Out-of-core hierarchical segmentation algorithm

First Authors[a,b,*], Second Auth[a,], Other Authors[b]

[a] *Université Paris-Est, LIGM, CNRS - ENPC - ESIEE Paris -UPEM, 2 Boulevard Blaise Pascal, Noisy-le-Grand 93160, France*
[b] *Une autre universit, quelque part dans l'univers, probablement multituetelle*

ABSTRACT

Hierarchies are cool. Out of core is mandatory for nowday images in the laboratory and the sky. Thus, we do out-of-core algorithms for hierarchy processing.

Algorithm 1: Out-of-core BPT

Data: $G = (V, E, w)$: an edge-weighted graph;
$\{S_0, \ldots, S_\ell\}$: a partition of V into slices ;
$\partial_{(0,1)}, \ldots, \partial_{(\ell-1,\ell)}$: the sets of edges such that, for any $i \in \{1, \ldots, \ell\}$, the set $\partial_{(i-1,i)}$ contains any edge in E that links an element of S_{i-1} to an element of S_i ;
Result: $(\mathcal{T}_0, \ldots, \mathcal{T}_\ell)$: a series of trees such that, for any i in $\{0, \ldots, \ell\}$, the tree $\mathcal{T}_i$ represents the the restriction of the binary partition tree of (V, E, w) to S_i

```
/* Causal traversal of the slices                                                    */
```
$\mathcal{T}_0 :=$ BPTAO(G_{S_0}) ; `/* call PlayingWithKruskal on the subraph of` G `induced by the slice` S_0 `*/`
$\mathcal{R}_0 :=$ select $\cup\partial_{(0,1)}$ from $\mathcal{T}_0$; `/* right border of` $\mathcal{T}_0$ `*/`
foreach *i from* 1 *to* ℓ **do**
  $\mathcal{T}_i :=$ BPTAO(G_{S_i}) ; `/* call PlayingWithKruskal on the subraph of` G `induced by` S_i `*/`
  $\mathcal{L}_i :=$ select $\cup\partial_{(i-1,i)}$ from $\mathcal{T}_i$; `/* left border of` $\mathcal{T}_i$ `*/`
  $\mathcal{M}_{(i-1,i)} :=$ join $\mathcal{R}_{i-1}$ and $\mathcal{L}_i$ through $\partial_{(i-1,i)}$; `/* build the left join of` $\mathcal{T}_i$ `*/`
  $\mathcal{TEMP} :=$ select S_i from $\mathcal{M}_{(i-1,i)}$; `/* part of the left join containing elements of` S_i `*/`
  $\mathcal{T}_i :=$ insert $\mathcal{TEMP}$ in $\mathcal{T}_i$; `/* update` $\mathcal{T}_i$ `according to its left join */`
  if $i < \ell$ **then** $\mathcal{R}_i :=$ select $\cup\partial_{(i,i+1)}$ from $\mathcal{T}_i$; `/* prepare the right border of` $\mathcal{T}_i$ `if needed */`
foreach *i from* $\ell - 1$ *to 0* **do** `/* Anticausal traversal of the slices */`
  $\mathcal{TEMP} :=$ select S_i from $\mathcal{M}_{(i,i+1)}$; `/* part of the right join containing elements of` S_i `*/`
  $\mathcal{T}_i :=$ insert $\mathcal{TEMP}$ in $\mathcal{T}_i$; `/* update` $\mathcal{T}_i$ `according to its right join */`
  if $i > 0$ **then** `/* prepare the left join of` $\mathcal{T}_i$`, if needed */`
    $\mathcal{TEMP} :=$ select $\cup\partial_{(i-1,i)}$ from $\mathcal{T}_i$; `/* Left border of` $\mathcal{T}_i$ `*/`
    $\mathcal{M}_{(i-1,i)} :=$ insert $\mathcal{TEMP}$ in $\mathcal{M}_{(i-1,i)}$; `/* update the left join wrt the left border of` $\mathcal{T}_i$ `*/`

*Corresponding author: Tel.: +51-54-233272
e-mail: `first.author@esiee.fr` (First Authors)

Algorithm 2: Select - from

Data: L: a set of vertices of the graph (V, E);
$\mathcal{T}$ a tree whose leaves are vertices of the graph $V(E)$;
Result: S: the subtree of $\mathcal{T}$ whose nodes are the ancestors of an element in L

```
1  i = 0 ; j =0;
2  while j < T.nbLeaves and i < |L| do
3      if L[i] ≺ T[j] then i++;
4      else if T[j] ≺ L[i] then j++;
5      else
6          mark(T[j]); mark(T.parent[j]) ;
7          i++; j++
8  foreach i from T.nbLeaves to T.nbNodes-1 do
9      if T[i] is marked then mark(T.parent[i]);
10 S := marked nodes of T
```

