Stage M2 ISI : Arbre des coupes multivalué pour l'analyse d'images

Romain Perrin

31 août 2019

Encadrement :
Benoît Naegel <b.naegel@unistra.fr>
Adrien Krähenbühl <krahenbuhl@unistra.fr>

Table des matières

1 Contexte

La morphologie mathématique est un domaine de recherche actif dans les thématiques de traitement de l'image. Une famille de techniques basées sur des représentations hiérarchiques des images a fournit de nombreuses structures de données et algorithmes permettant de traiter des images en niveaux de gris efficacement en s'appuyant sur les notions d'opérateurs connexes et de filtres d'attributs. Ces structures désormais bien connues, ne permettent pas de traiter correctement des images multivaluées. La recherche actuelle porte ainsi vers l'extension desdites structures aux images multivaluées (couleurs, multispectrales,...). Dans cette optique, nous proposons une structure dénommée arbre des coupes multivalué ainsi qu'une méthode de construction d'un ordre partiel basée sur les informations intrinsèques des images multivaluées dans des optiques de simplification et/ou de segmentation.

2 Notions

2.1 Morphologie mathématique

La morphologie mathématique est une branche de l'analyse d'images initialement développée en France par G.Matheron et J.Serra et se basant sur les notions d'algèbre, de théorie des ensembles ainsi que sur des principes géométriques [1], [6].

2.2 Relation d'ordre

Soient E un ensemble et $\leq$ une relation binaire sur E.
$\leq$ est une relation d'ordre sur E si, pour tout x, y et $z \in E$:
- $x \leq x$ (réflexivité)
- $x \leq y$ et $y \leq x \Rightarrow x = y$ (anti-symétrie)
- $x \leq y$ et $y \leq z \Rightarrow x \leq z$ (transitivité)

2.3 Treillis

Soit E un ensemble et $\leq$ une relation d'ordre sur E. $(E, \leq)$ est appelé *treillis* si tout couple d'éléments de E possède une borne inférieure et une borne supérieure. La borne inférieure (ou infimum) est le plus grand des minorants. La borne supérieure (ou supremum) est le plus petit des majorants. Un treillis est dit *complet* si chaque paire d'éléments possède une borne inférieure ou (propriété équivalente) une borne supérieure.

2.4 Opérateur connexe

Une notion centrale en morphologie mathématique est la notion d'opérateur connexe. Un opérateur connexe est un outil de filtrage qui agit par fusion de régions élémentaires appelées zones plates [2].

Une partition $\mathcal{P}$ d'une image est un ensemble de régions non vides et non chevauchantes qui remplit l'espace.

Plus formellement, un opérateur ψ est dit connexe si la partition des zones plates de son entrée f est toujours plus fine que la partition des zones plates de sa sortie $\psi(f)$ et l'on note $\mathcal{P}_f \sqsubseteq \mathcal{P}_{\psi_f} \forall f$.

Un opérateur ψ agissant sur une image f est dit :
- *croissant* si l'ordre présent entre deux images en entrée est préservé par le filtrage c'est-à-dire si $\forall f, g, f \leq g \Rightarrow \psi(f) \leq \psi(g)$.
- *idempotent* si l'application successive de l'opérateur ψ donne toujours le même résultat que l'application unique de ψ. Autrement dit un opérateur est idempotent si le filtrage d'un signal arbitraire produit un invariant $\forall f, \psi(\psi(f)) = \psi(f)$.
- *extensif* si l'image d'entrée est toujours plus petite que l'image de sortie $\forall f, f \leq \psi(f)$.
- *anti-extensif* si l'image de sortie est toujours plus petite que l'image d'entrée $\forall f, \psi(f) \leq f$.

Ces opérateurs connexes présentent quelques propriétés intéressantes. Ils ne sont pas capables de créer des nouveaux contours (de zones plates) ni de modifier les contours existants. En outre ils ne sont capables que de fusionner des contours entre différentes zones plates.

Ces propriétés permettent de garantir la préservation des contours ainsi que d'agir sur l'image à différentes échelles : filtrage bas-niveau mais aussi reconnaissance d'objets (haut-niveau).

2.5 Filtre d'attributs

Une transformation connexe consiste à supprimer les composantes connexes ne satisfaisant pas un certain critère (aire, périmètre...). Une transformation idempotente et croissante est appelée un *filtre*. Une transformation anti-extensive et idempotente est appelée un *amincissement* (thinning) [6].

Il existe deux approches principales afin d'utiliser les opérateurs connexes au traitement d'images [2]. Une première famille de méthodes consistent en un filtrage de l'image puis en une reconstruction. Une deuxième famille de méthodes consiste à utiliser une représentation basée sur des régions hiérarchiques de l'image, de simplifier la structure obtenue (généralement un arbre), puis de construire une image de sortie à partir de la structure.

3 Structures de données

Cette section présente quelques structures de données usuelles dans le domaine des représentations hiérarchiques d'une image. Celles-ci permettent de réaliser des traitements similaires ou proches de la méthode que nous proposons. Cette section est réalisée dans le cadre d'un état de l'art des structures analogues à la méthode proposée.

3.1 Arbre de partitionnement binaire

Un arbre de partitionnement binaire (binary partition tree ou BPT) est une structure permettant une représentation hiérarchique d'une image sous forme d'un arbre (binaire) [13]. Chaque nœud du BPT correspond à une région connexe. Les nœuds feuilles sont appelés *régions élémentaires*. Les nœuds intermédiaires modélisent l'union des deux régions contenues dans les nœuds fils. Le nœud racine contient le support de l'image représentée.

Le BPT permet d'explorer l'image représentée à différentes échelles et peut être utilisé pour effectuer des tâches de segmentation, d'extraction d'informations, de reconnaissance d'objets ainsi que de navigation visuelle [13]. Un avantage du BPT est de permettre l'introduction d'une métrique qui n'est pas intrinsèquement liée à l'image comme cela serait le cas d'autres structures généralement basées sur le contraste de l'image. L'ajout de cette métrique fait du BPT un modèle mixte entre informations tirées de l'image et connaissances a priori (métrique).

Il existe de plus des algorithmes efficaces permettant de calculer le BPT [12]. Cependant un inconvénient majeur du BPT est que celui-ci est calculé pour une image et une métrique données. De plus cette métrique dépendante d'une connaissance de haut niveau, est une métrique linaire plutôt inadaptée à des espaces n-dimensionnels tels que les images multispectrales, couleurs...

3.2 Arbre de partitionnement binaire multi-critères

Un nouvel algorithme de calcul du BPT a été proposée par Randrianasoa et al. [13] afin d'autoriser l'utilisation de plusieurs métriques et qui sied mieux aux images multivariées. Les auteurs ont montré deux exemples d'utilisation de leurs BPT multi-critères dans deux cas de figures.

Dans un premier temps le BPT multi-critères a été construit pour une seule image mais en utilisant plusieurs métriques. L'image utilisée fait partie d'un ensemble d'images de la ville de Strasbourg ; l'image utilisée est une image urbaine de type VHSR (Very High Spectral Resolution) à quatre bandes spectrales (NIR, R, G, B). Les résultats mettent en avant la capacité à extraire des zones correspondant à la fois à des objets urbains simples mais aussi à des objets complexes au sein de la même partition ce qui n'a pas pu être obtenu à l'aide du BPT classique (une image et une métrique).

Dans un second temps et afin de montrer la versatilité et la flexibilité du BPT multi-critères, celui-ci à cette fois-ci été utilisé dans un contexte d'imagerie multimodale et plus particulièrement de segmentation d'images radiologiques. Les images utilisées se composaient à la fois d'images PET (Positron Emission Tomography) mais aussi d'images CT (Computed Tomography) obtenues par rayons-X. Les images CT comportent des zones homogènes caractérisant des tissus et organes spé-

cifiques. Les images PET quant à elles, présentent des minima d'intensité locale aux positions des tumeurs actives mais avec une précision plus faible que les images CT. L'idée a été de coupler les deux espaces de valeurs en un seul afin d'extraire des zones homogènes réunissant la précision spatiale des images CT et la précision spectrale des images PET afin de segmenter de manière précise les tumeurs et les organes.

3.3 Arbre des formes

L'arbre des formes (tree of shapes ou ToS) est la carte topographique des niveaux de gris d'une image en terme d'inclusion des lignes de niveaux [17]. Chaque nœud de l'arbre représente une composante connexe dont le contour est une ligne de niveau. Le ToS offre une description du contenu de l'image en tant que collection de composantes connexes structurées dans un arbre modélisant les relations d'inclusions des composantes connexes. Cette structure peut être calculée efficacement [14] mais aussi efficacement stockée en mémoire [16].

Différentes tâches peuvent être réalisées à l'aide du ToS telles que du dé-bruitage, du filtrage, de la détection de caractéristiques locales, de l'indexation de textures, de la classification, de la segmentation... [15], [17].

S'appuyant sur les opérateurs connexes, le ToS bénéficie des propriétés intrinsèques auxdits opérateurs notamment la non création ou suppression des contours des composantes connexes et l'invariance par transformation affine (les contours ne peuvent être déplacés). Dans la mesure où le ToS se base sur l'inclusion de lignes de niveaux, il permet une analyse multi-échelles de l'image. En effet, les lignes de niveaux peuvent être de grande courbes fermées et ne sont pas des descripteurs locaux contrairement à beaucoup d'autres détecteurs.

3.4 Arbre des formes multivalué

Comme beaucoup d'autres arbres morphologiques, le ToS se base sur une relation d'ordre des valeurs de l'image. Or cet ordre doit être total. Dans le cas contraire, les composantes connexes peuvent se chevaucher et l'arbre obtenu n'est pas un arbre d'inclusion. Bien définit pour les images en niveaux de gris, le ToS ne l'est pas pour des images multivariées.

Plusieurs techniques ont été proposées pour s'abstraire de cette contrainte notamment en construisant un ordre total arbitraire mais sans donner réellement satisfaction. Carlinet et Géraud ont introduit la notion d'arbre des formes multivalué (multivariate tree of shapes ou MToS) dans [17] permettant de construire un arbre des formes à partir d'un ordre partiel.

La méthode proposée se compose de deux parties principales. La première partie consiste à construire un *graphe des formes* (GoS) à partir de plusieurs arbres des formes (ToS). L'image multivariée est séparée en plusieurs canaux et sur chaque canal, un ToS est construit. La seconde partie consiste à déduire un arbre à partir du GoS. Le résultat de cette méthode est un arbre morphologique présentant une invariance à tout changement ou inversion marginal de contraste.

3.5 Arbre des coupes

Les ensembles de niveaux d'une image en niveaux de gris sont les ensembles de points dont le niveau est au-dessus d'un certain seuil. Les composantes connexes de ces ensembles de niveaux peuvent être organisées dans une structure d'arbre grâce à leur relations d'inclusions. Cet arbre est alors appelé l'*arbre des coupes* [11] de l'image. Il est également connu sous divers autre noms tels que dendrone, arbre d'inclusion ou encore max-tree.

Les applications sont nombreuses : filtrage et segmentation d'images, segmentation vidéo [4], enregistrement d'image, compression d'image, visualisation de données.... Le structure est relativement bien connue et des algorithmes efficaces existent pour calculer l'arbre des coupes (voir [11] pour une comparaison de différents algorithmes de calcul du max-tree).

Le processus classique de filtrage consiste en trois étapes. La première étape est la construction de l'arbre des coupes de l'image. La seconde étape consiste à filtrer l'arbre des coupes c'est-à-dire choisir quel(s) nœud(s) conserver ou supprimer. Enfin la troisième étape consiste à faire correspondre les composantes de l'arbre filtré avec l'image originale afin de construire l'image filtrée.

3.6 Graphe des coupes

L'arbre des coupes souffre du même problème inhérent aux images multivariées que l'arbre des formes. La construction (efficace) de l'arbre des formes repose sur le caractère total de la relation d'ordre des valeurs de l'image [8]. Passat et Naegel ont proposé une extension du concept d'arbre des coupes aux ensembles partiellement ordonnés dans [9]. La structure obtenue dans le cas d'un ordre partiel est un graphe dénoté *graphe des coupes* ou component-graph.

