dedicated to agglomerative clustering which can handle sparse graph. While it enables creating flat clustering, also called partitioning, from a sparse graph using classical agglomerative clustering algorithms, hierarchies cannot be easily manipulated by users. The library *mlpack* [6], which is a C++ machine-learning library with Python bindings, also provides some hierarchical methods for point classification. It also possesses numerous algorithms using hierarchical representation of metric spaces and focused around the problem of efficient distance computation; these algorithms and the associated data structures are however not exposed in the user API. Hierarchical clustering methods have also become popular in image analysis and several libraries implement algorithms dedicated to application domains and, sometimes proposing Python bindings, like *Pink* [7], *Olena* [8], *Vigra* [9], or the more specialized *iamxt* [10]. Up to our knowledge, Higra is thus the first C++/Python library providing a large variety of algorithms dedicated to hierarchical graph analysis and not targeting a particular application.

This article is organized as follows. Section 2 presents the main aspects of the library architecture and gives a summary of its major functionalities. Section 3 provides two illustrative examples of the library on graph simplification and image filtering. The computational performances of some standard agglomerative hierarchical clustering algorithms are also assessed. Section 4 analyses the expected impact of the proposed library. Section 5 concludes the article.

2. Software description

In this section, we describe the structure of the library and the major functionalities implemented.

2.1. Software architecture

Higra is composed of a core, header-only, C++ 14 module, where performance critical data structures and algorithms are implemented, and of a Python module exposing the C++ module API and proposing higher level functions.

In Higra, graphs are analysed through their hierarchical representations. The hierarchical representation of a graph corresponds to a tree where each node is a cluster, also called a region, of the input graph. The two main data structures of the library are thus the graph class, implemented as an adjacency list, and the tree class, implemented as a parent array [11].

These classes are lowly coupled to their associated data, vertex and edge weights for graphs, and node weights for trees. Those data are represented by multi-dimensional arrays using the *xtensor* library [12] in C++ and *Numpy* [13] arrays in Python, hence enabling their easy and efficient manipulation with standard array programming. This separation is achieved by imposing that all the elements of a graph (vertices and edges) and all the elements of a tree (nodes) are identified by integers indices ranging from 0 to the number of elements minus one. Moreover, in order to facilitate the exchange of data between a graph and its hierarchical representation as a tree, we also impose that the indices of the tree leaves are exactly the indices of the vertices of the graph. In other words, an array representing vertex features on the graph can be used as an array representing leaf node features on the tree.

All dependencies of the core C++ module are header-only C++ library, thus ensuring that the library can be easily compiled and extended on a large variety of systems. C++ functions and data structures are mapped to Python types with Pybind11 [14]. Note that *xtensor* arrays are mapped seamlessly to *Numpy* arrays.

The library is carefully tested with more than 98% (resp. 90%) lines of codes of the C++ (resp. Python) module covered by unit tests. Continuous integration is used to avoid regression issues on major systems (Linux, Mac, and Windows).

2.2. Software functionalities

Higra contains a large amount of classical and recent algorithms for the construction, the manipulation, and the analysis of hierarchical graph representations:

- **efficient methods and data structures to handle the dual representations of hierarchical clustering**: trees [15] (dendrograms) and saliency maps [16] (ultrametric distances);
- **hierarchical clusterings**: quasi-flat zone hierarchy [17], hierarchical watersheds [18,19], agglomerative clustering [20] (single-linkage [11,21], average-linkage, complete-linkage, exponential-linkage [22], Ward, or user-provided linkage rule), constrained connectivity hierarchy [23];
- **component trees**: min and max trees [24,25];
- **manipulate and explore hierarchies**: simplification [26, 27], accumulators, cluster extraction, various attributes [28] (size, volume, dynamics, perimeter, compactness, moments, etc.), horizontal and non-horizontal cuts, alignment of hierarchies [29];
- **optimization on hierarchies**: optimal cuts, energy hierarchies [30,31];
- **algorithms on graphs**: accumulators, vertices and clusters dissimilarities, region adjacency graphs, minimum spanning trees and forests, watershed cuts [32];
- **assessment**: supervised assessment of graph clusterings and hierarchical clusterings [33,34];
- **image toolbox**: special methods for grid graphs, tree of shapes [35], hierarchical clustering methods dedicated to image analysis [36], optimization of Mumford–Shah energy [37].