Algorithm 3: join-through

Data: $\mathcal{T}_1$ and $\mathcal{T}_2$: the trees that must be joined;
$\partial = \{\{x_1, y_1\}, \ldots, \{x_n, y_n\}\}$: the border along which we want to merge, given as a set of edges that link leaves of $\mathcal{T}_1$ with leaves of $\mathcal{T}_2$. We assume that x_i lies on $\mathcal{T}_1$ and that y_i lies on $\mathcal{T}_2$;
Result: $\mathcal{M}$: the merge tree of the given data

```
1  ∂ := sort(∂);
2  M := T1 ⊔ T2;
3  foreach i from 0 to n − 1 do
4      (x, y) := ∂[i];
5      n1 := InfAncestor(x, M, (x, y)); n2 := InfAncestor(y, M, (x, y));
6      if n1 ≠ n2 then
7          z = M.size ; M[z] := (x,y); M ++;
8          p1 := M.parent[n1]; p2 := M.parent[n2];
9          M.parent[n1] := z; M.parent[n2] := z ;
10         if p1 ≠ nil then nLostChildren[p1] ++;
11         if p2 ≠ nil then nLostChildre[p2] ++;
12         do
13             if p2 ≺ p1 then swap(p1, p2);
14             M.parent[z] := p1 ; nLostChildren[p1] −−;
15             z := p1 ;
16             p1 := M.parent[p1];
17             if p0 ≠ p1 and p1 ≠ nil then nLostChildren[p1] ++;
18         while p1 ≠ p2;
19 RemoveNonBranchNodes(M);
20 Sort(M) ;                    /* sort the nodes of M according to the altitude ordering ≺ */
   /* Sorting can be done in linear time because M is partially sorted as it is built from the
      concatenation of sorted nodes                                                          */
```

Algorithm 4: Function InfAncestor

Data: $\mathcal{T}$: a tree ; x: a leaf node of $\mathcal{T}$; u: an edge of the graph space;
Result: n: the node of highest altitude among the ancestors of x in $\mathcal{T}$ with an altitude less than the one of u

```
1  n := x;
2  while M.parent[n] ≺ u do n := M.parent(n);
```

Algorithm 5: Procedure RemoveNonBranchNodes

Data: $\mathcal{T}$: a tree ;
Result: $\mathcal{T}$: a filtered version of the input tree where the branch nodes (*i.e.*, the non-leaf nodes with a single child) have been removed.

foreach *non-leaf node* n *of* $\mathcal{T}$ **do**
  if *nLostChildren[n] = 0* **then**
    $y := \mathcal{T}$.parent(n);
    while $y \neq$ *nil and nLostChildren[y]* < 0 **do**
      nLostChildren[y] := -2;
      $z := \mathcal{T}$.parent[y];
      $\mathcal{T}$.parent[y] := -2; `/* mark y as removed */`
      $y := z$;
    $\mathcal{T}$.parent[n] := y;

Algorithm 6: insert-in

Data: $\mathcal{T}$: a tree to update ;
$\mathcal{U}$: a tree whose leaves are also leaves of $\mathcal{T}$;
Result: $\mathcal{V}$: a tree whose node set is the union of the node sets of $\mathcal{T}$ and of $\mathcal{U}$ and such that the parent of a node x in $\mathcal{V}$ is the same as the one of x in $\mathcal{U}$ if x is a node of $\mathcal{U}$ or is the same as the parent of x in $\mathcal{T}$ otherwise.

`/* the nodes of` $\mathcal{T}$ `and` $\mathcal{U}$ `are supposed to be already sorted according to` $\prec$ `*/`
$t := 0$; $u := 0$; r :=0;
foreach *leaf* n *of* $\mathcal{T}$ **do** markAlive($\mathcal{T}.parent[\mathcal{T}.[t]]$);
while $t < \mathcal{T}.nbNodes$ *OR* $u < \mathcal{U}.nbNodes$ **do**
  if $\mathcal{T}[t] = \mathcal{U}[u]$ **then**
    $R_T[t]$:= r ; $R_U[u]$:= r ;
    Nodes[r] := (u, t) ;
    $t++$; $u++$;
  else if $\mathcal{T}[t] \prec \mathcal{U}[u]$ **then**
    if $\mathcal{T}[t]$ *is marked Alive* **then**
      markAlive($\mathcal{T}.parent[\mathcal{T}[t]]$);
      $R_T[t]$:= r;
      Nodes[r] := (nil, t);
    else $r--$;
    $t++$
  else
    $R_U[u] := r$;
    Nodes[r] := (u, nil);
    $u++$;
  r++;
$\mathcal{V}$:= init a tree structure for r nodes
foreach i *from 0 to* r **do**
  $(u, t) := Nodes[i]$;
  if $u \neq nil$ **then**
    $\mathcal{V}[r] := \mathcal{U}[u]$;
    if $\mathcal{U}.parent[u] = nil$ **then** $\mathcal{V}.parent[r] := nil$;
    else $\mathcal{V}.parent[r] := R_U[\mathcal{U}.parent[u]]$;
  else
    $\mathcal{V}[r] := \mathcal{T}[t]$;
    if $\mathcal{T}.parent[t] = nil$ **then** $\mathcal{V}.parent[r] := nil$;
    else $\mathcal{V}.parent[r] := R_T[\mathcal{T}.parent[t]]$;