En considérant plusieurs sous-ensembles différents, il est possible de décliner le graphe des coupes en trois variantes notées $\mathfrak{G}$, $\dot{\mathfrak{G}}$ et $\ddot{\mathfrak{G}}$ [9]. Deux algorithmes ont été proposés pour calculer respectivement les variantes $\mathfrak{G}$ et $\ddot{\mathfrak{G}}$. La variante $\dot{\mathfrak{G}}$ peut se déduire à partir de la variante $\mathfrak{G}$ et se calcule de fait en construisant d'abord la variante $\mathfrak{G}$ puis en appliquant un étape supplémentaire.

Le calcul du graphe des coupe reste cependant une opération coûteuse en temps, l'algorithme de calcul étant quadratique. Un travail réalisé précédemment en TER a consisté en l'implémentation de l'algorithme de calcul de la variante $\ddot{\mathfrak{G}}$ du graphe des coupes pour des images de labels [20]. L'implémentation a été réalisée en C++ [21] dans le cadre de la bibliothèque de traitement d'images *LibTIM* [22].

3.7 Arbre des coupes multivalué

Au même titre que d'autres structures (arbre de partitionnement binaire ou arbre des formes), l'arbre des coupes ne permet pas de traiter les images multivaluées. La principale difficulté réside dans le fait qu'il n'existe généralement pas d'ordre intrinsèque à ces ensembles n-dimensionnels. Bien que la structure de graphe des coupes ait été proposée et que des algorithmes existent pour le calculer, la complexité asymptotique de ces algorithmes posent problème.

Kurtz et al. [10] ont étudié l'influence de la relation d'ordre associée aux valeurs de l'image sur la construction du graphe des coupes. Bien que l'ordre soit partiel ce qui n'entre pas dans le cas de la construction d'un arbre des coupes, lorsque celui-ci est un arbre et non un graphe, le graphe des coupes résultant et également un arbre. L'arbre obtenu dans ce cas particulier est alors nommé *arbre des coupes multivalué*.

Tirant partie de la structure d'arbre de la relation d'ordre, un algorithme efficace a pu être proposé [10] se basant sur les algorithmes classiques et efficaces de construction de l'arbre des coupes. Une certaine adaptation est cependant nécessaire dans le mesure où l'ordre utilisé n'a pas forcément une structure d'arbre. C'est justement l'objectif de ce stage. En effet, on se propose d'explorer une méthode de construction d'un ordre hiérarchique (sous forme d'arbre donc) à partir duquel l'arbre des coupes multivalué sera calculé.

4 Méthode de construction proposée

Cette section a pour but de décrire le processus de construction de l'ordre hiérarchique bas ainsi que le calcul de l'arbre des coupes multivalué. Les différents algorithmes sont donnés dans le cas général sans prendre en considération l'espace des valeurs de l'image (niveaux de gris, couleurs, spectre...). Les différentes adaptations dues aux espaces de valeurs explorés sont détaillées dans la section suivante traitant des développements effectués.

4.1 Considérations

Une solution classique pour remédier au problème des ordres partiels est d'utiliser des ordres totaux ou de rendre totaux les ordres partiels considérés. Classiquement, pour les images en niveaux de gris, l'ordre naturel ou l'ordre naturel inverse conviennent très bien. En revanche, dans le cas des images multivaluées, il n'existe généralement pas d'ordre (total) inhérent à l'espace des valeurs considéré.

Si l'on se place du point de vue de l'espace RVB il existe bien un ordre marginal : un treillis complet comportant un minimum global (0,0,0) et un maximum global (255,255,255). Cependant ce treillis n'est pas un arbre mais bien un graphe et sort donc du cadre de l'arbre des coupes multivalué. De plus cet ordre marginal n'a que peu de sens car c'est avant tout un ordre artificiel.

L'idée est de tirer partie d'une connaissance extraite de l'image. Pour ce faire, on considère l'histogramme de l'image. Celui-ci contient les densités associées à chaque valeur de l'espace des valeurs de l'image. L'objectif est ici de construire un ordre hiérarchique bas en se basant sur cet histogramme.

4.2 Ordre hiérarchique bas

Soit I une image quelconque et V l'espace des valeurs de I. Soit H l'histogramme des valeurs de I. Pour chaque valeur $v \in V$, $H(v)$ est égal au nombre de pixels de I ayant la valeur v.

Un observation importante est que les valeurs v pour lesquelles les densités $H(v)$ sont les plus élevées sont plus susceptibles de correspondre visuellement à des morceaux d'objets ou des objets complets d'intérêts. Afin de construire une structure d'arbre un algorithme de construction d'arbre des coupes est utilisé sur les valeurs v de l'histogramme H (algorithme 1).

4.2.1 Construction d'un max-tree à partir de l'histogramme

L'algorithme 1 expose le processus de construction du max-tree sur les valeurs de l'histogramme. En réalité les valeurs ne sont pas utilisées directement mais l'on passe par une représentation intermédiaire des valeurs par un ensemble d'entiers. Il s'agit donc des indices des valeurs de l'ensemble des valeurs de l'image. L'indexation dépend de l'espace des valeurs considéré.

À titre d'exemple l'indexation pour l'ensemble des niveaux de gris est exactement l'ensemble des valeurs i.e. $\forall v, i \in [\![0, 255]\!], v = i$.
Pour l'ensemble RVB, l'indexation utilisée est la suivante : $\forall (r, v, b) \in [\![0, 255]\!]^3, \forall i \in [\![0, 2^{24}-1]\!], i = r + v * 2^8 + b * 2^{16}$.

L'algorithme 1 construit un max-tree sur les indices des valeurs de l'image. Il existe plusieurs algorithmes efficaces permettant de construire un max-tree, notamment les algorithmes de Salembier et de Berger (voir un comparatif des algorithmes de construction de max-tree dans [11]).

L'algorithme de base utilisé ici est l'algorithme de Berger [11]. Il est à noter qu'un critère supplémentaire lié au voisinage des valeurs a été ajouté (ligne 7). Celui-ci permet de corréler les valeurs proches afin de les classer dans la même branche de l'arbre. Les densités sont parcourues dans l'ordre décroissant (ligne 1). Cela a pour effet de positionner les intensités les plus élevées près des feuilles de l'arbre. Il est bien sûr important de noter que le voisinage est présenté ici de manière générique étant donné que celui-ci dépend de la nature de l'espace des valeurs de l'image.

Certaines propriétés sont à noter quant à la répartition des valeurs/indices.

- Au sein d'un pic donné, si v_c est la valeur centrale du pic (maximum local des densités) et v_c^{-1} et v_c^{+1} les deux valeurs voisines, alors v_c sera la valeur la plus proche des feuilles et le $max(v_c^{-1}, v_c^{+1})$ sera son parent et le $min(v_c^{-1}, v_c^{+1})$ le parent de son parent.

- Si v_{c1} et v_{c2} sont les valeurs centrales de deux pics différents (i.e. les deux valeurs ne sont pas voisines), alors il existe une valeur intermédiaire v_i entre les deux pics telle que v_{c1} soit dans une branche de l'arbre, v_{c2} sont dans une autre branche de l'arbre et v_i est un parent de v_{c1} et de v_{c2}. Cette propriété assure que l'on obtient bien un arbre et non une forêt d'arbres disjoints en fin de traitement.

L'arbre est représenté au moyens de tableaux contenant les indices des valeurs, il ne s'agit pas encore d'un arbre au sens classique et structurel du terme. Une dernière étape consiste à canoniser l'arbre via la fonction canonize.

4.2.2 Extraction d'une structure d'arbre

Comme indiqué précédemment, l'arbre n'est pas encore un arbre visuellement parlant. Il s'agit maintenant d'extraire les informations du max-tree et de stocker l'arbre dans une structure d'arbre classique. L'algorithme 2 réalise l'extraction d'un arbre à partir de l'histogramme et du max-tree calculé par l'algorithme 1.

La structure de données d'un nœud est décrite ici. Un nœud possède les attributs suivants :
- **index** est l'indice du nœud. Chaque nœud est identifié par un indice unique.
- **value** est la valeur du nœud. Ici la valeur est la densité dans l'histogramme.
- **father** est le parent du nœud (un seul par nœud).
- **childs** est la liste des fils du nœud.
- **pixels** contient ici les indices des valeurs de l'histogramme associés à ce nœud (les indices peuvent être regroupés dans le cadre de classes d'équivalence).

L'idée est assez simple, on parcours chaque indice de l'histogramme (ligne 1) et l'on crée un nouveau nœud (ligne 2) en stockant l'indice de l'histogramme dans l'attribut **pixels** et la densité associée à cet indice dans l'attribut **value**. En utilisant l'information de parenté entre les indices donnée par le max-tree, les attributs **father** et **childs** sont complétés (lignes 4 à 11). Si la densité associée à un indice est égale à la densité associée à l'indice de son parent, alors les deux indices

Algorithm 1: MAXTREE(*hist*, *bins*)

Input: *hist* : histogram values, *bins* : sorted histogram indices ($<$)
Output: *parent* : parent relation between the histogram's indices
// *parent* is an array of size(hist) initialized with -1
// *zpar*, *rank* and *repr* are arrays of size(hist) initialized with 0

```
1  for all indices i in bins do
2      parent[i] ← i
3      zpar[i] ← i
4      rank[i] ← 0
5      repr[i] ← i
6      zp ← i
7      for all valid neighbour indices j of i do
8          if parent[j] ≠ −1 then
9              zn ← FIND(J)
10             if zn ≠ zp then
11                 parent[repr[zn]] ← i if rank[zp] < rank[zn] then
12                     tmp ← zp
13                     zp ← zn
14                     zn ← tmp
15                 zpar[zn] ← zp zpar[zp] ← zn if rank[zn] = rank[zn] then
16                     rank[zp] ← rank[zp] + 1
17 CANONIZE(HIST)
```

Function find(x, parent)

Input: x : element, *parent* : array in which to find the parent of x
Output: the parent of x

```
1 if parent[x] ≠ x then
2     parent[x] ← find(parent[x], parent)
3 return parent[x]
```

Function canonize(hist, bins)

Input: *hist* : histogram values, *bins* : sorted histogram indices ($<$)

```
1 for all indices p in bins do
2     q ← parent[p]
3     if hist[q] = hist[parent[q]] then
4         parent[p] ← parent[q]
```

sont stockés dans le même nœud (celui de son parent) dans le cadre d'une classe d'équivalence (lignes 10 et 11).

Algorithm 2: BUILD_TREE(*hist*, *bins*, *parent*)

Input: *hist* : histogram values, *bins* : sorted histogram's indices (<), *parent* : parent relation in max-tree
Output: *nodes* : nodes of the graph

```
1  for all indices i of hist do
2      n ← create_node(hist[i])
3      nodes.append(n)
4  for all sorted indices i on bins do
5      if hist[i] ≠ hist[parent[i]] then
6          nodes[i].value ← hist[i]
7          nodes[i].pixels.append(i)
8          nodes[i].father ← nodes[parent[i]]
9          nodes[parent[i]].chils.append(nodes[i])
10     else
11         nodes[parent[i]].pixels.append(i)
12 return nodes
```

4.3 Construction de l'ordre hiérarchique bas

On dispose maintenant de notre max-tree basé sur l'histogramme de l'image en bonne et due forme. Il est maintenant temps de définir l'ordre hiérarchique sur les valeurs de l'image en construisant une relation d'ordre partiel sous forme d'ordre hiérarchique bas. Notre ordre comportera donc un minimum global correspondant au support de l'image représentée.

La définition de l'ordre passe par un processus à trois étapes. Une première étape consiste à calculer tous les labels (représentant les valeurs de l'image avec un niveau d'abstraction). Une seconde étape consiste à pré-calculer les comparaisons (<) entre lesdits labels. Enfin une série de prédicats utiles à la construction postérieure de l'arbre des coupes multivalué sont définis à l'aide de ces données composant ainsi la troisième et dernière étape du processus.

L'algorithme 3 permet de calculer tous les labels intervenant dans le relation d'ordre. La structure de données d'un labels est la suivante :
- **index** est l'indice unique identifiant le label (généré automatiquement).
- **value** est la valeur du label c'est-à-dire la valeur de l'espace des valeurs de l'image représentée par le label.
- **father** est la label parent du label considéré.
- **childs** est la liste des labels fils du label considéré.