3. Illustrative examples

This section presents illustrative examples of Higra usage and performances. The first example demonstrates the use of hierarchy simplification to improve clustering performances. The second example shows how Higra can be used to perform image filtering. Finally, the last example compares the performance of

Higra agglomerative clustering algorithms to their Scikit-Learn equivalent. These illustrations can be reproduced by downloading the Python notebook.[1]

3.1. Hierarchical clustering simplification

Fig. 1 demonstrates the construction of a single-linkage hierarchical clustering and its simplification by a cluster size criterion [26]. One classical issue with the single-linkage clustering is the presence of very small clusters branching very high in the hierarchy. Such clusters are non-relevant and can be sent back to the very bottom of the hierarchy. In the example, a hierarchy is constructed in line 7 from a graph made of 3 clusters. One can see that, in Higra, a hierarchy is indeed a tuple of 2 elements: a tree and an array which associates an altitude with each node of the tree (also called tree distance in the literature). A flat clustering into three classes is extracted from the hierarchy on line 10. The hierarchy is simplified on line 15: all clusters whose size is smaller than 7 elements are removed. Finally, the dendrogram of the simplified hierarchy is computed on line 16 and the new 3 class clustering is extracted on line 19. We can see that the simplified hierarchy does not contain any small cluster anymore and the corresponding flat clustering into 3 classes is greatly improved.

3.2. Image filtering with a watershed hierarchy

The example in Fig. 2 demonstrates the use of hierarchical clustering for image filtering [38]. The strategy followed here is to first construct a watershed hierarchy by area [18,19] of the gradient of the image represented as an edge-weighted graph. The basic idea of the watershed segmentation [39] is to consider an edge-weighted graph as a topographic surface and to associate each catchment basin of this topographic surface with a cluster. Intuitively, the watershed hierarchy by area is then obtained by sequentially filtering the edge weights of the graph with area closings [40] of increasing sizes and then computing the sequence of watershed segmentations of these filtered edge weights. Then, a flat clustering containing k clusters is extracted from the hierarchical representation. Finally, the colour, in the filtered image, of each pixel contained in a cluster is replaced by the mean colour, in the original image, of the pixels inside the cluster.

A 4-adjacency edge weighted graph is built from the gradient of an image on lines 5 and 6. Then a watershed hierarchy by area of the graph is constructed on line 9. The saliency map of the hierarchy, which weights each edge of the graph by the ultrametric distance between its extremities, is computed for illustrative purpose on line 10 and is plotted in the 2D Khalimsky grid [41,42] on line 12. This representation, sometimes called *ultrametric contour map* [43], enables to visualize hierarchical clustering constructed on a 4-adjacency graph as a contour image, where the strength of a contour is (inversely) proportional to its brightness. Then, the mean image colour inside each region of the hierarchy is computed on line 16. The object of class `HorizontalCutExplorer`, instantiated on line 19, eases the construction of horizontal-cuts (flat clustering) of the given hierarchy. It is used to extract several cuts containing different number of regions (line 19) and the images corresponding to these cuts are reconstructed using the mean image colour of their regions (line 20).

[1] https://higra.readthedocs.io/en/latest/notebooks.html.

3.3. Performance comparison with Scikit-Learn

The performance of the agglomerative hierarchical clustering algorithms of Higra and Scikit-Learn are compared in Fig. 3. We generated sparse graphs of various sizes using a k-nearest neighbours algorithm (with $k = 10$) on uniformly sampled points in $[0, 1]^2$. For each generated graph, we ran the single, complete, average, and Ward linkage agglomerative clustering algorithms from Higra (functions `binary_partition_tree_X_linkage` where `X` is replaced by the name of the linkage rule) and Scikit-Learn (functions `linkage_tree` for single, complete and average linkage rules, and `ward_tree` for Ward linkage) 7 times and we computed the mean and standard deviation runtime for each graph size. We can see that Higra is constantly faster than Scikit-learn by a factor comprised between 4 and 10 and the gap seems to increase as the size of the graphs increases.