1. Formalisms

1.1. Distributed hierarchies of partitions: select-from operator

Let V be a set. A *(partial) 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 *support (or ground)* of a partition $\mathbf{P}$, denoted by $gr(\mathbf{P})$, is the union of the regions of $\mathbf{P}$. A partition whose support is V is called a *complete partition of* V . A *hierarchy on* V is a series $(\mathbf{P}_0, \ldots, \mathbf{P}_\ell)$ of partitions of V such that, for any λ in $\{0, \ldots, \ell - 1\}$, every element of $\mathbf{P}_\lambda$ is included in an element of $\mathbf{P}_{\lambda+1}$. Let $\mathcal{H} = (\mathbf{P}_0, \ldots, \mathbf{P}_\ell)$ be a hierarchy. The integer ℓ is called the depth of $\mathcal{H}$ and, for any λ in $\{0, \ldots, \ell\}$, the partition $\mathbf{P}_\lambda$ is called the λ-scale of $\mathcal{H}$. In the following, if λ is an integer in $\{0, \ldots \ell\}$, we denote by $\mathcal{H}[\lambda]$ the λ-scale of $\mathcal{H}$. The hierarchy $\mathcal{H}$ is *complete* if $\mathcal{H}[0] = \{\{x\} \mid x \in V\}$ and if $\mathcal{H}[\ell] = \{V\}$. The hierarchy $\mathcal{H}$ is *binary* if, for any $\lambda \in \{1, \ldots, \ell\}$, we have $|\mathbf{P}_\lambda| \in \{|\mathbf{P}_{\lambda-1}| - 1, |\mathbf{P}_{\lambda|-1}\}$. We denote by $\mathcal{H}_\ell(V)$ the set of all hierarchies on V of depth ℓ, by $\mathbf{P}(V)$ 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 positive integer.

Let V be a set. We define the operator *select* as the map from $2^{|V|} \times \mathbf{P}(V)$ in $P(V)$ which associates to any subset X of V and to any partition $\mathbf{P}$ of V the subset of $\mathbf{P}$ which contains every region of $\mathbf{P}$ that contains an element of X. In the following, for the sake of clarity, we sometimes write *select* X *from* $\mathbf{P}$ instead of $select(X, \mathbf{P})$.

Let V be a set. We define the operator *Select* as 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 $(select\ X\ from\ \mathcal{H}[0], \ldots, select\ X\ from \mathcal{H}[\ell])$. If $\mathcal{H}$ is a hierarchy, in the following, we also write *Select* X *from* $\mathcal{H}$ instead of $\text{Select}(X, \mathcal{H})$.

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 $\{\text{Select}(R, \mathcal{H}) \mid R \in \mathbf{P}\}$.

Property 1 (reconstruction). *Let V be a set, let $\mathbf{P}$ be a complete partition on V, let $\mathcal{H}$ be a hierarchy on V, and let δ be the distribution of $\mathcal{H}$ over $\mathbf{P}$. Then, for any λ in $\{0, \ldots, \ell\}$, we have $\mathcal{H}[\lambda] = \cup\{\mathcal{D}[\lambda] \mid \mathcal{D} \in \delta\}$.*

Any hierarchy of partitions can be represented by a tree data structure. This representation is often used in algorithms that handle hierarchies.

Let $\mathcal{H}$ be a hierarchy in $\mathcal{H}_\ell$. Any region that belong to a partition of $\mathcal{H}$ is called a *region of* $\mathcal{H}$. We denote by $\mathcal{R}(\mathcal{H})$ the set of regions of $\mathcal{H}$, *i.e.*, $\mathcal{R}(\mathcal{H}) = \cup\{\mathbf{H}[\lambda] \mid \lambda \in \{0, \ldots, \ell\}$.

Let $\mathcal{H}$ be a hierarchy and let X and Y be two regions of $\mathcal{H}$. We say that X is an ancestor of Y if $Y \subseteq X$. The set of ancestors of a given region is totally ordered by the inclusion relation. If X is the smallest proper ancestor of Y, we

say that X is the *parent of* Y and we write $par(Y) = X$. If Y has no parent, then we write $par(Y) = nil$. The set of all regions of $\mathcal{H}$ equipped with the mapping par is a directed forest. Furthermore, if $\mathcal{H}$ is a complete hierarchy, then the pair $(\mathcal{R}(\mathcal{H}), par)$ is a directed tree, called the *component tree of* $\mathcal{H}$. A region of $\mathcal{H}$ which is not the parent of any region of $\mathcal{H}$ is called a *leaf-region* whereas any region which is the parent of a region is called a *non-leaf region*. We denote by $\mathcal{R}^\star(\mathcal{H})$ the set of non-leaf regions of $\mathcal{H}$

1.2. Binary partition hierarchies

The binary partition hierarchy plays a central role when one has to deal with hierarchies, notably in the framework of mathematical morphology. It allows one to obtain the set representation of a hierarchy defined from its saliency map or ultrametric distance representations. This hierarchy can also be easily post-processed to obtain other hierarchies useful in image analysis applications like the quasi-flat zones hierarchy, the constrained connectivity hierarchies, the hierarchical watersheds, HGB hierarchy or to obtain marker based image segmentations of an image. In this section, we provide a new definition of this hierarchy based on an (elementary) external binary operation on hierarchies. We investigate the algebraic properties of this operation, including commutativity. As shown in a forthcoming section, this allows us to design calculus (with associated algorithms) to obtain distributions of binary partition hierarchies.