Les nœuds du max-tree sont traités en démarrant par la racine (lignes 1 à 6) qui est aussi le nœud correspondant à la plus faible densité (souvent 0) due à la méthode utilisée pour construire ledit max-tree. Le traitement est effectué à l'aide d'une file (ligne 1). Si le nœud traité contient

plusieurs indices de valeurs dans le cas d'une classe d'équivalence, un label est créé pour chacun d'entre eux (lignes 11 à 17). les relation père-fils sont les mêmes qui celles régissant le max-tree avec en addition les relations de parenté internes aux classes d'équivalence désormais éclatées.

Algorithm 3: COMPUTE_LABELS(*nodes*, *bins*, *labels*)

Input: nodes of the max-tree *nodes*,
sorted histogram's indices *bins*,
list of labels (empty) *labels*
Output: label of the root of the max-tree $label_{root}$

```
1  fifo ← empty_fifo_queue()
2  root ← nodes[bins[0]]
3  label_root ← create_label(root.pixels[0])
4  label_root.father ← label_root
5  labels.append(label_root)
6  fifo.push((root, label_root))
7  while ¬fifo.empty() do
8      node_current, label_current ← fifo.pop()
9      value_current ← node_current.pixels[0]
10     label_previous ← label_current
11     for all indices i in node_current.pixels except 0 do
12         value_current ← node_current.pixels[i]
13         label_current ← create_label(value_current)
14         labels.append(label_current)
15         label_current.father ← label_previous
16         label_previous.childs.append(label_current)
17         label_previous ← label_current
18     for all nodes n in node_current.childs do
19         label_child ← create_label(value_current)
20         labels.append(label_child)
21         label_child.father ← label_current
22         label_current.childs.append(label_child)
23         fifo.push((node_child, label_child))
24 return label_root
```

La seconde étape consiste à pré-calculer les comparaisons entre les labels créés à l'aide de l'algorithme 3. En terme précis, il s'agit de calculer la fermeture transitive de la relation d'ordre. L'objectif est de pré-calculer et de stocker le résultat des comparaisons entre labels. Ainsi, et lorsque l'on aura besoin de comparer les labels lors de la création de l'arbre des coupes multivalué, les comparaisons pourront être considérées en temps constant.

L'algorithme 4 calcule la fermeture transitive et stocke le résultat dans un tableau bidimensionnel (ligne 1). Ainsi comparer deux labels revient à rechercher la valeur dans le tableau indexé sur les indices des labels plutôt que d'effectuer une recherche de lien direct entre les deux indices des

labels dans l'arbre (max-tree). Le critère de comparaison utilisé et stocké est $<$ étant donné qui c'est celui dont on aura besoin ultérieurement. Il est nécessaire avant ce processus de calculer la liste des successeurs de chaque nœud au moyen de la fonction computesucc.

Algorithm 4: TRANSITIVE_CLOSURE($label_{root}$, $labels$)

Input: $label_{root}$: label of the root node of the ordering, $labels$: list of all computed labels
Output: $order$: order relation ($<$) on labels

```
1 order ← array2D(size(labels), 0)
2 computesucc(label_root)
3 for all labels l in labels do
4 |   for all successor labels c in l.succ do
5 |   |   order[l.index][c.index] ← 1
6 |   |   order[c.index][l.index] ← −1
7 return order
```

Function computesucc(l)

Input: label l

```
1 for all labels l in l.childs do
2 |   l.succ.append(c)
3 for all labels c in l.childs do
4 |   l.succ ← merge(l.succ, compute_succ(c))
```

On dispose désormais de toutes les comparaisons entre labels ainsi que d'un ordre hiérarchique bas sous forme d'arbre. Il est alors possible de définir des prédicats utiles à la création de l'arbre des coupes multivalué.

Premièrement, on peut définir le prédicat **est inférieur à** sur les labels au moyen de la fonction ilt. Concrètement, pour comparer deux labels a et b, il suffit de regarder la valeur présente dans le tableau **order** en utilisant la clef $(indice_a, indice_b)$. Si la valeur est 1, alors $a < b$. Autrement dit, il existe un chemin de a vers b dans l'ordre hiérarchique construit. Sinon, cela ne veut pas nécessairement dire que $b < a$ dans la mesure où l'ordre construit est partiel, il existe donc des valeurs non comparables. Si $a \not< b$, soit $b < a$, soit a et b ne sont simplement pas des valeurs comparables i.e. les labels sont dans deux branches différentes de l'arbre.

Function ilt(a, b)

Input: a : label index, b : label index
Output: $True$ if $a < b$, $False$ otherwise

```
1 if order[a][b] = 1 then
2 |   return True
3 else
4 |   return False
```

Étant donnée que les labels peuvent être non comparables, et c'est également un critère important par la suite, il est utile de définir le prédicat **est comparable à** sur les labels. La fonction comparable permet ceci. Deux labels a et b sont comparables si et seulement si :
- ils sont identiques ($a = a$, $b = b$)
- $a < b$ dans l'ordre hiérarchique construit
- $b < a$ dans l'ordre hiérarchique construit

N'importe laquelle de ces propositions est une condition nécessaire et suffisante pour affirmer que a et b sont comparables au sens de l'ordre hiérarchique construit.

Dans la fonction comparable, on fait usage du prédicat $<$ via un double appel à la fonction ilt (on teste les deux sens). Encore une fois ilt fait appel à un tableau où les relations sont précalculées et s'obtient donc en temps constant. Quant à la comparaison $a = b$, il suffit de comparer les indices uniques identifiant les deux labels ce qui revient à une comparaison entre deux entiers.

Function comparable(a, b)

Input: a : label index, b : label index
Output: $True$ if a and a are comparable, $False$ otherwise
1 **return** $a = b \vee ilt(a, b) \vee ilt(b, a)$

Enfin, et puisque l'on utilise un ordre hiérarchique bas, il est intéressant de connaître le prédécesseur d'un label. La fonction prec retourne le prédécesseur d'un label donné. Ici il suffit d'utiliser l'arbre de l'ordre hiérarchique bas contenant les labels. Les nœuds de cet arbre sont stockés dans le tableau **nodes**.

Function prec(l)

Input: label index l,
list of all labels $labels$
Output: the index of the direct predecessor of l
1 **return** $labels[l].father.index$

4.4 Arbre des coupes multivalué

On dispose maintenant d'un ordre hiérarchique bas, structurant les labels (valeurs de l'image) sous la forme arbre ainsi que de trois prédicats *est inférieur à*, *est comparable à* et *prédécesseur de*. On peut désormais utiliser cet relation d'ordre pour construire l'arbre des coupes multivalué d'une image.

L'image utilisée pour la construction de l'arbre des coupes multivalué est en réalité une image d'indices. Les indices composant cette image sont les indices des labels, eux-même une abstraction de l'espace des valeurs de l'image. De fait, avant de calculer l'arbre des coupes multivalué, il est nécessaire de calculer cette image d'indices en amont. Le processus n'est pas détaillé ici dans la mesure où chaque espace de valeurs comporte un degré inhérent de complexité variable et il n'y a pas d'algorithme "unique".

L'algorithme présenté ici est semblable à celui proposé dans [10]. Néanmoins, il s'agit d'un algorithme nouveau reposant sur quelques principes. L'algorithme de base est effectivement celui présenté dans [10] qui lui-même se base sur l'algorithme d'inondation proposé par Salembier (voir à nouveau les différents algorithmes efficaces [11]) et sur les masques de Wilkinson [2]. Cependant l'algorithme présenté ici comporte des ajustements permettant notamment de construire l'arbre des coupes multivalué sans avoir à augmenter l'image. Le processus général se compose de deux étapes : une étape d'initialisation et une étape d'inondation récursive.

L'algorithme 5 correspond à l'étape d'initialisation, puis au lancement de l'inondation initiale. L'inondation initiale est lancée sur le (premier) pixel ayant la valeur de densité la plus faible dans l'histogramme i_{least} (ligne 17). Afin de construire correctement l'arbre des coupes multivalué, la valeur de l'inondation initiale est l'indice du label racine (i_{root}). Un certain nombre de listes et tableaux sont nécessaires par la suite :
- node_at_level permet de savoir si un nœud existe pour un indice donné
- number_nodes permet de connaître de nombre de nœud pour un indice donné
- hq est une liste ; elle contient la liste des indices de pixels de l'image à traiter pour un indice donné.
- index est une liste ; elle contient les nœuds de l'arbre des coupes aux différents niveaux (indices)

On commence simplement par créer le nœud racine (lignes 14 à 16) puis on lance l'inondation initiale (ligne 17).

La fonction flood réalise l'inondation récursive. Il faut ici tenir compte de caractère partiel de la relation d'ordre. Il faut également tenir compte lorsque l'on traite un pixel de la valeur d'inondation (indice). Deux cas de figure sont possibles :
- soit le pixel a pour indice (valeur dans l'image d'indices) le même indice que la valeur d'inondation auquel cas il s'agit d'un pixel **réel**.
- soit les deux valeurs diffèrent et le pixel est considéré comme **fictif**. C'est ici que le caractère de séparation des pics de l'histogramme dans des branches séparées de l'ordre hiérarchique prend tout son sens ; lorsque l'on considère un pixel en bordure d'un objet, son voisin (appartenant à un autre objet) sera considéré fictif. Un pixel fictif est placé dans le file **hq** mais cette fois-ci à son indice réel (ligne 6).

Dans le cas d'un pixel **réel**, on explore ses voisins donnés par *adj* (ligne 15). Chaque paire (p, q)

Algorithm 5: COMPUTE_MCT(I, i_{root}, i_{least}, adj, $labels$)

Input: index image I, index of the label of the ordering's root i_{root}, index of the least represented label in the image i_{least}, pixel adjacency adj, list of all labels $labels$

Output: the root of the MCT $root$

```
1  I_flat ← flattened_image(I)
2  h_min ← i_root
3  status ← array1D(size(I), ACTIVE)
4  x_min ← 0
5  for all labels l in labels do
6  |   node_at_level[l.index] ← False
7  |   number_nodes[l.index] ← 0
8  |   hq[l.index] ← []
9  |   index[l.index] ← []
10 for all pixels p in I_flat do
11 |   if I_flat[p] = i_least then
12 |   |   x_min ← p
13 |   |   break
14 root ← create_node(h_min)
15 index[h_min].append(root)
16 hq[h_min].append(x_min)
17 FLOOD(h_min, i_root, adj)
18 return root
```

avec p le pixel courant et q un pixel voisin. Si les labels sont comparables alors q est ajouté à la zone plate de p (lignes 24 à 27). Sinon, l'astuce est de créer un nœud ayant comme indice l'infimum (ou borne inférieure) des labels $inf(i_p, i_q)$ (lignes 18 à 23). Enfin, les lignes 31 à 36 permettent de créer les liens de parenté entre les nœuds créés (stockés dans $index$).

Algorithm 6: $remove\_fictitious\_nodes(root)$

```
   Input: root of the MCT root
1  for all children nodes c in root.childs do
2      remove_fictitious_nodes(c)
3      if size(c.pixels) = 0 then
4          root.childs.remove(c)
5          for all children nodes nc in c.childs do
6              root.childs.append(nc)
```

Pour achever la construction de l'arbre des coupes multivalué, il est possible que des nœuds **fictifs** aient été créés. Ces nœuds ne contiennent pas de pixels. La fonction 6 permet de les supprimer afin de travailler avec un arbre minimal. Le principe est simple, on appelle la fonction récursive sur la racine de l'arbre et, pour chaque nœud, si le nombre de pixels est nul, le nœud est supprimé (ligne 4) et les relations de parentés entre le parent du nœud supprimé et les enfants de celui-ci sont reconstruites (lignes 5 et 6).