4. Impact

Up to our knowledge, Higra is the first library that gathers so many algorithms for hierarchical graph analysis in one place. As such, providing reference implementations of classical and state-of-the-art algorithms will already provide great benefit by favouring the spread of those methods among the community, by easing reproducible research, and by helping the creation of relevant comparisons between methods.

As Higra works on generic graphs and does not assume a particular application, it can be used in a large variety of fields. While its usage requires some knowledge on hierarchical representations, it provides an easy-to-use Python interface which is extensively documented. This can further help to spread the use of the proposed approaches in the scientific community as well as serving as a pedagogical tool for teachers.

While some algorithms still have to be written in C++ to ensure high performances, we have found that the set of basic operators provided in the library used in conjunction with the vectorization of many operations on hierarchies indeed enables to directly write many algorithms acting on hierarchies efficiently in Python in a *Numpy style*. This enables to considerably simplify and speed up the development of new methods compared to classical C/C++ programming.

Higra is a new library, and as such, still has a limited base of contributors and users. Some of its core functions have been used for a long time in our research group and led to dozens of publications [11,16,19,32,34,44–48] (among others) and the usage of the library is spreading in new publications [26,49]. We plan to incorporate all our future research works on hierarchical graph representations in the library and we hope that Higra will benefit from a large adoption from the community leading to external users and contributors.

5. Conclusions

We presented Higra, a library for hierarchical sparse graph analysis that contains many standard and state-of-the-art algorithms in this field. Higra is made of a core C++ module and a user friendly Python interface. Pre-compiled binaries of Higra are available for Linux, Mac, and Windows 64 bits systems on the *Python Package Index - Pypi* and can be installed with a simple command: `pip install higra`.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

```
# create a graph from a random 2d point cloud
X, labels = sklearn.datasets.make_blobs(200, cluster_std=[1.0, 2.5, 0.5])
graph, edge_weights = make_graph_from_points(X, 'delaunay')
plot_graph(graph, vertex_positions=X, vertex_labels=labels)

# single-linkage clustering
tree, altitudes = binary_partition_tree_single_linkage(graph, edge_weights)
plot_partition_tree(tree, altitudes=altitudes, n_clusters=3)

# flat clustering
pred = labelisation_horizontal_cut_from_num_regions(tree, altitudes, 3)
plot_graph(graph, vertex_positions=X, vertex_labels=pred)

# hierarchy simplification
tree, altitudes = filter_small_nodes_from_tree(tree, altitudes, 7)
plot_partition_tree(tree, altitudes=altitudes, n_clusters=3)

# flat clustering
pred = labelisation_horizontal_cut_from_num_regions(tree, altitudes, 3)
plot_graph(graph, vertex_positions=X, vertex_labels=pred)
```