The binary partition hierarchy depends on an ordering of the edges of a graph that structures the set on which the hierarchy is build. Let us start this section with basic notions on graphs.

We define a graph as a pair $X = (V, E)$ where V is a finite set and E is composed of unordered pairs of distinct elements in V, *i.e.*, E is a subset of $\{\{x, y\} \subseteq \mathrm{V} \mid x \neq y\}$. Each element of V is called a *vertex* of X, and each element of E is called an edge of X. Let X be a graph. We denote by $V(X)$ (resp. $E(X)$), the set of every vertex (resp. edge) of X.

In the remaining part of this article, we assume that $G = (V, E)$ is a graph and that $\ell = |E|$.

We now give the definitions of some basic operators on graphs (see RefGraphMorpho for their mathematical morphology properties) that are used for presenting several orations on hierarchies in the sequel.

We denote by $\epsilon^\times$ the operator that maps to any subset X of V the subset of E made of the edges of G composed of two vertices in X, *i.e.* $\epsilon^\times(X) = \{\{x, y\} \in E \mid x \in X, y \in X\}$. We denote by $\delta^\bullet$ the operator that maps any subset F of E to the subset of V made of every vertex of V that belongs to an edge in F, *i.e.*, $\delta^\bullet(F) = \cup F$. We denote by $\delta^\times$ the operator that maps to any subset X of V the subset of E made of the edges of G that contains a vertex of X, *i.e.*, $\delta^\times(X) = \{\{x, y\} \in E \mid \{x, y\} \cap X \neq \emptyset\}$. Finally, we denote by γ the operators that maps to any two subsets X and Y of V the subset of E that contains every edge of E made of one vertex in X and one vertex in Y, *i.e.*,

$\gamma(X,Y) = \epsilon^{\times}(X \cup Y) \setminus \epsilon(X) \setminus \epsilon(Y)$.

A *total order on* E is a sequence $(u_1, \ldots, u_\ell)$ such that $\{u_1, \ldots, u_\ell\} = E$. Let $\prec$ be a total order on E. Let k in $\{1, \ldots, \ell\}$, we denote by $u_k^{\prec}$ the k-the 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 $u = u_k^{\prec}$. Let v be an edge of G. We write $u \prec v$ if the rank of u is less than the one of v , *i.e.*, $r^{\prec}(u) < r^{\prec}(v)$.

In the remaining part of this article, the symbol $\prec$ denotes any total order on E.

Let $\mathbf{P}$ be a partition of V and x be any element in V. We set $\phi_{\mathbf{P}}(x) = \emptyset$ if x does not belong to the support of $\mathbf{P}$ and, otherwise, we set $\phi_{\mathbf{P}}(x) = R$ where R is the unique region of $\mathbf{P}$ which contains x.

Let $\mathcal{X}$ be a hierarchy on V. Let $\{x, y\}$ be an edge in E and let k be the the rank of $\{x, y\}$ for $\prec$. The *update of* $\mathcal{X}$ *with respect to* $\{x, y\}$, denoted by $\mathcal{X} \oplus \{x, y\}$, is the hierarchy defined by

- $\mathcal{X} \oplus \{x, y\}[\lambda] = \mathcal{X}[\lambda]$, for any λ in $\{0, \ldots, k-1\}$; and
- for any λ in $\{k, \ldots, \ell\}$, $\mathcal{X} \oplus \{x, y\} = \mathcal{X}[\lambda] \setminus \{R_x, R_y\} \cup \{R_x \cup R_y\}$ where $R_x = \phi_x(\mathcal{H}[\lambda])$ and $R_y = \phi_y(\mathcal{H}[\lambda]))$.

Property 2. *Let* u *and* v *be two edges and let* $\mathcal{H}$ *be a hierarchy. The following statements hold true:*

1. $\mathcal{H} \oplus u = \mathcal{H} \oplus u \oplus u$
2. $\mathcal{H} \oplus u \oplus v = \mathcal{H} \oplus v \oplus u$

Let $E' \subseteq E$ and let $\mathcal{H}$ be a hierarchy, we set $\mathcal{H} \boxplus E' = \mathcal{H} \oplus u_1 \oplus \ldots \oplus u_{|E'|}$ where $E' = \{u_1, \ldots, u_{|E'|}\}$.

