Out-of-core Attribute Algorithms for Binary Partition Hierarchies

Josselin Lefèvre, Jean Cousty, Benjamin Perret, Harold Phelippeau

Out-of-core Attribute Algorithms for Binary Partition Hierarchies

Josselin Lefèvre[1,2], Jean Cousty[1], Benjamin Perret[1], Harold Phelippeau[2]

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

Abstract. Binary Partition Hierarchies (BPHs) and Minimum Spanning Trees are key structures in hierarchical image analysis. However, the explosion in the size of image data poses a new challenge, as the memory available in conventional workstations becomes insufficient to execute classical algorithms. To address this problem, specific algorithms have been proposed for out-of-core computation of BPHs, where a BPH is actually represented by a collection of smaller trees, called a distribution, thus reducing the memory footprint of the algorithms. In this article, we address the problem of designing efficient out-of-core algorithms for computing classical attributes in distributions of BPHs, which is a necessary step towards a complete out-of-core hierarchical analysis workflow that includes tasks such as connected filtering and the generation of other representations such as hierarchical watersheds. The proposed algorithms are based on generic operations designed to propagate information through the distribution of trees, enabling the computation of attributes such as area, volume, height, minima and number of minima.

1 Introduction

Hierarchies are versatile representations that are useful in many image analysis and processing problems and are attracting increasing interest [15]. Among them, binary partition hierarchies [2,18] (BPHs) paired to minimum spanning trees are key structures for several (hierarchical) segmentation methods: in particular, it has been shown [2,14] that the BPH can be used to efficiently compute quasi-flat zone hierarchies [2,13] (also called α-trees), watershed hierarchies [2,12] or seed-based watersheds [10]. Efficient algorithms for building BPHs on standard size images are well established [14], but with the continuous improvement of image acquisition systems, the image resolutions are increasing dramatically, resulting in images that can reach dozens of gigabytes. In [3, 11] the authors proposed algorithms to compute the BPH under the out-of-core constraint, *i.e.*, when the goal is to minimize the amount of memory required by the algorithms. In this framework, the BPH is spread into a collection of smaller local hierarchies called a distribution. Each local hierarchy is represented by a tree data structure that is small enough to fit in the main memory of a classical workstation.

In this article, we address the problem of the out-of-core computation of tree attributes, which is a necessary step to obtain a complete out-of-core hierarchical image analysis pipeline including for example: connected filters [17],

attribute openings [1] or extinction values computation [19]. Several authors have already explored the issue of attribute computation in related contexts, such as incremental attribute computation [20] and parallel or distributed algorithms for hierarchical image analysis. In [4,7,9], the authors investigate distributed memory algorithms for computation of min and max trees to perform user-defined attribute filtering and multiscale analysis of terabytes images. In [5], the method was extended to allow a posteriori attribute computation on distributed component trees. In [6], the computation of minimum spanning trees of streaming images is considered whereas a parallel algorithm for the computation of quasi-flat zones hierarchies has been proposed in [8], allowing to efficiently implement interactive filtering segmentation of video.

In this study, we propose a generic scheme and detailed algorithms for computing regional attributes on the distribution of a BPH under the out-of-core constraint. These algorithms do not require any additional global data structure and only require having two trees of the distribution simultaneously in memory. We consider classical geometric attributes such as area, volume, height, or rightmost intersecting slice of a region, and topological attributes such as the Boolean attribute indicating whether a region is a regional minimum and the number of regional minima included in each region. The implementation of the method in C++ and Python based on the hierarchical graph processing library Higra [16] is available online `https://github.com/PerretB/Higra-distributed`.

This article is organized as follows. Section 2 gives the formal problem statement recalling the notion of BPH and distribution of BPH before introducing the distribution of an attribute. Section 3 explains the proposed data structures and presents a general scheme for out-of-core attribute computation. Section 4 presents uses of this scheme for calculating attributes such as area, volume, height, local minima and number of minima.

2 Out-of-core Attribute Computation: Problem Statement

In this section, we give formal definitions of a BPH and of the distribution of a BPH, and we state the problem of computing an attribute over the distribution of a BPH.

2.1 Binary Partition Hierarchy by Altitude Ordering

Let us first recall the definition of a hierarchy of partitions. Then we define the binary partition hierarchy by altitude ordering.

Let V be a set. A *partition of* V is a set of pairwise disjoint subsets of V. Any element of a partition is called a *region* of this partition. The *ground* of a partition $\mathbf{P}$, denoted by $gr(\mathbf{P})$, is the union of the regions of $\mathbf{P}$. A partition whose ground is V is called a *complete partition of* V. Let $\mathbf{P}$ and $\mathbf{Q}$ be two partitions of V. We say that $\mathbf{Q}$ is a *refinement of* $\mathbf{P}$ if any region of $\mathbf{Q}$ is included in a region of $\mathbf{P}$. A *hierarchy on* V is a sequence $(\mathbf{P}_0, \ldots, \mathbf{P}_\ell)$ of partitions of V