Function flood(h, i_{root}, adj)

Input: flooding level (index) h, adjecency on pixels adj
Output: the first ancestor of the node

```
1  while ¬hq[h].empty() do
2      p ← hq[h].pop()
3      if I_flat[p] ≠ h then
4          if status[p] = ACTIVE then
5              m ← I_flat[p]
6              hq[m].push(p)
7              status[p] ← INQUEUE
8              while ilt(h, m) do
9                  m ← flood(m, adj)
10     else
11         status[p] ← number_nodes[h]
12         if index[h][status[p]].empty() then
13             index[h][status[p]] ← create_node(p)
14         update_attributes(index[h][status[p]])
15         for all neighbours q of p in adj do
16             m ← I_flat[q]
17             if status[q] ≠ INQUEUE then
18                 if ¬comparable(h, m) then
19                     binf ← infimum(h, m)
20                     hq[binf].append(q)
21                     node_at_level[binf] ← True
22                     if index[binf][number_nodes[binf]].empty() then
23                         index[binf][number_nodes[binf]] ← create_node(binf)
24                 else
25                     hq[m].append(q)
26                     status[m] ← INQUEUE
27                     node_at_level[m] ← True
28                 while ilt(h, m) do
29                     m ← flood(m, adj)
30 m ← prec(h)
31 if m ≠ h then
32     while ¬node_at_level[m] ∧ prec(m) ≠ m do
33         m ← prec(m)
34     make_link(index[m][number_nodes[m]], index[h][number_nodes[h]])
35 number_nodes[h] ← number_nodes[h] + 1
36 node_at_level[h] ← False
37 return m
```

4.5 Filtrage d'attributs

Nous disposons désormais d'un algorithme permettant de construire l'arbre des coupes multivalué à partir d'un ordre hiérarchique bas basé sur l'histogramme des valeurs d'une image. Divers attributs peuvent être calculés pour chaque nœud de l'arbre (périmètre, aire, compacité... voir la section suivante pour plus de détails sur les attributs utilisés). Une application classique est le filtrage d'attributs. Cette opération consiste à activer ou désactiver certains nœuds de l'arbre ne respectant pas un certain critère donné.

Il est important de noter que la méthode d'activation ou désactivation des nœuds influe sur le résultat une fois l'image reconstruite à partir de l'arbre filtré. Par ailleurs, cette opération s'appelle la *politique de réduction*. Il existe principalement trois politiques de réduction : **directe**, **min** et **max** [3], [10].

En effet, cela dépend de la nature du filtre utilisé [6]. Si le filtre est croissant, alors si un nœud n est actif, tous les nœuds en dessous (jusqu'à la racine) devraient être également actif. Dans le cas contraire, avec un filtre non croissant, il est possible de voir apparaître des "trous" dans l'arbre c'est-à-dire des nœuds non actifs entre des nœuds ce qui n'est pas souhaitable.

Ainsi lorsque le critère n'est pas croissant, par exemple si l'on décide de supprimer les nœuds dont un attribut est inférieur à une borne ou dont le même attribut est supérieur à une autre borne (ex : $aire < 50$ *ou* $aire > 100$) il est important d'appliquer certains correctifs [3].

Pour le filtrage **direct**, il est important de toujours fusionner le contenu des nœuds qui sont supprimés (inactifs) avec leurs plus proche ancêtre actif. Pour le filtrage **min**, si un nœud est conservé (actif), alors tous ses ancêtres doivent être préservés. Enfin pour le filtrage **max**, si un nœud est supprimé (inactif), alors tous ses descendants doivent également être supprimés.

Dans la méthode proposée, c'est la politique **directe** qui est choisie. L'algorithme 7 (et la fonction récursive recfiltering) présente de manière générique un filtrage d'attributs des nœuds de l'arbre des coupes multivalué. Il est à noter que l'image d'indices obtenue après création de l'arbre des coupes multivalué par 5 est également modifiée en fonction du filtrage effectué (lignes 7 et 8). Celle-ci permettra a posteriori de générer l'image résultat. Le principe du filtrage est simple, les nœuds ne respectant pas un certain critère sur les attributs dudit nœud se voient *supprimer* de manière fictive. Le nœud n'est pas réellement supprimé, il est désactivé. La valeur de ce nœud (label) est alors remplacée par la valeur (label à nouveau) du nœud qui est sont plus proche ancêtre non supprimé (lignes 1 et 2). Dans le cas contraire, le nœud conserve sa valeur originale (lignes 3 et 4). La fonction est appelée récursivement sur tous les nœuds fils du nœud traité (lignes 5 et 6).

Algorithm 7: FILTERING($image_{index}$, *root*, *criterion*)

Input: index image $image_{index}$, MCT root *root*, attribute criterion *criterion*
Output: filtered index image $image_{index}$

```
value ← root.original_value
for all children node c of root.childs do
    RECFILTERING(image_index, c, value, criterion)
return image_index
```

Function recfiltering($image_{index}$, *c*, *node.value*, *criterion*)

Input: index image $image_{index}$, MCT node *node*, value to give to a removed node *value*, attribute criterion *criterion*

```
if node does not respect criterion on targetted attributes then
    node.value ← value
else
    node.value ← node.original_value
for all children node c in node.childs do
    RECFILTERING(image_index, c, node.value, criterion)
for all pixels p in node.pixels do
    image_index[p] ← node.value
```

4.6 Reconstruction de l'image

L'étape finale du processus consiste à reconstruire une image à partir de l'arbre des coupes multivalué une fois les traitements effectués sur celui-ci (filtrage dans notre cas). Il n'y a pas unicité des méthodes de reconstruction. On peut différencier deux principales classes de méthodes de reconstruction : les méthodes dites **directes** et les méthodes dites **indirectes**.

Les méthodes dites **directes** s'appuient sur l'image d'indices filtrée construite par l'algorithme 5. Les méthodes dites **indirectes** nécessitent quant à elles un parcours de l'arbre des coupes multivalué filtré.

4.6.1 Reconstructions directes

Comme dit précédemment, les méthodes de reconstruction directes n'ont pas besoin d'utiliser l'arbre des coupes multivalué filtré ce qui aboutit à deux conséquences :
- le traitement se réduit principalement au parcours d'une image dont les dimensions sont identiques à l'image donnée en entrée
- l'arbre des coupes multivalué ainsi filtré n'est pas utilisé et n'a de fait nul besoin d'être parcouru

Deux méthodes de reconstruction dites directes sont proposées ici. Le première, dénotée **directe-labels** est ce que l'ont pourrait appeler la méthode de reconstruction *naïve*. Le principe est simple ; chaque valeur différente présente dans l'image originale donne lieu à la création d'un label abstrait la représentant, et ce quelque soit l'espace des valeurs de l'image utilisé. Cette méthode naïve consiste à effectuer le passage inverse (on a bien une bijection entre les valeurs présentes et les labels crées).

La seconde méthode, dénotée **direct-repr** effectue un choix supplémentaire par rapport à la méthode naïve. L'idée est que, pour tous les labels appartenant à une classe d'équivalence, plutôt que d'attribuer à chacun la valeur lui correspondant, l'on choisisse un représentant (*repr = representative ou représentant en français*) de classe et c'est la valeur correspondant à ce représentant qui sera attribuée à chaque label de la classe. On peut aisément considérer cette seconde version comme étant simplement la reconstruction naïve mais avec ajout de représentants de classes, les classes existant aussi dans la version naïve mais sans utilisation des représentants eux-mêmes.

Le principe est exactement le même pour les deux variantes de cet algorithme naïf (algorithmes 8 et 9). L'on crée une nouvelle image aux mêmes dimensions que l'image originale (ligne 1). Ensuite, on parcours les pixels de l'image d'indices filtrée. Pour chaque pixel (un indice entier), on recherche le label désigné par cet indice (ligne 4 et 5) et on obtient la valeur de l'espace des valeurs de l'image originale correspondante (ligne 6). Cette dernière opération est effectuée en temps constant dans la mesure où la bijection *valeurs de l'espace originale de l'image de densité non nulle dans l'histogramme* $\Leftrightarrow$ *labels* est stockée en mémoire. On notera l'ajout d'une instruction supplémentaire dans l'algorithme 9 qui est l'obtention de la classe d'équivalence du label extrait et de son représentant de classe (ligne 6 de l'algorithme 9).

Algorithm 8: DIRECT_RECONSTRUCTION_LABELS($filtered\_index\_image$, $labels$)

Input: $filtered\_index\_image$: filtered index image obtained when building the MCT,
$labels$: list of all computed labels

Output: $reconstructed\_image$: reconstructed image (original value space)

```
image_reconstructed ← NewImage(filtered_index_image.shape)
for line i in filtered_index_image.width do
    for column j in filtered_index_image.height do
        index_of_labels ← filtered_index_image[i][j]
        label ← labels[index_of_labels]
        value ← GetOriginalValue(label)
        reconstructed_image[i][j] ← value
return reconstructed_image
```

Algorithm 9: DIRECT_RECONSTRUCTION_REPR($filtered\_index\_image$, $labels$)

Input: $filtered\_index\_image$: filtered index image obtained when building the MCT,
$labels$: list of all computed labels

Output: $reconstructed\_image$: reconstructed image (original value space)

```
image_reconstructed ← NewImage(filtered_index_image.shape)
for line i in filtered_index_image.width do
    for column j in filtered_index_image.height do
        index_of_labels ← filtered_index_image[i][j]
        label ← labels[index_of_labels]
        representative ← GetRepresentativeOfClass(GetEquivalenceClass(label))
        value ← GetOriginalValue(representative)
        reconstructed_image[i][j] ← value
return reconstructed_image
```

4.6.2 Reconstructions indirectes

Deux types de reconstructions indirectes sont proposées ici : il s'agit de la reconstruction dite **moyenne** et de la reconstruction dite **médiane**. Comme dit précédemment, ces méthodes de reconstruction indirectes se basent sur le parcours des nœuds de l'arbre des coupes multivalué et non un parcours de l'image d'indices filtrée.

Les deux méthodes de reconstruction indirectes peuvent être effectuées à l'aide d'un seul et même algorithme, et ce à condition de remplacer une instruction (calcul de moyenne) par une autre (calcul de médiane). Pour plus de lisibilité, un seul algorithme générique est présenté ici. Néanmoins, il est important de remarquer que le parcours des nœuds implique une divergence de cas suivant le statut des nœuds. Par statut des nœuds l'on désigne en réalité deux catégorisations concomitantes ; la distinction classique entre les **nœuds intermédiaires** et les **feuilles** mais également la distinction entre les **nœuds conservés** et les **nœuds supprimés** par le filtrage. Il y a donc quatre cas différents à étudier.

Le principe de base de ces méthodes indirectes résident dans le fait que, à chaque niveau de l'arbre (nœud), l'information à reconstruire dépend de la combinaison des informations reconstruites aux niveaux inférieurs. On voit bien la structure récursive de l'algorithme se dessiner. Pour chacun des quatre cas, un algorithme est présenté.

Les premier cas est aussi le plus trivial. On considère la cas d'une **feuille conservée** après filtrage (algorithme keptleaf). Étant donné que le nœud considéré est une feuille, il n'a aucun descendant, et de fait, l'information à reconstruire ne dépend que du nœud lui-même. Il n'y a pas de moyenne (ou médiane) à calculer dans la mesure où chaque nœud ne contient qu'un seul et unique label. Dans ce cas précis, il suffit d'effectuer la conversion *label* vers *valeur de l'espace des valeurs de l'image originale* (ligne 3) comme lors d'une reconstruction directe puis d'affecter la valeur ainsi obtenue aux pixels contenus dans ledit nœud (lignes 4 et 5). Enfin, il faut transmettre l'information recrée aux niveaux supérieurs c'est-à-dire aux parents. Cette information est transmise sous la forme de deux listes. La première liste nommée **weighted_values** est une liste de tuples composés d'un entier indiquant le nombre de pixels concernés par le label traité et la valeur associée à ce label. Puisqu'ici on est en présence d'un nœud feuille, il n'y a qu'un seul tuple dans la liste (ligne 6). La seconde liste est la liste des pixels auxquels il faudra affecter une valeur lorsque l'on traitera un nœud non supprimé. Le nœud en présence étant une feuille conservée, aucun pixel du nœud n'y est ajouté ; on transmet donc une liste vide car les pixels du nœuds on déjà été affectés (lignes 4 et 5).

Function keptleaf(*node*, *labels*, *image*)

Input: *node* : node of the MCT to process, *labels* : list of all computed labels, *image* : reconstructed image (to fill)

Output: *weighted_values* : list of couples of type (label, number), *pixels* : list of pixels to be assigned a label

```
1 weighted_values ← []
2 pixels ← []
3 original_value ← labels[node.original_value].value
4 for pixel p in node.pixels do
5     image[p] ← original_value
6 current_tuple ← MakePair(node.nb_pixels, original_value)
7 weighted_values.append(current_tuple)
8 return weighted_values, pixels
```

Le second cas est celui d'une **feuille supprimée** présenté par l'algorithme removedleaf. Le traitement est similaire à celui d'une feuille conservée. La conversion vers la valeur correspondant au label du nœud est effectuée (ligne 3), le tuple contenant le nombre de pixels du nœud et la valeur associée est crée (ligne 4) et constitue la liste de tuples (ligne 5). La seule différence réside dans le fait que le nœud est supprimé et que les pixels ne sont pas affectés et sont donc ajoutés dans la liste de pixels transmise aux parents (ligne 6).