Let X be a set, we denote by $\perp_X$ the hierarchy defined by $\perp_X [\lambda] = \{\{x\} \mid x \in X\}$, for any λ in $\{0, \ldots \ell\}$.

Definition 3. *Let* $\prec$ *be an ordering on* E*. The* binary partition hierarchy for $\prec$*, denoted by* $\mathcal{B}^{\prec}$ *is the hierarchy* $\perp_V \boxplus E$*. Let* A *be a subset of* V*, we define the* binary partition hierarchy for $\prec$ on A*, denoted by* $\mathcal{B}_{\mathcal{A}}^{\prec}$*, as the hierarchy* $\perp_A \boxplus \epsilon^{\times}(A)$.[1]

Property 4. *Let* $\prec$ *be an ordering. The binary partition hierarchy for* $\prec$ *is complete and binary.*

In this article, we consider the problem of computing the distribution of the binary partition hierarchy over a partition of the space under the out-of-core constraint, that is minimizing the size of the data structures loaded simultaneously in memory during the construction. To this end, we finish this section by analyzing some algebraic properties of the external operation $\boxplus$, together with a nice characterization that relies on the supremum of hierarchies. These properties will then be used in forthcoming sections to design distributed computations of the binary partition hierarchies.

[1] unnecessary text : We should be consistent in notation, that is either write $\boxplus^{\prec}$ or remove the $^{\prec}$ superscript everywhere.

Let $\mathbf{X}$ and $\mathbf{Y}$ be two partitions of V. We say that $\mathbf{Y}$ is a refinement of $\mathbf{X}$ if any region of $\mathbf{Y}$ is included in a region of $\mathbf{X}$. Let $\mathcal{X}$ and $\mathcal{Y}$ be two hierarchies in $\mathcal{H}_\ell(V)$. We say that $\mathcal{Y}$ is smaller than $\mathcal{X}$ if, for any λ in $\{0,\dots,\ell\}$, the partition $\mathcal{Y}[\lambda]$ is a refinement of $\mathcal{X}[\lambda]$. If $\mathcal{Y}$ is smaller than $\mathcal{X}$, we write $\mathcal{Y} \sqsubseteq \mathcal{X}$. The set $\mathcal{H}_\ell(V)$ equipped with the relation $\sqsubseteq$ is a lattice whose supremum and infimum are denoted by $\sqcup$ and $\sqcap$ respectively [2].

Let $u = \{x, y\}$ be an edge of E, we denote by $\mathcal{H}_u$ the hierarchy such that:

- for any λ in $\{0,\dots r(u)-1\}$, $\mathcal{H}_u[\lambda] = \{\{x\},\{y\}\}$; and
- for any λ in $\{r(u),\dots,\ell\}$, $\mathcal{H}_u[\lambda] = \{u\}$.

Property 5. *Let $\mathcal{X}$ and $\mathcal{Y}$ be two hierarchies in $\mathcal{H}_\ell(V)$, let u be an edge in E, and let F, G be two subsets of E. Then, the following equalities hold true:*

1. $\mathcal{H} \oplus u = \mathcal{H} \sqcup \mathcal{H}_u$;
2. $\mathcal{H} \boxplus F = \mathcal{H} \sqcup (\sqcup\{\mathcal{H}_u \mid u \in F\})$;
3. $(\mathcal{X} \sqcup \mathcal{Y}) \boxplus F = \mathcal{X} \sqcup (\mathcal{Y} \boxplus F) = (\mathcal{X} \boxplus F) \sqcup \mathcal{Y}$;
4. $\mathcal{X} \boxplus (F \cup I) = (\mathcal{X} \boxplus F) \boxplus I = (\mathcal{X} \boxplus I) \boxplus F$; *and*
5. $(\mathcal{X} \sqcap \mathcal{Y}) \boxplus F = (\mathcal{X} \boxplus F) \sqcap (\mathcal{Y} \boxplus F)$.

Corollary 6. *Let X be a subset of V. Then, $\mathcal{B}^\prec_X = (\sqcup\{\mathcal{H}_u \mid u \in \epsilon^\times(X)\}) \sqcup \perp_X$.*

1.3. Minimum spanning trees

As recalled in this section, the binary partition hierarchy is deeply linked to minimum spanning trees.

Let R be a region of $\mathcal{B}^\prec$. The *rank of* R, denoted by $r(R)$, is the lowest integer λ such that R is a region of $\mathcal{B}^\prec[\lambda]$. We consider the map μ from $\mathcal{R}^\star(\mathcal{H})$ in E such that, for any non-leaf region R of $\mathcal{B}^\prec$, we have $\mu^\prec(R) = u^\prec_{r(R)}$. We say that $\mu^\prec(R)$ is the *building edge of* R.

Given the graph G and an ordering $\prec$ of E, the algorithm presented in [**?**] computes the component tree of the binary partition hierarchy by $\prec$ as well as the building edge $\mu^\prec(R)$ associated to any non-leaf region R of the hierarchy. This result is deeply linked to minimum spanning tree. In particular, as recalled below, the range of μ defines a minimum spanning tree of an edge-weighted graph when the ordering $\prec$ ranks the edges of the graph with respect to weights.