such that, for any λ in $\{0,\dots,\ell-1\}$, the partition $\mathbf{P}_\lambda$ is a refinement of $\mathbf{P}_{\lambda+1}$. Let $\mathcal{H} = (\mathbf{P}_0,\dots,\mathbf{P}_\ell)$ be a hierarchy. The integer ℓ is called the *depth of* $\mathcal{H}$ and, for any λ in $\{0,\dots,\ell\}$, the partition $\mathbf{P}_\lambda$ is called the λ-*scale of* $\mathcal{H}$. In the following, if λ is an integer in $\{0,\dots,\ell\}$, we denote by $\mathcal{H}[\lambda]$ the λ-scale of $\mathcal{H}$. For any λ in $\{0,\dots,\ell\}$, any region of the λ-scale of $\mathcal{H}$ is also called a *region of* $\mathcal{H}$. The hierarchy $\mathcal{H}$ is *complete* if $\mathcal{H}[0] = \{\{x\} \mid x \in V\}$ and if $\mathcal{H}[\ell] = \{V\}$. We denote by $\mathcal{H}_\ell(V)$ the set of all hierarchies on V of depth ℓ, by $\mathcal{P}(V)$ the set of all partitions on V, and by $2^{|V|}$ the set of all subsets of V.

In the following, the symbol ℓ stands for any strictly positive integer.

We define a *graph* as a pair $G = (V, E)$ where V is a finite set and E is composed of ℓ unordered pairs of distinct elements in V. Each element of V is called a *vertex of* G, and each element of E is called an *edge of* G. The Binary Partition Hierarchy (BPH) by altitude ordering relies on a total order on E, denoted by $\prec$. Let k in $\{1,\dots,\ell\}$, we denote by $u_k^\prec$ the k-th element of E for the order $\prec$. Let u be an edge in E, the *rank of* u *for* $\prec$, denoted by $r^\prec(u)$, is the unique integer k such that $u = u_k^\prec$. We set $\mathcal{B}[0] = \{\{\, x \,\} \mid x \in V\}$. The *partial binary partition hierarchy* $\mathcal{B}[k]$ *at rank* k is the hierarchy on V defined by $\mathcal{B}[k] = (\mathcal{B}[k-1] \setminus \{R_x, R_y\}) \cup \{R_x \cup R_y\}$ where $u_k^\prec = \{x, y\}$ and where R_x (resp. R_y) is the unique region of $\mathcal{B}[k-1]$ that contains x (resp. y). The partial binary partition hierarchy at rank ℓ is the *binary partition hierarchy by* $\prec$ and it is denoted by $\mathcal{B}^\prec$. The *rank* $r(R)$ of a region R of $B^\prec$ is the lowest scale λ at which the region appears in the hierarchy and, if $r(R) > 0$, the *building edge of* R, denoted by $\mu^\prec(R)$, is the edge $\mu^\prec(R) = u_{r(R)}^\prec$ (*i.e.*, the edge that lead to "building" region R).

2.2 Distribution of Binary Partition Hierarchy on a Causal Partition

Intuitively, distributing a hierarchy consists in splitting it into a set of smaller hierarchies such that: 1) each smaller hierarchy corresponds to a *selection* of a subpart of the whole that intersects a slice of the graph and 2) the initial hierarchy can be reconstructed by "gluing" those smaller hierarchies.

Let V be a set. The operation *sel* is the map from $2^{|V|} \times \mathcal{P}(V)$ to $\mathcal{P}(V)$ which associates to any subset X of V and to any partition $\mathbf{P}$ of V the subset $\mathrm{sel}(X, \mathbf{P})$ of $\mathbf{P}$ which contains every region of $\mathbf{P}$ that contains an element of X. The operation *select* is the map from $2^{|V|} \times \mathcal{H}_\ell(V)$ in $\mathcal{H}_\ell(V)$ which associates to any subset X of V and to any hierarchy $\mathcal{H}$ on V the hierarchy $\mathrm{select}\,(X, \mathcal{H}) = (\mathrm{sel}(X, \mathcal{H}[0]), \dots, \mathrm{sel}(X, \mathcal{H}[\ell]))$.

We are then able to define the distribution of a hierarchy thanks to the *select* operation. Let V be a set, let $\mathbf{P}$ be a complete partition on V and let $\mathcal{H}$ be a hierarchy on V. The distribution of $\mathcal{H}$ over $\mathbf{P}$ is the set $\delta_\mathcal{H} = \{\mathrm{select}\,(R, \mathcal{H}) \mid R \in \mathbf{P}\}$ and for any region R of $\mathbf{P}$, $\mathrm{select}\,(R, \mathcal{H})$ is called a *local hierarchy of* $\delta_\mathcal{H}$ In Figure 1, the distribution $(\mathcal{H}_0, \mathcal{H}_1, \mathcal{H}_2)$ of the BPH $\mathcal{H}$ is computed over the causal partition $(\{a, d\}, \{b, e\}, \{c, f\})$.

In the following sections, we consider the special case of a distribution built on a 4-adjacency graph representing a 2d image divided into slices (this is not a

limiting factor, and the method can easily be adapted to any regular grid graph such as 6 adjacency for 3d images). Let h and w be two integers representing the height and the width of an image. Then, the vertex set V is the Cartesian product $\{0, \cdots, h-1\} \times \{0, \cdots, w-1\}$ and the edge set E is given by the well-known 4-adjacency relation on V. Let k be a positive integer, the *causal partition of V into $k+1$ slices* is the sequence $(S_0, \ldots, S_k)$ such that for any t in $\{0, \cdots, k\}$, $S_t = \{(i,j) \in V \mid t \times \frac{w}{k} \leq i < (t+1) \times \frac{w}{k}\}$. Each element of the sequence $(S_0, \ldots, S_k)$ is called a *slice* and it can be seen that each slice can have up to two neighboring slices.