Function removedleaf(*node*, *labels*, *image*)

Input: *node* : node of the MCT to process, *labels* : list of all computed labels, *image* : reconstructed image (to fill)

Output: *weighted_values* : list of couples of type (label, number), *pixels* : list of pixels to be assigned a label

```
1 weighted_values ← []
2 pixels ← []
3 original_value ← labels[node.original_value].value
4 current_tuple ← MakePair(node.nb_pixels, original_value)
5 weighted_values.append(current_tuple)
6 pixels ← nb_pixels
7 return weighted_values, pixels
```

Le troisième cas est celui d'un **nœud intermédiaire supprimé** présenté par l'algorithme removednode. Cette fois-ci, il faut tenir compte de l'information reconstruite provenant des fils. Cette information est présente dans deux listes ; une contenant les pixels à affecter et une contenant les tuples caractérisant toutes les informations reconstruites récursivement depuis les feuilles. Ces deux informations sont récupérées localement (lignes 3 à 5). Ensuite, la conversion classique du label local à sa valeur équivalente dans l'image originale est effectuée (ligne 6). Ici le nœud est supprimé ce qui signifie que les pixels locaux ne seront pas affectés à cette étape mais seulement lorsqu'un nœud parent non supprimé sera traité. Il s'agit donc d'ajouter un nouveau tuple caractérisant les pixels locaux (ligne 7), de l'ajouter à la liste des tuples récupérée précédemment (ligne 8) et d'ajouter les pixels eux-mêmes à la liste des pixels également récupérée précédemment (ligne 9). Pour terminer ces deux listes sont transmises au parent du nœud faisant ainsi remonter l'information tout en l'enrichissant en y ajouter les détails locaux.

Function removednode(*node*, *labels*, *image*)

Input: *node* : node of the MCT to process, *labels* : list of all computed labels, *image* : reconstructed image (to fill)

Output: *weighted_values* : list of couples of type (label, number), *pixels* : list of pixels to be assigned a label

```
1  weighted_values ← []
2  pixels ← []
3  for child node c in node.childs do
4  |  child_value, child_pixels ← IndirectReconstruction(c, labels)
5  |  Merge(weighted_values, pixels, child_values, child_pixels)
6  original_value ← labels[node.original_value].value
7  current_tuple ← MakePair(node.nb_pixels, original_value)
8  weighted_values.append(current_tuple)
9  pixels ← nb_pixels
10 return weighted_values, pixels
```

Le dernier cas est celui d'un **nœud intermédiaire conservé** présenté par l'algorithme keptnode. Comme pour un nœud intermédiaire supprimé il est nécessaire de récupérer les informations combinées des nœuds descendants (lignes 3 à 5). Comme pour les trois autres cas, la conversion vers la valeur correspondant au label du nœud est effectuée (ligne 6). Étant donné que le nœud traité est conservé, il faut calculer la valeur à attribuer. Celle-ci dépend à la fois de la valeur locale (obtenue à la ligne 6), mais aussi de toutes les valeurs pondérées obtenues précédemment (lignes 3 à 5). La valeur finale est calculée en moyennant (ou en choisissant la médiane) les valeurs pondérés (weighted_values à la ligne 7). Ceci donne la valeur à attribuer à tous les pixels récupérés y compris les pixels locaux (lignes 8 et 9). De fait, la liste des pixels à attribuer devient vide (ligne 12). Il est important de noter qu'au niveau des valeurs pondérées, la liste n'est surtout pas vidée car il faut conserver la richesse de l'information qui y est présente. On se contente simplement d'y ajouter un nouveau tuple contenant le nombre de pixels du nœud local (pas la somme des pixels affectés) et la nouvelle valeur moyenne (ou médiane) calculée et utilisée juste avant (ligne 10 et 11). Pour finir, et ce comme pour chacun des quatre cas, les deux listes sont transmises au parent (ligne 13).

Function keptnode(*node*, *labels*, *image*)

Input: *node* : node of the MCT to process, *labels* : list of all computed labels, *image* : reconstructed image (to fill)

Output: *weighted_values* : list of couples of type (label, number), *pixels* : list of pixels to be assigned a label

```
1  weighted_values ← []
2  pixels ← []
3  for child node c in node.childs do
4  |  child_values, child_pixels ← IndirectReconstruction(c, labels)
5  |  Merge(weighted_values, pixels, child_values, child_pixels)
6  original_value ← labels[node.original_values].value
7  value ← Mean(weighted_values, node.nb_pixels, original_value)
8  for pixel p in node.pixels do
9  |  image[p] ← value
10 new_tuple ← MakePair(node.nb_pixels, value)
11 weighted_values.append(new_tuple)
12 pixels ← []
13 return weighted_values, pixels
```

L'algorithme 10 présente est l'algorithme général de reconstruction indirecte. À noter cependant qu'il est fait abstraction du type précis de la reconstruction (moyenne ou médiane). L'algorithme permet une reconstruction de l'image via un parcours récursif des nœuds de l'arbre des coupes multivalué. Le processus étant récursif du fait des relations de dépendances quant aux calculs des valeurs, l'appel initial est effectué sur la racine (ligne 1) qui sera alors le dernier nœud dont les pixels seront affectés d'une valeur moyenne (ou médiane). Il est à noter que l'image reconstruite (*reconstructed_image* dans l'algorithme) est remplie par effets de bord. Il est également à noter que *indirect_reconstruction* est une fonction dont l'algorithme n'est pas présenté ici. En réalité, cette fonction cache tout simplement un appel à une des quatre fonctions résolvant chacune un cas précis de reconstruction suivant le type du nœud. Cette fonction se réduit de fait à appeler :

- la fonction **keptleaf** pour un nœud de type feuille et dont le label filtré est resté identique au label original
- la fonction **removedleaf** pour un nœud de type feuille et dont le label filtré diffère du label original
- la fonction **keptnode** pour un nœud intermédiaire et dont le label filtré est resté identique au label original
- la fonction **removednode** pour un nœud intermédiaire et dont le label filtré diffère du label original

Algorithm 10: INDIRECT_RECONSTRUCTION(*root*, *labels*, *original_image_shape*)

Input: *root* : root node of the MCT, *labels* : list of all computed labels,
original_image_shape : shape of the original image
Output: *reconstructed_image* : reconstructed image (original value space)

1 $reconstructed\_image \leftarrow NewImage(original\_value\_shape)$
2 $IndirectReconstruction(root, labels, reconstructed\_image)$
3 **return** *reconstructed_image*

5 Travail réalisé

Cette section décrit les différents travaux réalisés dans le cadre du stage jusqu'ici dans l'ordre chronologique. Les différents problèmes, les raisonnements, la solutions proposées y compris les versions intermédiaires non présentes dans la méthode proposée dans la section précédente sont présentés ici. Cette section est écrite avec un style plus personnel, et tente de décrire l'expérience vécue et les cheminements de pensée.

5.1 Lancement du projet

La première étape de projet a été d'étudier la structure d'arbre des coupes multivalué notamment à travers la lecture de [10]. Ayant travaillé dans le cadre du TER de M1 sur l'implémentation de la variante $\ddot{\mathfrak{G}}$ du graphe des coupes [20], [21], j'étais déjà familier des structures d'arbre et de graphe des coupes. La compréhension de l'extension (arbre des coupes multivalué) en a donc été facilitée.

Un code déjà existant et permettant notamment de construire ledit arbre sur des images en niveaux de gris était disponible [22]. L'étude de la structure et des algorithmes s'est donc poursuivie à travers une étude du code. Le projet est développé en Python [3], un langage de programmation que je n'avais alors par encore pratiqué. J'en ai de fait profité pour me former au langage Python (dans sa version 3 exclusivement) en parallèle de l'étude du code. Cela m'a donc permis d'apprendre un nouveau langage orienté objet, d'ajouter une nouvelle corde à mon arc pourrait-on dire. Du fait des connaissances acquises à l'université et en dehors concernant les langages objets, l'apprentissage du Python a été rapide et efficace.

5.2 Développements pour les images en niveaux de gris

La première tâche de développement a principalement consisté à réorganiser le code en séparant les fonctions et en les encapsulant dans diverses classes afin de supprimer les recours aux variables globales dont l'utilisation en Python est assez sujette à confusion.

En terme de développement pur, l'histogramme calculé à l'aide du module **Numpy**, n'était pas complet. En effet, les densités nulles ne sont pas ajoutées par le module, laissant survenir des erreurs d'indexation liées aux tailles fixes des tableaux (listes en Python). J'ai donc proposé une solution corrective insérant et reconstruisant les densités nulles au sein de l'histogramme résolvant ainsi ce problème qui, pour ainsi dire, ne survenait pas constamment.

Enfin pour finir avec cette session de développements, j'ai étoffé les critères de filtrage de l'arbre des coupes multivalué en ajoutant la possibilité de rendre le filtre non croissant. Jusqu'alors, un critère unique servait au filtrage par exemple *suppression des nœuds dont l'aire est* < 50 *pixels*. J'ai ajouté la possibilité d'utiliser deux bornes avec deux critères de liaison (*et* et *ou*). Ainsi il était possible d'utiliser un critère tel que *suppression des nœuds ayant une aire* < 50 *ou une aire* > 100. Une vérification a été nécessaire afin de garantir que la politique de réduction était bien respectée (la politique choisie était directe).

5.3 Développements pour les images multivaluées

À partir ce de moment, on disposait d'une solution effective pour les images en niveaux de gris avec plusieurs critères de filtrage possibles. Il était alors possible d'étendre le traitement aux images multivaluées.

L'espace de valeurs utilisé pour cette première extension est l'espace RVB (Rouge-Vert-Bleu). L'histogramme est cette fois-ci calculé en utilisant le module **OpenCV** qui permet de calculer des histogrammes pour des images couleurs. Le problème définit pour l'histogramme en niveaux de gris n'est pas présent avec OpenCV : l'histogramme est donné avec mes densités nulles. Inutile donc d'ajouter un correctif.

Un autre problème se pose en revanche. Effectivement, l'espace RVB $([\![0, 255]\!], [\![0, 255]\!], [\![0, 255]\!])$ est bien plus dense qui l'espace $[\![0, 255]\!]$ des niveaux de gris. Le nombre élevé de valeurs de cet espace (2^{24}) rend le calcul du max-tree sur l'histogramme relativement long ; après tout le code est développé en Python et sans optimisations à ce moment-là. Un problème annexe pointe le bout de son nez à l'étape suivante : la construction de l'ordre hiérarchique bas. Le nombre élevé de nœuds (labels) à créer ne tient pas en mémoire, sans parler du calcul de la fermeture transitive qui devient simplement utopique. À titre informatif avec la taille de la structure de données de nœud utilisée et le nombre de labels crées (tous les labels sont créés donc 2^{24}) il aurait fallu 256 To de RAM rien que pour stocker les relations... Il fallait donc trouver des solutions pour remédier à ces problèmes.

Une solution unique n'étais pas envisageable ; plusieurs solutions (une par problème) étaient plus appropriées.

Pour réduire le temps de calcul du max-tree (mais aussi de l'extraction sous forme d'arbre ultérieure), j'ai réduit le nombre d'indices utilisés lors du calcul de l'histogramme. Le facteur de réduction (limité pour des raisons d'indexation) peut alors être un nombre, puissance de 2 (1, 2, 3, 8, 16, 32, 64 ou 128). les couleurs sont alors réduites et fusionnées dans des bandes (des cubes dans l'espace RVB) dont la largeur est alors le facteur de subdivision paramétrable. Pour mettre en place cette réduction, il a été nécessaire de développer une classe permettant d'effectuer la corrélation en les indices RVB classiques $[\![0, 255]\!]$ et les indices réduits après application du facteur de subdivision $[\![0, \frac{256}{coefficient\_subdivision} - 1]\!]$.