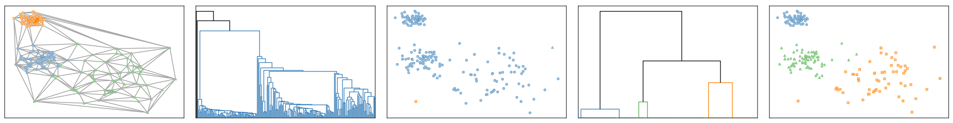

Fig. 1. Hierarchical clustering simplification. From left to right: the input graph with the true vertex labels, the dendrogram of the single linkage hierarchy, the horizontal cut composed of 3 clusters in the single linkage hierarchy, the dendrogram of the simplified single linkage hierarchy, and the horizontal cut composed of 3 clusters in the simplified single linkage hierarchy.

```
image    = imread("101087.jpg").astype(np.float64) / 255
gradient = imread("101087_SED.png").astype(np.float64) / 255

# Edge weighted 4-adjacency graph
graph        = get_4_adjacency_graph(gradient.shape[:2])
edge_weights = weight_graph(graph, gradient, WeightFunction.mean)

# Watershed hierarchy by area and its saliency map
tree, altitudes = watershed_hierarchy_by_area(graph, edge_weights)
sm              = saliency(tree, altitudes)
imshow(image); imshow(1 - gradient)
imshow(1 - graph_4_adjacency_2_khalimsky(graph, sm) ** 0.5)

# Get horizontal cuts containing different number of regions
# and colorize them with the mean pixel values inside each region
mean_color = attribute_mean_weights(tree, image)
cut_helper = HorizontalCutExplorer(tree, altitudes)
for c in [25, 50, 100]:
    cut        = cut_helper.horizontal_cut_from_num_regions(c)
    simplified = cut.reconstruct_leaf_data(tree, mean_color)
    imshow(simplified)
```

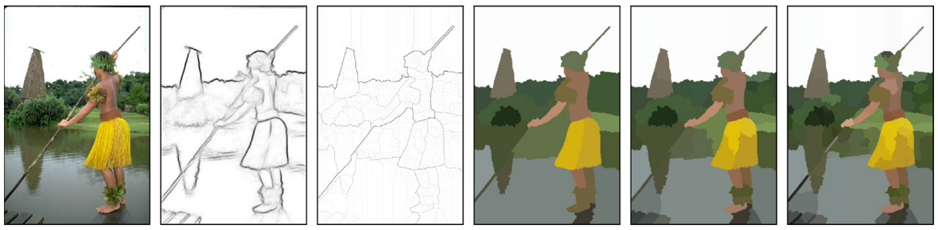

Fig. 2. Image simplification with a watershed hierarchy. From left to right: original image, gradient, saliency map of the watershed hierarchy by area of the gradient, simplified image reconstructed from the hierarchy with respectively 25, 50, and 100 regions.