Important notation. *In this article, the symbols $G = (V, E)$, $\prec$, and $\mathcal{H}$ denote a 4-adjacency graph of ℓ edges, a total order on its edge set E, and the associated BPH by $\prec$, respectively. Furthermore, the symbol $\delta_{\mathcal{H}}$ denotes the distribution of $\mathcal{H}$ over the causal partition $(S_0, \cdots, S_k)$ of V into $k+1$ slices.*

2.3 Distribution of Attributes

An *attribute on $\mathcal{H}$* is a mapping A associating an attribute value $A(R)$ to every region R of $\mathcal{H}$. In this article, we consider attributes values that can be either Boolean or scalar and we are interested in computing attributes from the distribution of a BPH over a causal partition. To this end, given an attribute A on $\mathcal{H}$, we define the *distribution of A over $\delta_{\mathcal{H}}$* as the series $\delta_A = (A_{\mathcal{B}_0}, \cdots, A_{\mathcal{B}_k})$ such that for any i in $\{0, \ldots, k\}$ and any R in $\mathcal{B}_i$, we have $A(R) = A_{\mathcal{B}_i}(R)$. Hence, in this article, our main goal is to solve the following problem.

Problem. *Given the distribution $\delta_{\mathcal{H}}$ of the hierarchy $\mathcal{H}$, compute the distribution δ_A of an attribute A without explicitly computing $\mathcal{H}$ nor A while maintaining the out-of-core constraint, that is having a limited amount of memory at each computation steps.*

3 Propagate Algorithm: a Fundamental Brick for Out-of-core Attribute Computation

In this section, we introduce *Propagate* Algorithm which is called as a fundamental step of all subsequent attribute computations. Since a region may contain pixels belonging to several slices, it is generally not possible to compute the attribute of a region using only the information available in a single local hierarchy of the distribution. Thus, the general out-of-core attribute computation scheme considers a first step of local computation followed by a second step consisting of merging local information, done with Propagate Algorithm. More precisely, the proposed computation proceeds as follows:

1. Compute a *partial attribute* value locally on each tree of the distribution;
2. *Propagate* partial attribute values from neighboring trees in the causal direction, *i.e.*, from slices of lower indices to those of higher indices. At the end of this step, the attribute values of the last tree of the distribution are correct;

3. Backpropagate the "correct" attribute values in the anti-causal direction, *i.e.*, from slices of higher indices to those of lower indices. At the end of this step, all the attribute values of all the trees in the distribution are correct.

In this scheme, each local hierarchy of the distribution is visited twice (once in the causal pass and once in the anti-causal pass) and at any step, we never need to have more than 2 neighboring hierarchies simultaneously in memory.

Before describing precisely *Propagate* Algorithm in Section 3.2, we first introduce the necessary data structures in Section 3.1. Then, the computation of specific attributes using *Propagate* Algorithm is addressed in Section 4.

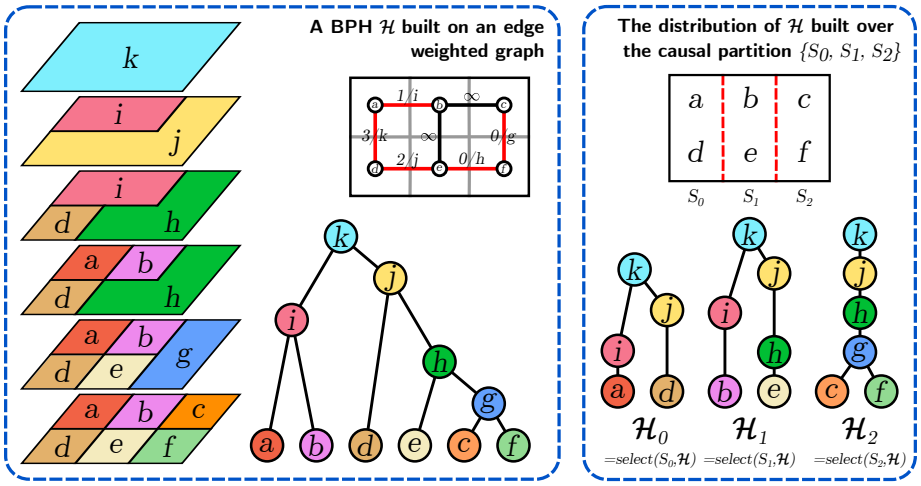

Fig. 1. Left, a BPH $\mathcal{H}$ built on an edge weighted graph (*index*/*weight*) and the visualisation of the nested series of partition. Red edges of the graph belong to the minimum spanning tree. Right, the distribution $\{\mathcal{H}_0, \mathcal{H}_1, \mathcal{H}_2\}$ of $\mathcal{H}$ over the causal partition $\{S_0, S_1, S_2\}$.

3.1 Data Structures

The data structures used in our algorithms are designed to contain only the necessary and sufficient information so that we never need to have all the data in the main memory at once. The tree-based data structure representing a local hierarchy $\mathcal{H}$ assumes that the regions, also called nodes of the hierarchy in this context, are indexed in a particular order and relies on three arrays: 1) a mapping of the indices from the local context (a given slice) to the global one (the whole graph) noted $\mathcal{H}$`.map`, 2) a parent array denoted by $\mathcal{H}$`.par` encoding the parent relation between the tree nodes, and 3) an array $\mathcal{H}$`.alt` giving, for each non-leaf-node of the tree, the weight of its corresponding building edge also called *altitude*.