Au niveau de l'ordre hiérarchique, l'idée a été de purement et simplement de ne pas créer les labels correspondant à une densité nulle. Seuls les labels ayant une densité non nulle sont créés et ceux ayant une densité nulle sont réunis dans une classe d'équivalence des densités nulles placés à la racine de l'ordre. De fait le calcul de la fermeture transitive et donc le stockage des relations $<$ entre labels était résolu par la même occasion.

Il a bien fallu adapter les méthodes de construction de l'image d'indices (utilisées pour calculer l'arbre des coupes multivalué), de reconstruction de l'image dans son espace de valeurs originales et enfin l'algorithme de calcul de l'arbre des coupes pour fonctionner correctement avec les précédentes adaptations.

Par la suite, l'ajout d'éléments structurants lors du calcul du max-tree de l'histogramme pour plus de flexibilité a été émise. J'ai donc développé pour la méthode RVB trois éléments struc-

turants agissant sur le calcul du max-tree de l'histogramme. Pour être plus précis, ces éléments structurants peuvent être utilisés en remplacement du traditionnel voisinage (1-voisinage). Jusqu'alors, et c'est toujours la solution par défaut, le voisinage RVB est simplement les six voisins, combinaisons des facteurs $R+1$, $R-1$, $V+1$, $V-1$, $B+1$, $B-1$.

Les éléments structurants ajoutés sont les suivants :
- une **balle** de rayon r
- un **cube** de côté c
- une **croix** de longueur d'axe l

Afin d'explorer un autre espace de couleurs, le modèle HSV (Hue Saturation Value) a ensuite été considéré. D'autres modèles plus proches de la perception visuelle humaine pourront être étudiés par la suite. Le choix du modèle HSV est principalement dû au fait que le traitement s'effectuera uniquement sur le teinte (bande H ou Hue) ce qui en fait une solution rapide à développer et tester car assez similaire en complexité de traitement à la solution en niveaux de gris.

Afin de préparer au développent de ce nouvel espace, j'ai effectué une refonte des classes du projet. Des classes abstraites ont été développées et les implémentations de ces classes sont définies pour chaque espace de valeurs différent. Pour ce qui est de l'espace HSV, le max-tree est calculé sur la teinte (H) et les saturation et valeurs (S et V) sont stockées en mémoire. Autrement dit, le filtrage d'attributs s'effectue uniquement sur des critères de teinte de même que l'ordre hiérarchique bas ne dépend que des densités de teintes.

5.4 Développements génériques

En parlant d'attributs, il n'y en avait à l'origine qu'un seul : l'aire. L'aire est pratique à calculer car elle peut être mise à jour lors de l'inondation récursive pendant la construction même de l'arbre des coupes multivalué (voir l'algorithme 5 et la fonction flood).

Deux autres attributs ont alors été définis : le périmètre ainsi que la compacité.

5.4.1 Périmètres

Dans notre cas, il y a deux manières de considérer le périmètre ou devrais-je simplement dire que l'on peut considérer deux types de périmètres...

Pour bien illustrer ces considérations, il est important d'en revenir aux fondamentaux. Un nœud contient un certain nombre de pixels non nul après application de la fonction 6. Ces pixels représentent une zone plate (un composante connexe). Cet ensemble représente une et une seule forme mais celle-ci peut tout à fait contenir des **trous**. Cela signifie que l'on peut différencier deux périmètres : le périmètre extérieur (le *vrai* périmètre) et la somme des périmètres.

Le périmètre classique est obtenu en extrayant le contour extérieur de la forme résultant des pixels du nœud considéré. La somme des périmètres, comme son nom l'indique, est obtenue en extrayant tous les contours de la forme (y compris le contour des trous) et en sommant les valeurs de ces différents périmètres.

Une première idée a été de développer une classe BinaryShape permettant d'attribuer à chaque nœud une image binaire représentant la forme contenue par le nœud. À chaque ajout d'un pixel dans un nœud de l'arbre des coupes multivalué dans la fonction flood, le pixel est ajouté dans un tableau représentant la forme du nœud à cet instant. Divers calculs sont effectué pour faire en sorte que la forme soit représentée sur un ensemble plus petit que l'image traitée. La forme contenue dans le tableau aura pour dimensions la taille de la forme + 2 car un contour (contenant 0) est ajouté pour garantir que la méthode d'extraction du contour utilisée ultérieurement puisse produire un résultat correct. J'ai fait en sorte que l'ajout d'un pixel via ses coordonnées générales dans l'image traitée soient converties en coordonnées locales de la forme binaire. Il s'agit ni plus ni moins d'un changement de repère.

La bordure est obtenue à l'aire d'une convolution et le périmètre calculé à partir d'une seconde convolution telles que définies dans [19]. Malheureusement, du fait de la nature des images réelles, cette technique ne s'est pas montrée très concluante. Les images réelles présentant des zones plates relativement peu étendues (beaucoup de zones ne contiennent en effet qu'un ou deux pixels), la méthode est inefficace sur ces très petites formes.

Une autre méthode, utilisant codage progressif du bord (codage de Huffman) a été envisagée, mais celle-ci est inapte à traiter les formes trouées or c'est un cas de figure fréquent. Finalement la solution retenue a été de simplement compter le nombre de pixels de la bordure obtenue. Cette solution donne à ce jour une meilleure estimation que les autres méthodes testées.

Il faut distinguer l'extraction du périmètre extérieur et celui de la somme des périmètres. En

effet, pour extraire le périmètre extérieur il est indispensable de s'assurer que la forme binaire ne comporte pas de trous et, le cas échéant, les combler. Après cette étape seulement le périmètre extérieur peut être calculé en comptant le nombre de pixels restants.

Il est donc possible d'extraire au choix le périmètre extérieur ou bien la somme des périmètres. Cependant, il faut noter que c'est un processus relativement couteux en temps. Il faut tenir compte du fait qu'il faille ajouter les pixels à un forme binaire lors du calcul de l'arbre coupes multivalué et ce pour chaque pixel de l'image traitée. Lors de chacun de ces ajouts, des calculs inhérents au changement de repère sont effectués ajoutant de nombreuses instructions supplémentaires. On peut aussi noter qu'il est nécessaire de disposer d'un espace mémoire élargi puisqu'il faut stocker ces formes binaires (tableaux bidimensionnels). Enfin il faut réaliser un convolution par forme binaire (et donc par nœud) puis compter les pixels obtenus. J'ai alors décidé de proposer deux versions de l'algorithme 5 et de la méthode flood : une comprenant le calcul d'un des deux périmètres (le choix est laissé à l'utilisateur) et une version ne calculant pas ces dernier (celle présentée dans la section précédente).

5.4.2 Compacité

La compacité est un ratio entre l'aire et le périmètre de l'ordre de $\frac{4*\pi*Area}{perimeter^2}$.

Calculer la compacité est trivial à partir du moment où l'on dispose de l'aire et du périmètre. Grâce au structures **BinaryShape** et **BinaryContour** décrite au-dessus, il est possible d'obtenir le périmètre. L'aire quant à elle, est depuis le début calculée dans la mesure où c'est simplement le nombre de pixels présents dans le nœud qui est un attribut du même nœud.

C'est le périmètre extérieur qui est choisi pour calculer la compacité. Cela implique que le choix de filtrer l'arbre des coupes multivalué sur l'attribut **compacité** nécessite de calculer au moins le périmètre extérieur.

5.4.3 Histogramme

Il y avait encore quelques problèmes au niveau de l'histogramme sur lequel repose toute la méthode. Si l'image de départ est bruitée, l'histogramme discret ne donne pas la bonne estimation de densités. Afin de remédier à ce problème, j'ai appliqué une fenêtre de Parzen [18] afin de lisser l'histogramme et d'obtenir une meilleure estimation de densité de probabilités.

Il faut bien évidemment préciser que ces développements sont à spécifier pour les différents espaces de valeurs utilisés c'est-à-dire l'espace des niveaux de gris, l'espace RVB et enfin l'espace HSV. Concernant l'espace des niveaux de gris et l'espace HSV, et ce du fait que seule la bande de teinte (H ou hue) du modèle HSV est utilisée, il s'agit ni plus ni moins d'un histogramme unidimensionnel. La fenêtre de Parzen dans ce deux cas-là est donc réellement une fenêtre dont la largeur est paramétrable. Elle agit de fait sur le n-voisinage unidimensionnel. La fenêtre est de fait de largeur $2 * n + 1$ (n pixels voisins de chaque côté plus un pixel central).

Pour l'espace RVB, il s'agit d'un cube plutôt qu'une fenêtre dans la mesure où l'on est en présence d'un espace tridimensionnel. La fenêtre (ou le cube) représente le n-voisinage tridimensionnel et la taille du cube est alors $2 * n + 1$ par $2 * n + 1$ par $2 * n + 1$. Il faut toutefois prendre en compte le

facteur de subdivision permettant de réduire taille de l'histogramme (clusterisation). n doit être vu non pas comme un nombre de valeurs de l'histogramme mais comme un nombre de clusters dans ce même histogramme. Cette définition est compatible avec la précédente en remarquant que lorsque le facteur de subdivision est égal à 1, n désigne bien un nombre de valeurs dans l'histogramme, chacune étant un cluster contenant une seule et unique valeur. Dans le cas contraire, si le facteur de subdivision est s, alors n désigne n cluster de s valeurs soit $n * s$ valeurs originales de l'histogramme.

5.4.4 Méthodes de reconstruction

Il n'y a pas unicité de reconstruction. En effet, pour obtenir un image résultat, il est nécessaire de convertir l'image de labels obtenue après filtrage (algorithme 7) et de repasser dans l'espace initial des valeurs de l'image.

On peut distinguer deux classes de méthodes de reconstruction : les méthodes directes et les méthodes indirectes.

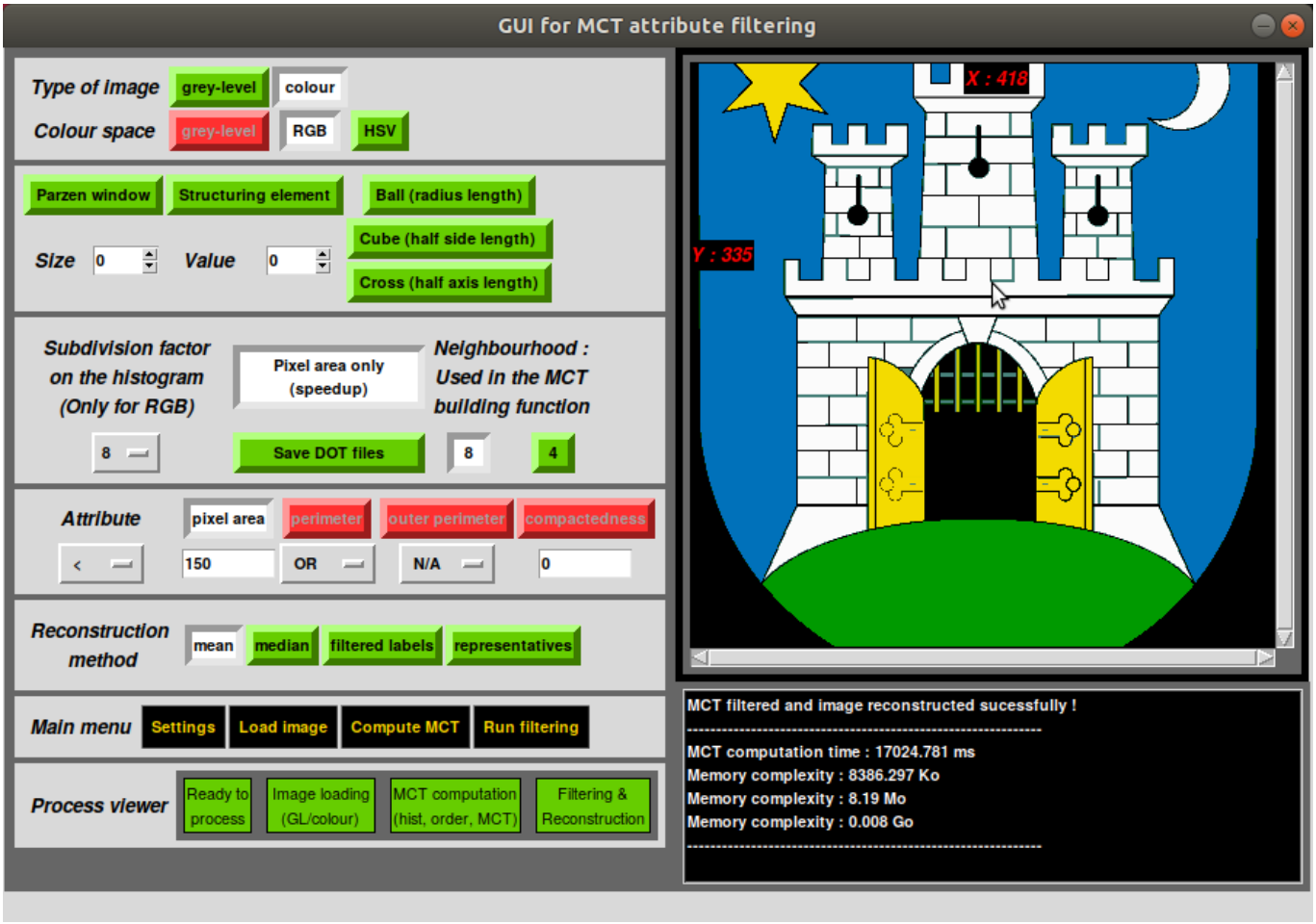
Les méthodes directes sont ce que l'on pourrait appeler les méthodes naïves. Elles utilisent l'image d'indices construite par 7. On peut néanmoins faire une petite distinction entre la méthode purement naïve (voir l'algorithme 8) et la reconstruction par représentant de classes d'équivalence qui lui est en tout point similaire à la seule exception de l'utilisation d'un représentant de classe plutôt que du label lui-même (voir l'algorithme 9).