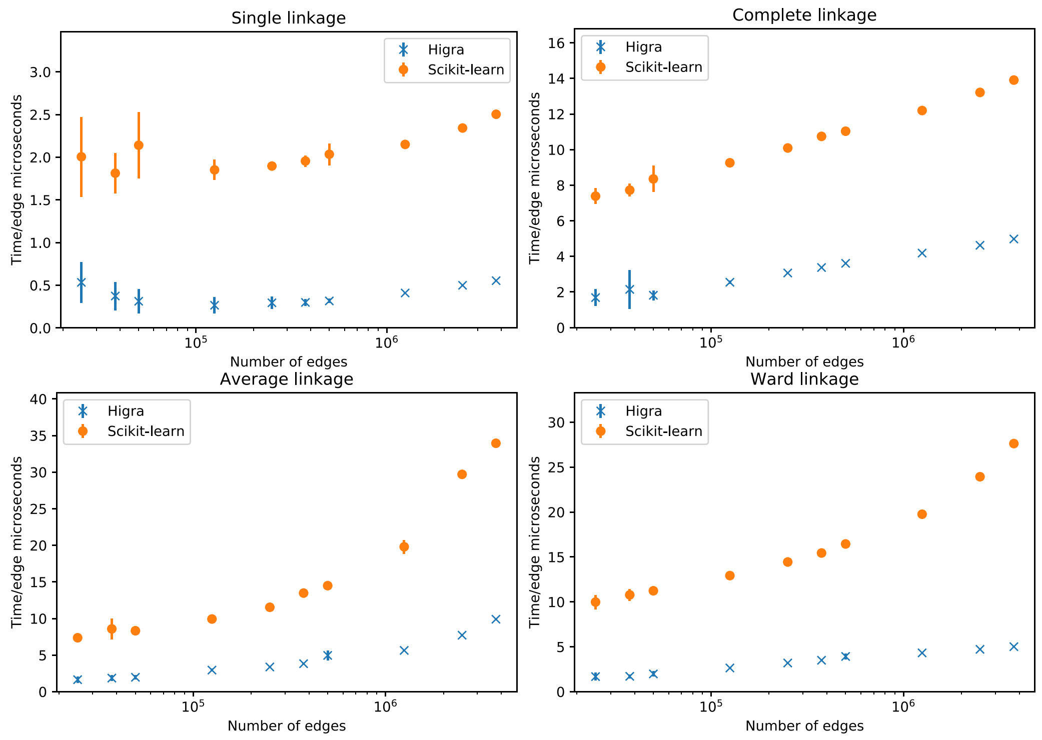

Fig. 3. Performance comparison between Higra and Scikit-Learn on single, complete, average, and Ward linkage agglomerative clustering.

Acknowledgements

The authors are grateful to CNPq (Universal 421521/2016-3 and PQ 307062/2016-3), FAPEMIG (PPM-00006-16 and PPM-00006-18) and PUC Minas for the financial support to this work. This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001 and by CAPES/COFECUB 88887.191730/2018-00. We also acknowledge financial support from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 721463 to the SUNDIAL ITN network

References

[1] Graph-Tool. https://graph-tool.skewed.de/. [Online; Accessed 27 June 2019].
[2] NetworkX. https://networkx.github.io/. [Online; Accessed 27 June 2019].
[3] igraph. https://igraph.org/. [Online; Accessed 27 June 2019].
[4] Jones E, Oliphant T, Peterson P, et al. SciPy: Open source scientific tools for Python. http://www.scipy.org/. [Online; Accessed 27 June 2019].
[5] Scikit-Learn. https://scikit-learn.org/. [Online; Accessed 27 June 2019].
[6] Curtin RR, Edel M, Lozhnikov M, Mentekidis Y, Ghaisas S, Zhang S. mlpack 3: a fast, flexible machine learning library. J Open Source Softw 2018;3:726. http://dx.doi.org/10.21105/joss.00726, URL https://doi.org/10.21105/joss.00726.
[7] Pink. https://perso.esiee.fr/coupriem/Pink/doc/html/. [Online; Accessed 27 June 2019].
[8] Olena. https://www.lrde.epita.fr/wiki/Olena. [Online; Accessed 27 June 2019].
[9] Vigra. https://github.com/ukoethe/vigra. [Online; Accessed 27 June 2019].
[10] Souza R, Rittner L, Machado R, Lotufo R. iamxt: Max-tree toolbox for image processing and analysis. SoftwareX 2017;6:81–4. http://dx.doi.org/10.1016/j.softx.2017.03.001.
[11] Najman L, Cousty J, Perret B. Playing with Kruskal: Algorithms for morphological trees in edge-weighted graphs. In: Hendriks CLL, Borgefors G, Strand R, editors. Mathematical morphology and its applications to signal and image processing. Lecture notes in computer science, vol. 7883, Berlin, Heidelberg: Springer; 2013, p. 135–46. http://dx.doi.org/10.1007/978-3-642-38294-9_12.
[12] xtensor. https://github.com/QuantStack/xtensor. [Online; Accessed 27 June 2019].
[13] Oliphant T. NumPy: A guide to NumPy. USA: Trelgol Publishing; 2006, URL http://www.numpy.org/. [Online; Accessed 27 June 2019].
[14] Pybind11. https://github.com/pybind/pybind11. [Online; Accessed 27 June 2019].
[15] Carlsson G, Mémoli F. Characterization, stability and convergence of hierarchical clustering methods. J Mach Learn Res 2010;11:1425–70.
[16] Cousty J, Najman L, Kenmochi Y, Guimarães S. Hierarchical segmentations with graphs: Quasi-flat zones, minimum spanning trees, and saliency maps. J Math Imaging Vision 2018;60(4):479–502.
[17] Meyer F, Maragos P. Morphological scale-space representation with levelings. In: Nielsen M, Johansen P, Olsen OF, Weickert J, editors. Scale-space theories in computer vision. Berlin, Heidelberg: Springer; 1999, p. 187–98.
[18] Meyer F. The dynamics of minima and contours. In: Maragos P, Schafer RW, Butt MA, editors. Mathematical morphology and its applications to image and signal processing. US, Boston, MA: Springer; 1996, p. 329–36. http://dx.doi.org/10.1007/978-1-4613-0469-2_38.
[19] Cousty J, Najman L. Incremental algorithm for hierarchical minimum spanning forests and saliency of watershed cuts. In: Soille P, Pesaresi M, Ouzounis GK, editors. Mathematical morphology and its applications to image and signal processing. Berlin, Heidelberg: Springer; 2011, p. 272–83.
[20] Murtagh F, Contreras P. Algorithms for hierarchical clustering: an overview. Data Min Knowl Discov 2012;2(1):86–97.
[21] Gower JC, Ross GJS. Minimum spanning trees and single linkage cluster analysis. J R Stat Soc C 1969;18(1):54–64.
[22] Yadav N, Kobren A, Monath N, Mccallum A. Supervised hierarchical clustering with exponential linkage. In: Chaudhuri K, Salakhutdinov R, editors. Proceedings of the 36th international conference on machine learning. Proceedings of machine learning research, vol.97, Long Beach, California, USA: PMLR; 2019, p. 6973–83.
[23] Soille P. Constrained connectivity for hierarchical image partitioning and simplification. IEEE Trans Pattern Anal Mach Intell 2008;30(7):1132–45. http://dx.doi.org/10.1109/TPAMI.2007.70817.
[24] Salembier P, Oliveras A, Garrido L. Anti-extensive connected operators for image and sequence processing. IEEE Trans Image Process 1998;7(4):555–70. http://dx.doi.org/10.1109/83.663500.
[25] Jones R. Connected filtering and segmentation using component trees. Comput Vis Image Underst 1999;75(3):215–28. http://dx.doi.org/10.1006/cviu.1999.0777.
[26] Perret B, Cousty J, Guimaraes SJF, Kenmochi Y, Najman L. Removing non-significant regions in hierarchical clustering and segmentation. 2019, p. 1–7, submitted for publication.
[27] Barcelos IB, da Fonseca GB, Najman L, Kenmochi Y, Perret B, Cousty J, et al. Exploring hierarchy simplification for non-significant region removal. In: 32th SIBGRAPI conference on graphics, patterns and images, Rio de Janeiro, Brazil, October, 2019; 2019. p. 1–7. (accepted for publication).