More precisely, given a binary partition hierarchy $\mathcal{H}$ with n regions, every integer between 0 and $n-1$ is associated to a unique region of $\mathcal{H}$. Moreover, this indexing of the regions of $\mathcal{H}$ follows a topological order such that: 1) any leaf

region is indexed before any non-leaf region; 2) two leaf regions $\{x\}$ and $\{y\}$ are sorted with respect to an arbitrary order on the element V (*e.g.*, the raster scan order in 2d) and 3) two non-leaf regions are sorted according to their altitude, *i.e.*, the order of their building edges for $\prec$. This order can be seen as an extension of the order $\prec$ on E to the set $V \cup E$ that enables 1) to efficiently browse the nodes of a hierarchy according to their scale of appearance in the hierarchy and 2) to efficiently match regions of V with the leaves of the hierarchy. By abuse of notation, this extended order is also denoted by $\prec$ in the following.

To keep track of the global context, a link between the indices in the local tree and the global indices in the whole graph is stored in the form of an array `map` which associates: 1) to the index i of any leaf region R, the vertex x of the graph G such that $R = \{x\}$, *i.e.* `map[i]=x`; and 2) to the index i of any non-leaf region R, its building edge, *i.e.* `map[i]=`$\mu^{\prec}(R)$.

The parent relation of the tree is stored in an array `par` such that `par[i]=j` if the region of index j is the parent of the region of index i. For the root node r, which has no parent, we set `par[r]=r`.

The binary partition hierarchy is built for a particular ordering $\prec$ of the edges of G. In practice, this ordering is induced by weights computed over the edges of G. To this end, we store an array `alt` of size $|R^\star|$, *i.e.* the number of non-leaf regions, elements such that, for every region R in $R^\star$ of index i, `alt[i]` is the weight of the building edge $\mu^{\prec}(R)$ of region R, with $R^\star$ be the set of non-leaf regions R in $\mathcal{H}$. The edges can then be compared according to the following total order induced by the weights: we set $u \prec v$ if the weight of u is less than the one of v or if u and v have equal weights but u comes before v with respect to the raster scan order.

3.2 Attribute Propagation

Let us now give a precise description of Propagate Algorithm (see Algorithm 1) that is a fundamental brick to compute regional attributes in an out-of-core manner. This algorithm allows for transforming a first version of an attribute, which can be seen as local to each slice, to a global version. To this end, the algorithm considers the regions which are replicated in two neighbouring hierarchies associated with successive slices and propagate their attribute values: 1) firstly, in a causal order (lines 1 to 3), from the slices of lower indices to the ones of higher indices; and 2) secondly, in an anti-causal order (lines 5 to 6), from the higher indices to the lower ones. During the causal propagation, a binary operator is used to merge the attribute values of regions replicated in consecutive slices S_i and S_{i+1}. This binary operator (denoted by $\oplus$ in Algorithm 1) is given as a parameter of the algorithm. In the following sections, depending on the kind of attributes, we consider the sum, the maximum, or the minimum of two values. During the anti-causal propagation, the attribute value of the region in the slice of higher index is copied as the attribute value of the same replicated region in the slice of lower index.

Algorithm 1: PROPAGATE

Params: The distribution $(\mathcal{B}_0, \ldots, \mathcal{B}_k)\}$ of a BPH; a series $(\texttt{attr}_{\mathcal{B}_0}, \ldots, \texttt{attr}_{\mathcal{B}_k})$ of attributes; and a binary operator $\oplus$

Result: A new series $(\texttt{attr}^{\downarrow}_{\mathcal{B}_0}, \ldots, \texttt{attr}^{\downarrow}_{\mathcal{B}_k})$ of attributes where values of regions replicated on consecutive slices have been propagated.

1 $\texttt{attr}^{\uparrow}_{\mathcal{B}_0} := \texttt{attr}_{\mathcal{B}_0}$
2 **foreach** i from 1 to k **do**
3 $\quad \texttt{attr}^{\uparrow}_{\mathcal{B}_i} := \text{MERGE}(\mathcal{B}_{i-1}, \mathcal{B}_i, \texttt{attr}^{\uparrow}_{\mathcal{B}_{i-1}}, \texttt{attr}_{\mathcal{B}_i}, \oplus)$
4 $\texttt{attr}^{\downarrow}_{\mathcal{B}_k} := \texttt{attr}^{\uparrow}_{\mathcal{B}_k}$
5 **foreach** i from $k-1$ to 0 **do**
6 $\quad \texttt{attr}^{\downarrow}_{\mathcal{B}_i} := \text{MERGE}(\mathcal{B}_{i+1}, \mathcal{B}_i, \texttt{attr}^{\downarrow}_{\mathcal{B}_{i+1}}, \texttt{attr}^{\uparrow}_{\mathcal{B}_i}, \triangleleft)$
7 **return** $\left(\texttt{attr}^{\downarrow}_{\mathcal{B}_0}, \ldots, \texttt{attr}^{\downarrow}_{\mathcal{B}_k}\right)$