Les méthodes indirectes n'utilisent pas l'image d'indices filtrée mais l'arbre des coupes multivalué. Elle nécessitent plus de réflexion quant à la manière de traiter l'arbre. C'est un des raisons pour explique la présentation fragmentée fonctions traitant chacune un cas précis de nœud (voir les fonctions keptnode, keptleaf, removednode, et removedleaf).

5.4.5 Interface graphique

Afin de rendre l'utilisation de la méthode proposée plus simple, un travail de développement supplémentaire consistait à développer interface graphique réunissant tous les paramètres et tous les espaces de valeurs disponibles. Les options étant devenues nombreuses, et certaines n'étant que spécifiques à certains espaces de valeurs contrairement à d'autres étant génériques, il devenait fastidieux de lancer le programmes en ligne de commande. Une interface graphique a alors été développée en utilisant **TKinter** le module de création d'interfaces basique de Python.

Les différents espaces de valeurs sont tous inclus dans l'interface permettant une plus grande flexibilité : un seul programme tout en un faisant appel au classes et traitements requis plutôt que trois programmes séparés avec des options confuses. J'ai également ajouté dans l'interface la possibilité de visualiser l'image reconstruite après filtrage permettant une meilleure visualisation des résultats. Quoi qu'il en soit les images résultats sont toujours sauvegardées avec des noms incluant les options et critères de filtrage. L'interface permet également de lancer plusieurs fois le filtrage tout en ne calculant l'arbre des coupes multivalué qu'une seule fois par image. Il est donc plus efficace d'utiliser l'interface lorsque l'on travaille sur une même image mais avec différents critères.



(a) Interface

FIGURE 1 – (a) L'interface graphique développée (menu principal).

6 Résultats

Cette section présente quelques résultats obtenus. Sont présentés notamment des analyses de performances ainsi que des évaluations des différents paramètres de la méthode proposée.

6.1 Considérations

Il y a plusieurs considérations à prendre en compte pour exposer les résultats de la méthode. Tout d'abord, il est intéressant d'effectuer des analyses de performances dans la mesure où il existe plusieurs type de reconstructions possibles et donc potentiellement plusieurs complexités de traitement. De même, la méthode se basant sur l'histogramme des images traitées, il est intéressant d'essayer de corréler la densification de l'histogramme avec la taille de l'arbre des coupes et la complexité temporelle et spatiale.

Dans un second temps, il est intéressant d'exposer les possibilité de filtrage à travers des exemples afin d'illustrer les possibilités offertes par cette nouvelle méthode. Enfin, et pour terminer, il est intéressant de mettre en lumière l'influence des différents paramètres proposés par la méthode.

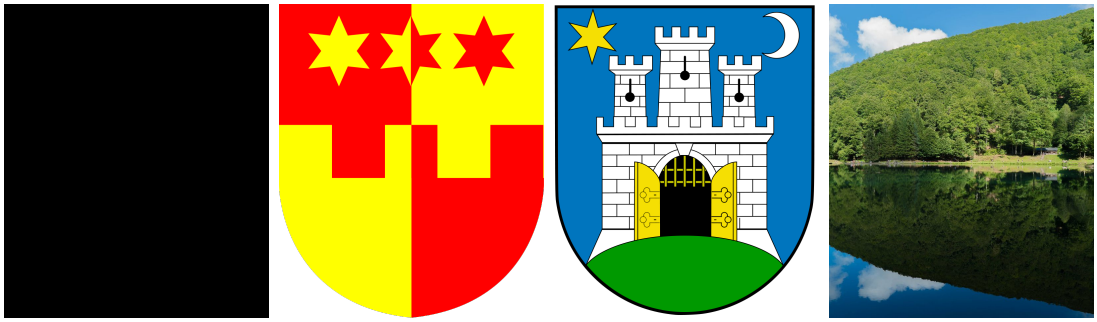
6.2 Analyses de performances

Afin d'analyser les performances de la méthode en terme spatial et temporel, des scripts shell on été écrits afin de générer 20 exécutions de chacun des trois programmes (niveaux de gris, modèle RVB et modèle HSV). Le temps d'exécution total du programme est pris en compte mais aussi le temps de construction de l'arbre des coupes multivalué seul (sans comptabiliser le filtrage et la reconstruction de l'image résultat).

En terme de complexité spatiale, trois structures sont comptabilisées :

- l'arbre des coupes multivalué lui-même c'est-à-dire la somme de tous les nœuds composant l'arbre
- le tableau contenant tous les labels crées et permettant la construction de l'arbre des coupes multivalué
- le tableau bidimensionnel des relations de comparaison ($<$) sur les labels et définissant (à l'aide de max-tree) la relation d'ordre

Différentes images (au nombre de quatre) sont utilisées, de richesse de l'histogramme croissante et donc au nombre de nœuds croissant. Les images utilisées ont toutes les mêmes dimensions (800x946 pixels).



(a) image 1 (b) image 2 (c) image 3 (d) image 4

FIGURE 2 – (a-d) Les images utilisées pour les tests de performances ci-dessous.

6.2.1 Niveaux de gris

Image	reconstruction	nombre de nœuds	temps total (s)	temps MCT (s)	mémoire (Mo)
image 1	label	2	80.7	16.0	6.1
	repr	2	82.0	17.0	6.1
	moyenne	2	88.0	15.4	6.1
	médiane	2	80.6	15.5	6.1
image 2	label	3287	59.4	16.5	7.0
	repr	3287	60.2	15.8	7.0
	moyenne	3287	60.3	15.7	7.0
	médiane	3287	59.7	16.8	7.0
image 3	label	12190	50.7	22.6	8.9
	repr	12190	52.5	22.3	8.9
	moyenne	12190	50.6	21.8	8.9
	médiane	12190	59.8	21.9	8.9
image 4	label	298428	97.3	53.2	68.8
	repr	298428	97.8	53.7	68.8
	moyenne	298428	106.5	53.4	68.8
	médiane	298428	107.3	52.2	68.8

6.2.2 Modèle Rouge Vert Bleu (RVB)

Image	reconstruction	nombre de nœuds	temps total (s)	temps MCT (s)	mémoire (Mo)
image 1	label	2	43.7	7.1	5.8
	repr	2	42.6	7.4	5.8
	moyenne	2	31.0	7.4	5.8
	médiane	2	32.0	7.4	5.8
image 2	label	3287	48.0	8.2	6.4
	repr	3287	46.1	8.3	6.4
	moyenne	3287	34.6	8.0	6.4
	médiane	3287	33.0	8.0	6.4
image 3	label	12190	66.0	17.0	8.2
	repr	12190	63.3	17.0	8.2
	moyenne	12190	51.4	17.4	8.2
	médiane	12190	53.2	17.2	8.2
image 4	label	298428	109.4	33.1	99.0
	repr	298428	111.6	33.8	99.0
	moyenne	298428	436.7	34.6	99.0
	médiane	298428	432.8	36.4	99.0

6.2.3 Modèle Hue Saturation Value (HSV)

Image	reconstruction	nombre de nœuds	temps total (s)	temps MCT (s)	mémoire
image 1	label	2	145.8	15.8	5.9
	repr	2	146.2	16.1	5.9
	moyenne	2	145.0	17.1	5.9
	médiane	2	147.3	16.3	5.9
image 2	label	3287	145.7	15.9	6.6
	repr	3287	139.9	15.9	6.6
	moyenne	3287	134.8	17.7	6.6
	médiane	3287	140.2	17.1	6.6
image 3	label	12190	148.6	16.3	6.3
	repr	12190	145.8	16.4	6.3
	moyenne	12190	149.5	16.4	6.3
	médiane	12190	142.1	16.6	6.3
image 4	label	298428	141.6	25.4	16.9
	repr	298428	151.0	25.7	16.9
	moyenne	298428	140.4	26.4	16.9
	médiane	298428	143.5	26.5	16.9

6.2.4 Observations

Avant toutes choses, il faut noter que les exécutions on été effectuées en activant l'option *area-only* permettant de ne pas calculer les périmètres et de ce fait, de ne pas avoir à générer les structures de *BinaryShape* et de *BinaryContour* qui rendent le traitement bien plus long en fonction du nombre de nœuds de l'arbre des coupes multivalué. Concernant plus spécifiquement le modèle RVB, le paramètre de subdivision a été positionné à 8. Cela implique que l'histogramme traité n'est pas de dimensions $256 * 256 * 256$ mais de $\frac{256}{8} * \frac{256}{8} * \frac{256}{8} = 32 * 32 * 32$.

Quelque soit le modèle analysé, on peut tout de suite noter que le temps de calcul de l'arbre des coupes multivalué (MCT ou Multivalued Component Tree) est proportionnel au nombre de nœuds qu'il comporte. Cela n'est pas visible dans les tableaux proposés, mais il est évident que les dimensions de l'image traitée influent également sur la durée de calcul de l'arbre des coupes multivalué. Ce n'est pas visible ici dans la mesure où les images choisies sont de mêmes dimensions.

Au niveau des méthodes de reconstruction, on peut remarquer que les temps de calculs généraux (mais aussi les temps de calculs du MCT) montrent une séparation de méthodes de reconstruction **directes** (label et repr) et des méthodes de reconstruction **indirectes** (moyenne et médiane).

La complexité mémoire, c'est-à-dire le stockage des nœuds de l'arbre des coupes multivalué, des labels crées ainsi que des le stockage des relations $<$ pré-calculées, est elle aussi proportionnelle au nombre de nœuds comme l'on pouvait s'en douter. Néanmoins, cela dépend aussi de la densité de l'histogramme étant donné que celle-ci influe sur le nombre de labels crées.

L'image libellée 4 est une photographie réelle et non une image que l'on pourrait qualifier de schématique (histogramme peu dense, peu de nœuds). On constate une certaine explosion temporelle, notamment au niveau des méthodes de reconstruction indirectes, et ce uniquement pour le modèle RVB. Cela peut s'expliquer par la plus grande taille de l'espace de valeurs utilisé.

Pour terminer, il faut noter que les calculs de périmètres (ou de compacité) ne sont pas effectués ici mais ils apportent un explosion temporelle (mémoire dans on bien moindre mesure) extrêmement conséquente. Cela est en relation directe avec l'explosion du nombre de nœuds de l'arbre.

6.3 Illustrations du filtrage

Cette section essaye de présenter divers résultats de filtrage. Y sont notamment exposés, les différents critères de filtrage. Enfin quelques cas notables sont montrés.