[28] Zhang D, Lu G. Review of shape representation and description techniques. Pattern Recognit 2004;37(1):1–19.

[29] Pont-Tuset J, Arbeláez P, Barron JT, Marques F, Malik J. Multiscale combinatorial grouping for image segmentation and object proposal generation. IEEE Trans Pattern Anal Mach Intell 2017;39(1):128–40. http://dx.doi.org/10.1109/TPAMI.2016.2537320.

[30] Guigues L, Cocquerez JP, Le Men H. Scale-sets image analysis. Int J Comput Vis 2006;68(3):289–317. http://dx.doi.org/10.1007/s11263-005-6299-0.

[31] Kiran BR, Serra J. Global–local optimizations by hierarchical cuts and climbing energies. Pattern Recognit 2014;47(1):12–24. http://dx.doi.org/10.1016/j.patcog.2013.05.012.

[32] Cousty J, Bertrand G, Najman L, Couprie M. Watershed cuts: Minimum spanning forests and the drop of water principle. IEEE Trans Pattern Anal Mach Intell 2009;31(8):1362–74.

[33] Heller KA, Ghahramani Z. Bayesian hierarchical clustering. In: Proceedings of the 22nd international conference on machine learning. ICML '05, ACM; 2005, p. 297–304. http://dx.doi.org/10.1145/1102351.1102389.