In order to identify the regions that are replicated in two consecutive hierarchies $\mathcal{X}$ and $\mathcal{Y}$, PROPAGATE Algorithm calls the auxiliary function MERGE (see Algorithm 2). This function simultaneously browses the regions of the two hierarchies in increasing order. When two nodes x and y of, respectively, $\mathcal{X}$ and $\mathcal{Y}$ are found as occurrences of the same region (*i.e.*, when $\mathcal{X}.\texttt{map}[x] = \mathcal{Y}.\texttt{map}[y]$, see line 3 in Algorithm 2) the attribute values $\texttt{attr}_{\mathcal{X}}[x]$ and $\texttt{attr}_{\mathcal{Y}}[y]$ are merged and the result is stored as the new attribute value of y: $\texttt{attr}'_{\mathcal{Y}}(y) := \texttt{attr}_{\mathcal{X}}(x) \oplus \texttt{attr}_{\mathcal{Y}}(y)$. If *assign-first* operator, denoted by $\triangleleft$, is given to Merge as the merging operator (*i.e.*, if $\oplus = \triangleleft$), then the result $\texttt{attr}_{\mathcal{X}}(x) \oplus \texttt{attr}_{\mathcal{Y}}(y)$ is simply the value $\texttt{attr}_{\mathcal{X}}(x)$.

Algorithm 2: MERGE

Params: Two hierarchies $\mathcal{X}$ and $\mathcal{Y}$; two attributes $\texttt{attr}_{\mathcal{X}}$ and $\texttt{attr}_{\mathcal{Y}}$ associated with these hierarchies ; and a binary operator $\oplus$

Result: A new attribute $\texttt{attr}'_{\mathcal{Y}}$, update of $\texttt{attr}_{\mathcal{Y}}$

1 $x := 0; y := 0$ // x iterates over $\mathcal{X}$ and y over $\mathcal{Y}$
2 **while** $x < |\mathcal{X}|$ **or** $y < |\mathcal{Y}|$ **do**
3 $\quad$ **if** $x < |\mathcal{X}|$ **and** $y < |\mathcal{Y}|$ **and** $\mathcal{X}.\texttt{map}[x] = \mathcal{Y}.\texttt{map}[y]$ **then**
4 $\quad\quad \texttt{attr}'_{\mathcal{Y}}[y] := \texttt{attr}_{\mathcal{X}}[x] \oplus \texttt{attr}_{\mathcal{Y}}[y]$
5 $\quad\quad x := x + 1;\ y := y + 1$
6 $\quad$ **else if** $\mathcal{X}.\texttt{map}[x] \prec \mathcal{Y}.\texttt{map}[y]$ **then** $x := x + 1$
7 $\quad$ **else** $y := y + 1$; $\texttt{attr}'_{\mathcal{Y}}[y] := \texttt{attr}_{\mathcal{Y}}[y]$
8 **return** $\texttt{attr}'_{\mathcal{Y}}$

Figure 2 illustrates the use of PROPAGATE Algorithm. More precisely, in Figure 2b, an initial attribute value is mapped to each node (subscript) of the local hierarchies (*i.e.*, $(attr_{\mathcal{B}_0}, attr_{\mathcal{B}_1}, attr_{\mathcal{B}_2})$). The values of $(attr^{\uparrow}_{\mathcal{B}_0}, attr^{\uparrow}_{\mathcal{B}_1}, attr^{\uparrow}_{\mathcal{B}_2})$ obtained after the causal pass of PROPAGATE are then shown in Figure 2c. In particular, red arrows indicate the mapping between replicated regions detected by MERGE as well as the computation which is made to obtain the value of the updated attribute (here $\oplus = +$ is given to MERGE). Finally, Figure 2d shows the result of $(attr^{\downarrow}_{\mathcal{B}_0}, attr^{\downarrow}_{\mathcal{B}_1}, attr^{\downarrow}_{\mathcal{B}_2})$ obtained after the anticausal pass.

It can be observed that every node of each tree is browsed once during the execution of MERGE Algorithm. Thus, the overall time-complexity of MERGE Algorithm is linear with respect to the number of regions of $\mathcal{X}$ and $\mathcal{Y}$. Furthermore, given a causal partition with $k+1$ slices, it can be seen that PROPAGATE performs exactly $2 \times k$ calls to MERGE and that the overall time complexity of PROPAGATE Algorithm is $O(k+N)$ where N is the sum of the numbers of regions of the local hierarchies $\mathcal{B}_0, \ldots, \mathcal{B}_k$.

4 Out-of-core Attributes Algorithms

In this section, following the general scheme proposed in Section 3, we present out-of-core algorithms to compute common attributes in hierarchical analysis.

Rightmost Slice. We first consider *Rightmost slice* attribute that maps to every region R of $\mathcal{H}$, the highest index of a slice containing a vertex of R. Aside from being a simplest attribute whose values can be computed with the help of PROPAGATE, *Rightmost slice* attribute, denoted by `right`, will be used subsequently as a preprocessing necessary before computing more complex attributes such as minima number. More precisely, it is used when one needs to select a single representative of a region that is replicated over several slices. In such case, we arbitrarily pick the representative of a region R as the occurrence of R in local hierarchy $\mathcal{B}_i$ such that $\texttt{right}_{\mathcal{B}_i}(R) = i$.