In the sequel, we denote by w a map from E to $\mathbb{R}$ that weights the edges of E. Therefore, the pair (G, w) is called an edge weighted graph, and, for any u in E, the value $w(u)$ is called the weight of u.

An ordering $\prec$ on E is an altitude ordering for w if, for any two edges u and v in E, $w(u) < w(v)$ implies $u \prec v$.

[2] unnecessary text : Aller voir les travaux de Ronse sur ce qui se fait en algebre des hierarchies,inflating, merginc etc.

Property 7. *Let $\prec$ is an altitude ordering for w. Then, there exists a minimum spanning tree (V, E') for w such that the map $\mu^{\prec}$ is a bijection between $R^{\star}(\mathcal{B}^{\prec})$ and E'.*

The algorithm presented in RefPlayingKruskal allows one to compute both the hierarchy $\mathcal{B}^{\prec}$ (or, more precisely, its component tree) and the map $\mu^{\prec}$.

1.4. Join and insert operators on hierarchy

In this section, we introduce the two main binary operations, called join and insert, that allow us to compute binary partition hierarchies in a out-of-core manner, that is perform a computation while trying to minimize the quantity of information which is simultaneously needed during the calculus. The basic idea is to work independently on small pieces of the space, join the information found on adjacent pieces and propagate (or insert) this information into other pieces. After presenting the definition of these operators, we investigate some basic properties which are then used in the following section to establish the out-of-core calculus of distributed binary partition hierarchies.

The *extent* of a hierarchy is the union of the set of its regions and its *ground* is the support of its 0-scale. The ground and the extent of a hierarchy $\mathcal{H}$ are denoted by $gr(\mathcal{H})$ and $ex(\mathcal{R})$, respectively.[3]

Definition 8 (join). *Let $\mathcal{X}$ and $\mathcal{Y}$ be two hierarchies in $\mathcal{H}_{\ell}(V)$* [4]*. The* join of $\mathcal{X}$ and $\mathcal{Y}$*, denoted by $join(\mathcal{X}, \mathcal{Y})$, is the hierarchy defined by $join(\mathcal{X}, \mathcal{Y}) = (\mathcal{X} \sqcup \mathcal{Y}) \boxplus F$, where F is the neighborhood of the grounds of $\mathcal{X}$ and of $\mathcal{Y}$,* i.e.*, $F = \gamma(gr(\mathcal{X}), gr(\mathcal{Y}))$.*

The next property indicates that the binary partition hierarchy for $\prec$ on a set $A \cup B$, can be obtained by first considering the binary partition hierarchies on A and on B and then considering the join of these two hierarchies.

Property 9. *Let A and B be two subsets of V. Then, we have:*

- $join(\mathcal{B}^{\prec}_{A},\ \mathcal{B}^{\prec}_{B}) = \mathcal{B}^{\prec}_{A \cup B}$.

Definition 10 (insert). *Let $\mathcal{X}$ and $\mathcal{Y}$ be two hierarchies.*

We say that $\mathcal{X}$ is insertable in *$\mathcal{Y}$ if, for any λ in $\{0, \ldots, \ell\}$, for any region Y of $\mathcal{Y}[\lambda]$, Y is either included in a region X of $\mathcal{X}[\lambda]$ or is included in $V \setminus gr(\mathcal{X}[\lambda])$*[5]*.*

[3] unnecessary text : If $\mathcal{X}$ and $\mathcal{Y}$ are two hierarchies of disjoint extent, their supremum $\mathcal{X} \sqcup \mathcal{Y}$ is the such that $(\mathcal{X} \cup \mathcal{Y})[\lambda] = \mathcal{X}[\lambda] \cup \mathcal{Y}[\lambda]$ for any λ in $\{0, \ldots, \ell\}$.

[4] unnecessary text : of disjoint extent

[5] the notion of insertable hierarchy and the operation of insert can be defined in a little more general setting but in this case Lemmas 11 and 12 would not hold true in general. Alternative def: We say that $\mathcal{X}$ *is insertable in* $\mathcal{Y}$ if, for any λ in $\{0, \ldots, \ell\}$, for any region Y of $\mathcal{Y}[\lambda]$, $Y \cap gr(\mathcal{X}[\lambda]) \in \{Y, \emptyset\}$

Let $\mathcal{X}$ be insertable in $\mathcal{Y}$. The insertion of $\mathcal{X}$ into $\mathcal{Y}$ *is the hierarchy $\mathcal{Z}$, such that, for any λ in $\{0,\dots,\ell\}$, $\mathcal{Z}[\lambda] = \mathcal{X}[\lambda] \cup \{R \in \mathcal{Y}[\lambda] \mid R \cap gr(\mathcal{X}[\lambda]) = \emptyset\}$. The insertion of $\mathcal{X}$ into $\mathcal{Y}$ is denoted by insert$(\mathcal{X},\mathcal{Y})$.*