[34] Perret B, Cousty J, Guimarães SJ, Maia DS. Evaluation of hierarchical watersheds. IEEE Trans Image Process 2018;27(4):1676–88. http://dx.doi.org/10.1109/TIP.2017.2779604.

[35] Géraud T, Carlinet E, Crozet S, Najman L. A quasi-linear algorithm to compute the tree of shapes of nD images. In: Hendriks CLL, Borgefors G, Strand R, editors. Mathematical morphology and its applications to signal and image processing. Berlin, Heidelberg: Springer; 2013, p. 98–110.

[36] Maninis K, Pont-Tuset J, Arbeláez P, Gool LV. Convolutional oriented boundaries: From image segmentation to high-level tasks. IEEE Trans Pattern Anal Mach Intell 2018;40(4):819–33.

[37] Mumford D, Shah J. Optimal approximations by piecewise smooth functions and associated variational problems. Comm Pure Appl Math 1989;42(5):577–685. http://dx.doi.org/10.1002/cpa.3160420503.

[38] Salembier P, Garrido L. Binary partition tree as an efficient representation for image processing, segmentation and information retrieval. IEEE Trans Image Process 2000;9(4):561–76.

[39] Beucher S, Meyer F. The morphological approach to segmentation: the watershed transformation. Math Morphology Image Process 1993;34:433–81.

[40] Vincent L. Morphological area openings and closings for grey-scale images. In: O Y-L, Toet A, Foster D, Heijmans HJAM, Meer P, editors. Shape in picture. Berlin, Heidelberg: Springer; 1994, p. 197–208.

[41] Kovalevsky VA. Finite topology as applied to image analysis. Comput Vis Graph Image Process 1989;46(2):141–61.

[42] Khalimsky E, Kopperman R, Meyer PR. Computer graphics and connected topologies on finite ordered sets. Topology Appl 1990;36(1):1–17.

[43] Arbelaez P, Maire M, Fowlkes C, Malik J. Contour detection and hierarchical image segmentation. IEEE Trans Pattern Anal Mach Intell 2011;33(5):898–916. http://dx.doi.org/10.1109/TPAMI.2010.161.

[44] Perret B, Collet C. Connected image processing with multivariate attributes: an unsupervised Markovian classification approach. Comput Vis Image Underst 2015;133:1–14. http://dx.doi.org/10.1016/j.cviu.2014.09.008.

[45] Santana Maia D, de Albuquerque Araujo A, Cousty J, Najman L, Perret B, Talbot H. Evaluation of combinations of watershed hierarchies. In: Angulo J, Velasco-Forero S, Meyer F, editors. Mathematical morphology and its applications to signal and image processing: 13th international symposium, ISMM 2017, Fontainebleau, France, may 15–17, 2017, proceedings. Springer International Publishing; 2017, p. 133–45. http://dx.doi.org/10.1007/978-3-319-57240-6_11.

[46] Maia Santana D, Cousty J, Najman L, Perret B. Recognizing hierarchical watersheds. In: Couprie M, Cousty J, Kenmochi Y, Mustafa N, editors. Discrete geometry for computer imagery. Cham: Springer International Publishing; 2019, p. 300–13.

[47] Santana Maia D, Cousty J, Najman L, Perret B. Watersheding hierarchies. In: Burgeth B, Kleefeld A, Naegel B, Passat N, Perret B, editors. Mathematical morphology and its applications to signal and image processing. Cham: Springer International Publishing; 2019, p. 124–36.

[48] Robic J, Perret B, Nkengne A, Couprie M, Talbot H. Self-dual pattern spectra for characterising the dermal-epidermal junction in 3D reflectance confocal microscopy imaging. In: Burgeth B, Kleefeld A, Naegel B, Passat N, Perret B, editors. Mathematical morphology and its applications to signal and image processing. Cham: Springer International Publishing; 2019, p. 508–19.

[49] Chierchia G, Perret B. Ultrametric fitting by gradient descent. In: Advances in neural information processing systems, vol. 32. NeurIPS, 2019, in press. arXiv:1905.10566.