To compute the *Rightmost slice* of each region, firstly, for each local hierarchy $\mathcal{B}_i$ in the distribution $\delta_{\mathcal{H}}$, we locally initialize an attribute $I_{\mathcal{B}_i}(R) = i$ and then we call PROPAGATE with the supremum operator.

```
1 Function RightmostSlice(δ_H = {B_0, ..., B_k})
2     foreach i from 0 to k do
3         right_{B_i}[R] := i for each region R of B_i
4     return PROPAGATE((right_{B_0}, ..., right_{B_k}), δ_H, ∨)
```

Area. Let us now consider the area attribute that maps to every region of a hierarchy the number of pixels in that region. In the tree-based representation of a hierarchy, the area of a node n can be obtained recursively by setting $\texttt{area}(n) = 1$ for every leaf n and $\texttt{area}(n) = \sum_{c \in \texttt{children}(n)} \texttt{area}(c)$ for every non-leaf node n with $\texttt{children}(n) = \{c \mid n = \texttt{parent}(c)\}$. In the case of a distribution, for each local tree, we initialize the area of the nodes with the previous recursive formula (lines 2-5 below) followed by a call to PROPAGATE with the + operator.

```
1 Function Area(δ_H = {B_0, ..., B_k})
2     foreach i from 0 to k do
3         Initialize area values with 1 for every leaf and 0 for every
            non-leaf
4         foreach non-root region R of B_i in topological order do
5             area_{B_i}[parent[R]] += area_{B_i}[R]
6     return PROPAGATE((area_{B_0}, ..., area_{B_k}), δ_H, +)
```

Note that if we consider superpixels instead of pixels or the integral of a function over the domain, the algorithm remains valid provided an adaptation of its initialization at line 3.

Volume. In the tree-based representation of a hierarchy, the volume $V(n)$ of a node n can be obtained recursively by $V(n) = 0$ for every leaf n and $V(n) = A(n) \times \ |altitude(parent(n)) - altitude(n)\ | + \sum_{c \in children(n)} V(c)$ for every non-leaf node n. To compute the volume for a distribution, we firstly initialize the volume locally with the recursive formula. Similarly to the area, at this point, each region will lack the volumes of included regions that do not belong to the same local hierarchy. To correct this value, we propagate the partial volume by calling PROPAGATE with the + operator.

1 **Function** *Volume*($\delta_{\mathcal{H}} = \{\mathcal{B}_0, \ldots, \mathcal{B}_k\}$)
2 **foreach** i from 0 to k **do**
3 Locally initialize $\texttt{area}_{\mathcal{B}_i}$ *i.e.*, lines 3-5 of *Area*
4 **foreach** non-root region R of $\mathcal{B}_i$ in topological order **do**
5 $\texttt{vol}_{\mathcal{B}_i}[R] += \texttt{area}_{\mathcal{B}_i}[R] \times (\texttt{alt}[\mathcal{B}_i.\texttt{par}[R]] - \texttt{alt}[R])$
6 $\texttt{vol}_{\mathcal{B}_i}[\mathcal{B}_i.\texttt{par}[R]] += \texttt{vol}_{\mathcal{B}_i}[R]$
7 **return** PROPAGATE($(\texttt{vol}_{\mathcal{B}_0}, \ldots, \texttt{vol}_{\mathcal{B}_k})$, $\delta_{\mathcal{H}}$, +)

Height. The height of a region R of a BPH $\mathcal{H}$ is the difference between the altitude of the region R and the altitude of the lowest region included in R. From the tree based representation of a hierarchy, it can be computed with the following recursion: $H(n) = 0$ for every leaf n and $H(n) = altitude(parent(n)) - altitude(n) + \max\{H(c), c \in \texttt{children}(n)\}$ for every non-leaf node n. In a distribution, after initializing the height locally, following the above recursion, it is possible that the children leading to maximize the height in the global hierarchy do not belong to the same local hierarchy. It is therefore necessary to "search" for these children within the distribution in order to maximize the height. To do so we call *Propagate* with the supremum operator.

1 **Function** *Height*($\delta_{\mathcal{H}} = \{\mathcal{B}_0, \ldots, \mathcal{B}_k\}$)
2 **foreach** i from 0 to k **do**
3 Initialize $\texttt{heig}_{\mathcal{B}_i}$ to 0 for every node
4 **foreach** non-root region R of $\mathcal{B}_i$ in topological order **do**
5 $\texttt{heig}_{\mathcal{B}_i}[\mathcal{B}_i.\texttt{par}[R]] :=$
 $\max(\texttt{heig}_{\mathcal{B}_i}[\mathcal{B}_i.\texttt{par}[R]],\ \texttt{alt}[\mathcal{B}_i.\texttt{par}[\mathcal{B}_i.\texttt{par}[R]]] -$
 $\texttt{alt}_{\mathcal{B}_i}[\mathcal{B}_i.\texttt{par}[R]] + \texttt{heig}_{\mathcal{B}_i}[R])$
6 **return** PROPAGATE($\big(\texttt{heig}_{\mathcal{B}_0}, \ldots, \texttt{heig}_{\mathcal{B}_k}\big)$, $\delta_{\mathcal{H}}$, $\vee$)

Note that this method can also be used to compute the topological height, *i.e.* the maximum length of a paths from a node to a descendant leaf node.