Lemma 11. *Let $\mathcal{X}$ and $\mathcal{Y}$ be two hierarchies in $\mathcal{H}_\ell(V)$ such that X is insertable in Y. Then, we have*

- *insert$(\mathcal{X},\mathcal{Y}) = \mathcal{X} \sqcup \mathcal{Y}$.*

The next lemma states that the selection is distributive over the insertion. Thus, when one needs to calculate a selection of the insertion of a hierarchy into another one, then it is enough to insert a selection of the first hierarchy in the selection of the second. In other words, the insertion can be performed in subparts of the hierarchy instead of on the whole hierarchy, a desirable property in the context of out-of-core computation.

Lemma 12. *Let A be a subset of V and let $\mathcal{X}$ and $\mathcal{Y}$ be two hierarchies such that $\mathcal{X}$ is insertable in $\mathcal{Y}$. Then, we have*

- *Select$(A,$ insert$(\mathcal{X},\mathcal{Y})) =$ insert$($Select$(A,\mathcal{X}),$ Select$(A,\mathcal{Y}))$*

The following lemma shows that when one needs to add a certain set of edges to a hierarchy, the addition can be performed "locally" on a subpart of the hierarchy (of the regions containing an element of the edges to be added) before being inserted in the initial hierarchy.

Lemma 13. *Let $\mathcal{X}$ be a hierarchy in $\mathcal{H}_\ell(V)$ and let F be a subset of E. Then, we have*

- *$\mathcal{X} \boxplus F =$ insert$($Select$(\delta^\bullet(F),\mathcal{X}) \boxplus F, \mathcal{X})$.*

The next property indicates a out-of-core calculus for computing the distribution of a binary partition hierarchy over a bi-partition $\{A,B\}$. This calculus is out of core in the sense that it does not need to consider the binary partition tree on the whole set $A \cup B$.

Property 14. *Let A and B be two subsets of V. Let R_A (resp. L_B) be the subset of A (resp. B) that contains every vertex of A (resp. B) adjacent to B (resp. A). Let $\mathcal{R} =$ Select$(R_A, \mathcal{B}_A^\prec)$ and $\mathcal{L} =$ Select$(L_B, \mathcal{B}_B^\prec)$. Then, we have:*

- *Select$(A, \mathcal{B}_{A\cup B}^\prec) =$ insert$($Select$(A,$ join$(\mathcal{R},\ \mathcal{L})), \mathcal{B}_A^\prec)$.*

The next property is the key ingredient to extend the above calculus to partitions with more than two sets. This extension will be studied in the next section.

Property 15. *Let A, B and C be three subsets of V that are pairwise disjoint. Let R_A (resp. L_B) be the subset of A (resp. B) that contains every vertex of A (resp. B) adjacent to B (resp. A). Let $\mathcal{R} =$ Select$(R_A, \mathcal{B}_A^\prec)$, $\mathcal{L}_1 =$ Select$(L_B, \mathcal{B}_B^\prec)$, and $\mathcal{L}_2 =$ Select$(L_B, \mathcal{B}_{A\cup B\cup C}^\prec)$. Then, we have :*

1. $join(\mathcal{R},\ \mathcal{L}_2) = insert(\mathcal{L}_2, join(\mathcal{R},\ \mathcal{L}_1))$
2. $Select(A, \mathcal{B}^{\prec}_{A\cup B\cup C}) = insert(Select(A_B, join(\mathcal{R},\ \mathcal{L}_2)), \mathcal{B}^{\prec}_A)$
3. $Select(A, \mathcal{B}^{\prec}_{A\cup B\cup C}) = Select(A, insert(\mathcal{L}_2, \mathcal{B}^{\prec}_A))$[6].

1.5. Causal distribution of BPTs

In this section, after providing the definition of a causal partition (sometimes refered to as a streaming in the literature), we introduce an out-of-core calculus to compute the distribution of a binary partition hierarchy on a causal partition. The main result of this section, namely Theorem 17 establishes the correctness of this calculus. In the next section, following this calculus, we introduce an efficient out-of-core algorithm to compute the binary partition hierarchy from real data like large images.

A causal partition of V is a series $(S_0, \ldots, S_k)$ of subsets of V such that:

1. $\{S_0, \ldots, S_k\}$ is a partition of V;
2. $\delta^{\times}(S_0) = \epsilon^{\times}(S_0) \cup \gamma(S_0, S_1)$;
3. $\delta^{\times}(S_i) = \epsilon^{\times}(S_i) \cup \gamma(S_i, S_{i-1}) \cup \gamma(S_i, S_{i+1})$, for any i in $\{1, \ldots, k-1\}$; and
4. $\delta^{\times}(S_k) = \epsilon^{\times}(S_k) \cup \gamma(S_k, S_{k-1})$.

Any element of a causal partition δ is called a *slice of* δ.

Let us now present the induction formulae to compute the distribution of binary partition hierarchy over a causal partition of the space.

Definition 16 (causal and anti-causal formulae). *Let $(S_0, \ldots, S_k)$ be a causal partition of V. For any i in $\{1, \ldots, k\}$, let L_i be the set of every vertex in S_i adjacent to a vertex in S_{i-1}, and, for any i in $\{0, \ldots, k-1\}$, let R_i be set of every vertex in S_i adjacent to a vertex in S_{i+1}. We set*

1. $\mathcal{B}^{\uparrow}_0 = \mathcal{B}^{\prec}_{S_0}$;
2. *for any i in $\{1, \ldots, k\}$,* $\mathcal{M}^{\uparrow}_i = join(Select(R_{i-1}, \mathcal{B}^{\uparrow}_{i-1}),\ Select(L_i, \mathcal{B}^{\prec}_{S_i}))$;
3. *for any i in $\{1, \ldots, k\}$,* $\mathcal{B}^{\uparrow}_i = insert(Select(L_i, \mathcal{M}^{\uparrow}_i), \mathcal{B}^{\prec}_{S_i})$;
4. $\mathcal{B}^{\downarrow}_k = \mathcal{B}^{\uparrow}_k$ *and* $\mathcal{M}^{\downarrow}_k = \mathcal{M}^{\uparrow}_k$;
5. *for any i in $\{0, \ldots, k-1\}$,* $\mathcal{B}^{\downarrow}_i = insert(Select(R_i, \mathcal{M}_{i+1}), \mathcal{B}^{\uparrow}_i)$; *and*
6. *for any i in $\{1, \ldots, k-1\}$,* $\mathcal{M}^{\downarrow}_i = insert(Select(L_i, \mathcal{B}^{\downarrow}_i), \mathcal{M}^{\uparrow}_i)$.[7].

[6] Formulation 3 is more simple (requires less operations) than formulation 1 to obtain $\text{Select}(A, \mathcal{B}^{\prec}_{A\cup B\cup C})$ without the need of computing $\mathcal{B}^{\prec}_{A\cup B\cup C}$. However, it does not lead to an efficient out-of-core algorithm. Should we keep it or drop it out?

[7] unnecessary text : It can be noticed that Equations (5) and 6 can be replaced by the following single formula:

On the one hand, the three first formula of Definition 16 provide us with a causal (inductive) calculus: the calculus of the i-th term of the series relies on the calculus of the $(i-1)$-th term of the series. In particular, it can be noted that $\mathcal{M}_i^{\uparrow}$ depends on $\mathcal{B}_{i-1}^{\uparrow}$ and that $\mathcal{B}_i^{\uparrow}$ depends on $\mathcal{M}_i^{\uparrow}$, for any i in $\{1,\ldots,k\}$. On the other hand, the three last formulae of Definition 16 provide us with an anti-causal calculus since the terms of index i rely on the one of index $(i+1)$: $\mathcal{B}_i^{\downarrow}$ depends on $\mathcal{M}_{i+1}^{\downarrow}$ and $\mathcal{M}_i^{\downarrow}$ depends on $\mathcal{B}_i^{\downarrow}$. Note also that $\mathcal{M}_i^{\downarrow}$ depends on $\mathcal{M}_i^{\uparrow}$ and that $\mathcal{B}_i^{\downarrow}$ depends on $\mathcal{B}_i^{\uparrow}$. Roughly speaking, it means that the anti-causal calculus must be started after the causal ones. One can also observe that the computation of each term requires only the knowledge of hierarchies whose supports are included in two consecutive slices of the given causal partition. In this sens, we can say that the above formulae form an out-of-core calculus.

In the next section, we propose efficient algorithms to perform these out-of-core calculus. Before presenting these algorithms, let us establish Theorem 17 that states the correctness of these formulae to compute distributed binary partition hierarchies.

Theorem 17. *Let $(S_0,\ldots,S_k)$ be a causal partition of V. Let i be any element in $\{0,\ldots,k\}$. The following statements hold true:*

- $\mathcal{B}_i^{\uparrow} = \mathit{Select}(S_i, \mathcal{B}^{\prec}_{\cup\{S_j \mid j\in\{0,\ldots i\}\}})$*;*
- $\mathcal{B}_i^{\downarrow} = \mathit{Select}(S_i, \mathcal{B}_V^{\prec})$*; and*
- *the set $\{\mathcal{B}_i^{\downarrow}\}$ is the distribution of $\mathcal{B}_V^{\prec}$ over the partition $\{S_0,\ldots,S_k\}$.*

4' $\mathcal{B}_i^{\downarrow} = \mathrm{Select}(S_i, \mathrm{insert}(\mathcal{L}, \mathcal{B}_{S_i}^{\uparrow}))$, where $\mathcal{L} = \mathrm{Select}(L_{i+1}, \mathcal{B}_{i+1}^{\uparrow})$, for any i in $\{0,\ldots,k-1\}$.
Despite being simpler, this formulation does not lead to an efficient out-of-core algorithm