Minima. A regional minimum of a weighted graph is a set of vertices connected by edges of weight k and whose adjacent edges are all of strictly higher weight. In the tree-based representation of a BPH, a node n can be identified as a minimum if 1) the altitude of the parent of n is different from the one of n, and 2) the altitude of any non-leaf node included in the sub-tree rooted in n is

equal to the altitude of n. We can formalize and test these two criteria in our framework. Firstly, let $L_{\mathcal{B}_i}(n) =$ true if $altitude(n) < altitude(parent(n))$ and false otherwise: this can be calculated locally, as only the altitude of the parents is required. The second criterion is defined recursively as $E^L_{\mathcal{B}_i}(R) = false$ for any leaf R and as $E^L_{\mathcal{B}_i}(R) = \bigwedge_{c \in children(R)} \neg L_{\mathcal{B}_i}(c)$ for an any non-leaf region R. This last criterion depends on the children of R, so we need to make sure it is satisfied for all children in the distribution. We can verify this by calling PROPAGATE with the infimum operator to compute a *global* criterion $E^G_{\mathcal{B}_i}$. Then, an occurrence of a region R in the distribution is a minimum if $M_{\mathcal{B}_i}(R) = L_{\mathcal{B}_i}(R) \wedge E^G_{\mathcal{B}_i}(R)$.

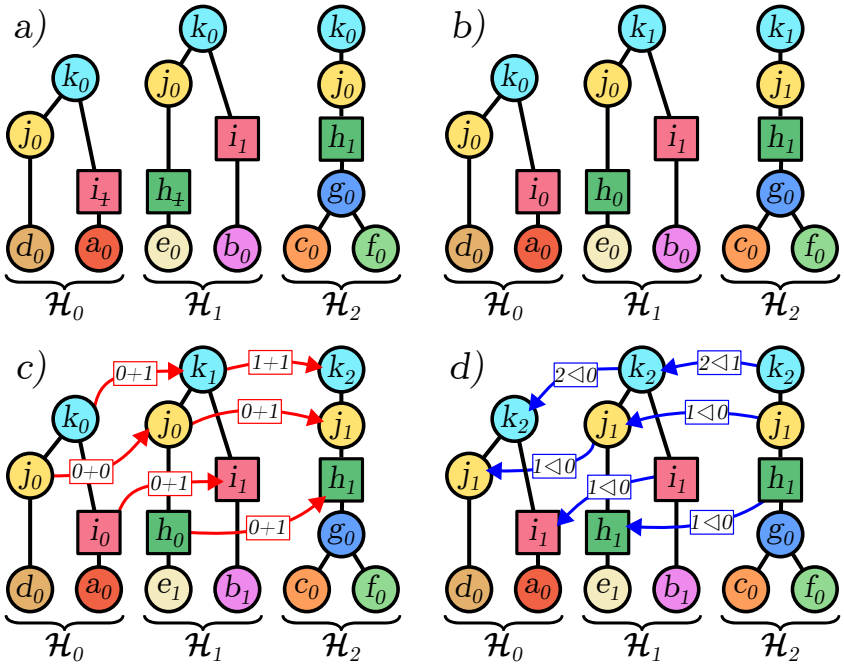

Fig. 2. Minima number attribute computation. The subscript of each node indicate its attribute value. **a.** The two local minima i and h (square node) have both two occurrences in the distribution. To ensure that only one occurrence of each minimum contributes, only the rightmost replica is set to 1, *e.g.*, region h in $\mathcal{H}_1$ count as 0 because its rightmost occurrence is in $\mathcal{H}_2$. **b.** Local initialisation of `minnb` on the basis of modified minima values. **c.** Causal pass with the operator $+$. The arrows link two occurrences of a region in neighbouring hierarchies. The arrow points to the region whose attribute is updated. **d.** Anti-causal pass, the values associated with rightmost occurrences of each region area back-propagated with $\triangleleft$.

Minima Number. In this part, we are interested in the number of minima included in each region of a hierarchy. In a tree, we can obtain it recursively by considering, $M(n) = 0$ for every leaf n, $M(n) = 1$ for every minimum n and $M(n) = \sum_{x \in \texttt{childre}(n)} M(n)$ for any other node.

At first glance, the calculation of this attribute seems similar to that of area. For the latter, we propagate the sum value of the leaves, and no leaf can

```
1 Function Minima(δ_H = {B_0, ..., B_k})
2   foreach i from 0 to k do
3     foreach region R of B_i in topological order do
4       includesMin[R] := false; mini_{B_i}[R] := false
5     foreach non-leaf region R of B_i in topological order do
6       if alt[R] < alt[B_i.par[R]] and includesMin[R] = false then
7         mini_{B_i}[R] := true; includesMin[R] := true
8       includesMin[B_i.par[R]] := includesMin[R]
9   return PROPAGATE((mini_{B_0}, ..., mini_{B_k}), δ_H, ∧)
```

contribute more than once to the final result because a leaf belongs to one and only one local hierarchy. However, as the minima are defined on non-leaf regions, they can be replicated in several local hierarchies *e.g.*, on Figure 2 the node h is replicated in $\mathcal{H}_1$ and $\mathcal{H}_2$.If this situation, if we use the same strategy as for area, a single minimum would contribute several times to the minima count. We must therefore ensure that, one and only one occurrence of a minimum contributes to the calculation of the attribute for the ancestors of thus minimum. As a solution, we use *RightmostSlice* attribute to arbitrarily discriminate the last occurrence of each region. We can therefore guarantee that only one of these occurrences will contribute to the local minimum account. In Figure 2 a), since the occurrence of the minimum h in $\mathcal{H}_1$ is not the rightmost, it does not contribute to the bottom-up sum contrary to the occurrence in $\mathcal{H}_2$. It should be noted that contrary to the previously introduced attributes, this one necessitate two calls to *Propagate* (minima and *RightmostSlice* can be computed simultaneously) rather than one.

```
1 Function NumberMinima(δ_H = {B_0, ..., B_k})
2   right := RightmostSlice(δ_H); mini := Minima(δ_H)
3   foreach i from 0 to k do
4     Initialize nbmin_{B_i} to 0 for every node
5     foreach non-root region R of the each B_i in topological order do
6       if mini_{B_i}[R] = true and right_{B_i}[R] = i then
7         nbmin_{B_i}[R] := 1
8       nbmin_{B_i}[B_i.par[R]] += nbmin_{B_i}[R]
9   return PROPAGATE((nbmin_{B_0}, ..., nbmin_{B_k}), δ_H, +)
```

5 Discussion and conclusion

In this paper, we proposed a global scheme for computing attributes for a distribution of BPH. This method is based on a linear complexity algorithm and requires having only the information about two adjacent regions in main memory at any step of the algorithm. We have given applications for computing common attributes such as area, volume, and local minima. It should be noted that the use of this methodology is not limited to the presented attributes, but can easily be used to compute other attributes such as bounding boxes or to determine watershed edges. In future work, we plan to study the time and memory

consumption of this methodology and to extend it to the computation of more complex attributes such as extinction values or smallest common ancestors.

References

1. Breen, E.J., Jones, R.: Attribute openings, thinnings, and granulometries. Computer Vision and Image Understanding **64**(3), 377–389 (1996)
2. Cousty, J., Najman, L., Perret, B.: Constructive links between some morphological hierarchies on edge-weighted graphs. In: ISMM. pp. 86–97 (2013)
3. Cousty, J., Perret, B., Phelippeau, H., Carneiro, S., Kamlay, P., Buzer, L.: An algebraic framework for out-of-core hierarchical segmentation algorithms. In: DGMM. pp. 378–390 (2021)
4. Gazagnes, S., Wilkinson, M.H.F.: Distributed connected component filtering and analysis in 2d and 3d tera-scale data sets. IEEE TIP (2021)
5. Gazagnes, S., Wilkinson, M.H.: Parallel attribute computation for distributed component forests. In: 2022 IEEE ICIP. pp. 601–605 (2022)
6. Gigli, L., Velasco-Forero, S., Marcotegui, B.: On minimum spanning tree streaming for hierarchical segmentation. PRL **138**, 155–162 (2020)
7. Götz, M., Cavallaro, G., Geraud, T., Book, M., Riedel, M.: Parallel computation of component trees on distributed memory machines. TPDS (2018)
8. Havel, J., Merciol, F., Lefèvre, S.: Efficient tree construction for multiscale image representation and processing. JRTIP **16** (2019)
9. Kazemier, J.J., Ouzounis, G.K., Wilkinson, M.H.: Connected morphological attribute filters on distributed memory parallel machines. In: ISMM. pp. 357–368 (2017)
10. Lebon, Q., Lefèvre, J., Cousty, J., Perret, B.: Interactive segmentation with incremental watershed cuts. In: CIARP. pp. 189–200 (2024)
11. Lefèvre, J., Cousty, J., Perret, B., Phelippeau, H.: Join, select, and insert: Efficient out-of-core algorithms for hierarchical segmentation trees. In: DGMM. pp. 274–286 (2022)
12. Meyer, F.: The dynamics of minima and contours. In: ISMM. pp. 329–336 (1996)
13. Meyer, F., Maragos, P.: Morphological scale-space representation with levelings. In: International Conference on Scale-Space Theories in Computer Vision. pp. 187–198 (1999)
14. Najman, L., Cousty, J., Perret, B.: Playing with Kruskal: algorithms for morphological trees in edge-weighted graphs. In: ISMM. pp. 135–146 (2013)
15. Passat, N., Kurtz, C., Vacavant, A.: Virtual special issue: "hierarchical representations: New results and challenges for image analysis". PRL **138**, 201–203 (2020)
16. Perret, B., Chierchia, G., Cousty, J., Guimarães, S.J.F., Kenmochi, Y., Najman, L.: Higra: Hierarchical graph analysis. SoftwareX **10**, 100335 (2019)
17. Salembier, P., Serra, J.: Flat zones filtering, connected operators, and filters by reconstruction. IEEE TIP (8), 1153–1160 (1995)
18. Salembier, P., Garrido, L.: Binary partition tree as an efficient representation for image processing, segmentation, and information retrieval. TIP **9**(4), 561–576 (2000)
19. Silva, A.G., de Alencar Lotufo, R.: Efficient computation of new extinction values from extended component tree. PRL **32**(1), 79–90 (2011)
20. Silva, D.J., Alves, W.A., Hashimoto, R.F.: Incremental bit-quads count in component trees: Theory, algorithms, and optimization. PRL **129**, 33–40 (2020)