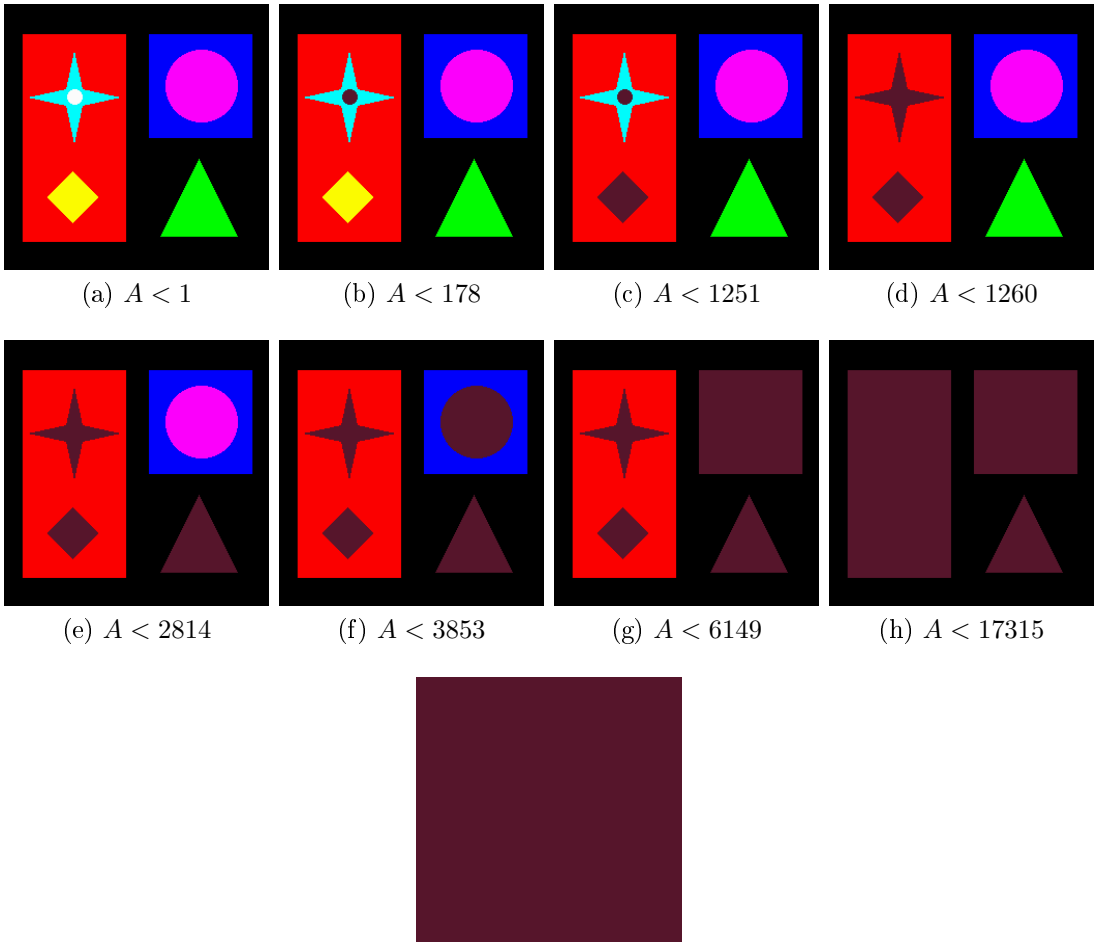
6.3.1 Simplification de l'image

Voici une illustration présentant les possibilités offertes par un filtrage pour lequel le critère consiste à supprimer les nœuds de l'arbre qui sont inférieurs à une certaine valeur en terme d'aire (nombre de pixels).

L'image (a) est l'image originale et contient sept formes différentes. Ici il s'agit d'une image factice fabriquée dans le but d'illustrer les possibilités offertes par le critère de filtrage choisi.

Les images (b) à (i) sont les résultats des filtrages successifs donc la valeur de l'aire est indiquée en légende. L'on supprime de fait progressivement les zones plates (ou composantes connexes) de l'image. Le processus de filtrage et de reconstruction fait en sorte que le plus proche antécédent non supprimé soit attribué au nœuds supprimés. Intuitivement, on voit donc un effet dit de **simplification** de l'image. Les détails sont progressivement *effacés* pour se fondre dans des ensembles plus génériques. Bien sûr, et ce comme pour toutes les structures de représentations hiérarchiques des images, aucun contour n'est modifié ou supprimé.

Cet effet de simplification permet par exemple de faire disparaitre le disque blanc visible au centre de l'étoile (a) qui prend alors la valeur médiane associée à la classe d'équivalence contenant le support de l'image (i). Autre exemple, on voir apparaître simplement le rectangle, le carré et le triangle sans les formes contenues en leurs seins.



(a) $A < 1$ (b) $A < 178$ (c) $A < 1251$ (d) $A < 1260$

(e) $A < 2814$ (f) $A < 3853$ (g) $A < 6149$ (h) $A < 17315$

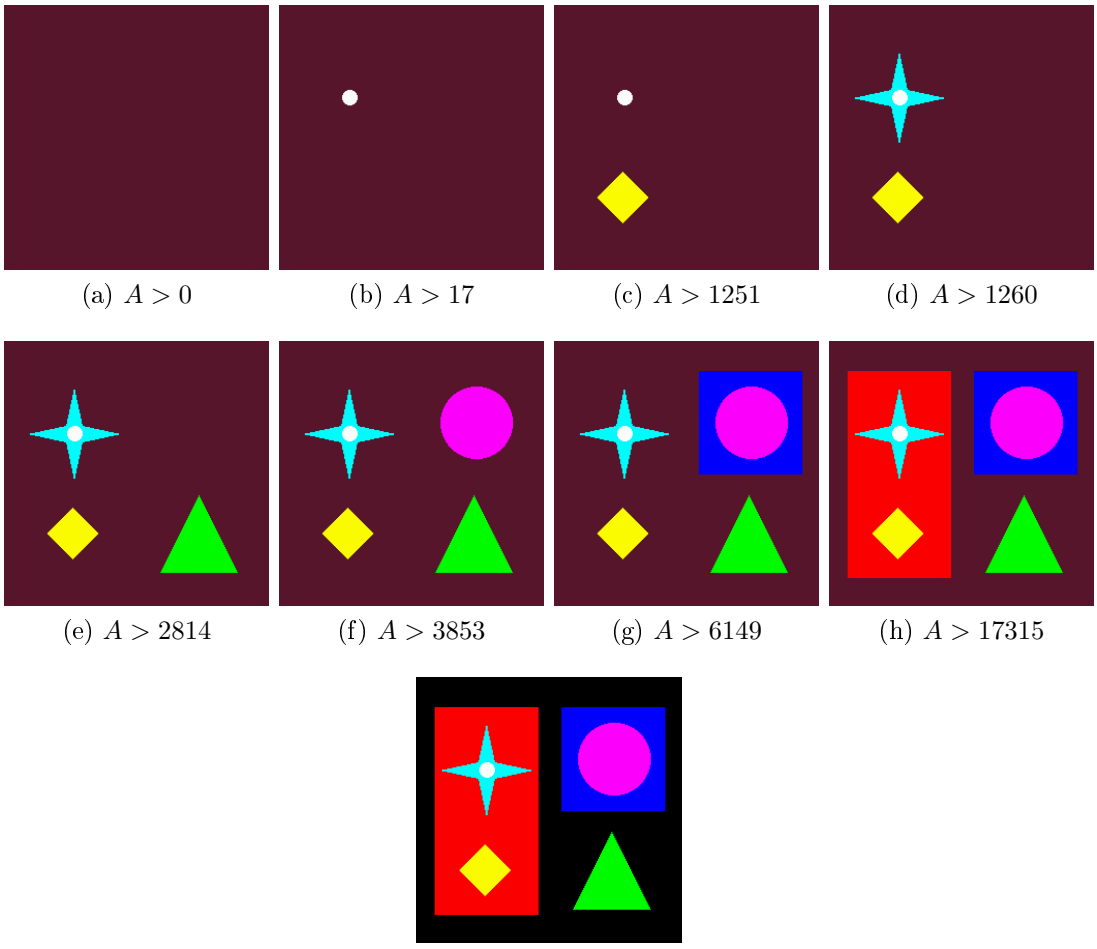
(i) $A < 32724$

FIGURE 3 – (a-i) Chaque image est le résultat d'un filtrage dont les nœuds inférieurs à la description sont supprimés (la reconstruction est une reconstruction moyenne).

6.3.2 Extraction des détails

On peut explorer le processus inverse c'est-à-dire supprimer les nœuds dont l'aire est supérieure à une certaine valeur. Évidemment, on réalise ici le processus inverse, faisant disparaitre les formes génériques au profit des détails.

L'on peut extraire le fameux disque (b) qui dans le cadre de cette image est le plus petit détail présent. On fait apparaître progressivement les détails, ajoutant progressivement les formes le plus grandes jusqu'à reconstruire entièrement l'image originale (i).



(a) $A > 0$ (b) $A > 17$ (c) $A > 1251$ (d) $A > 1260$

(e) $A > 2814$ (f) $A > 3853$ (g) $A > 6149$ (h) $A > 17315$

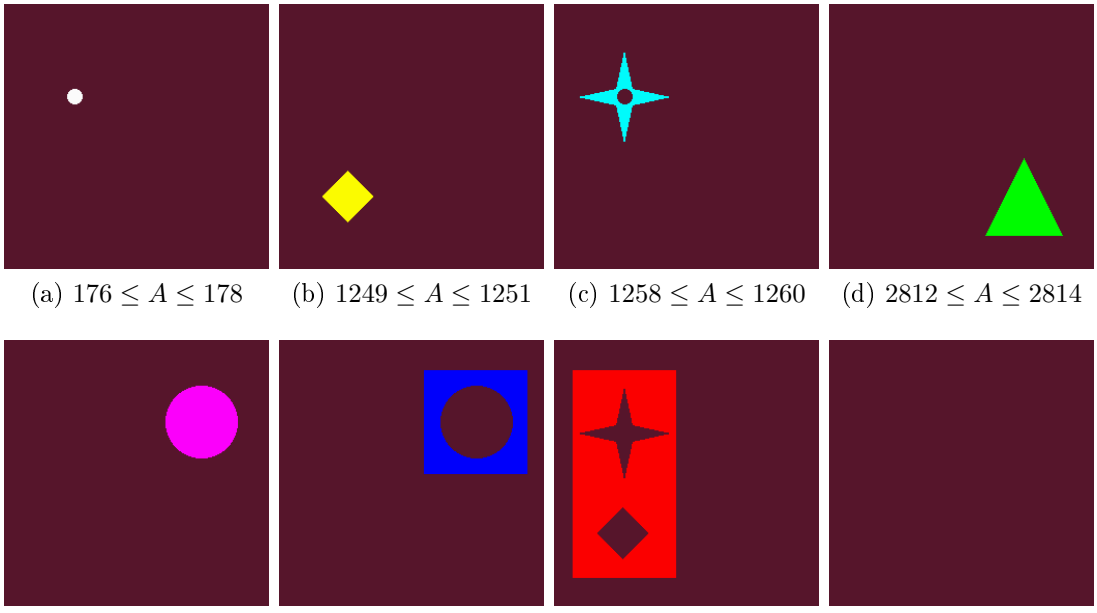
(i) $A > 32724$

FIGURE 4 – (a-i) Chaque image est le résultat d'un filtrage dont les nœuds supérieurs à la description sont supprimés (la reconstruction est une reconstruction moyenne).

6.3.3 Extraction des formes

Il est possible de combiner les deux critères précédents pour extraire des formes inclues dans un certain intervalle. En supprimant les nœuds dont l'aire est inférieure à une certaine valeur et les nœuds dont l'aire est supérieure à une certaine autre valeur (supérieure à la première) il est désormais possible d'extraire les formes individuellement.

Les formes individuelles (b à g) ne sont pas possible à obtenir avec les critères précédents. On peut noter que le disque blanc (a) lui était possible à obtenir de par sa nature de forme d'aire minimale. La même remarque peut être faite pour le support de l'image (h) qui lui est possible à obtenir par tous les critères.



(a) $176 \leq A \leq 178$ (b) $1249 \leq A \leq 1251$ (c) $1258 \leq A \leq 1260$ (d) $2812 \leq A \leq 2814$

(e) $3851 \leq A \leq 3853$ (f) $6147 \leq A \leq 6149$ (g) $17313 \leq A \leq 17315$ (h) $65535 \leq A \leq 65537$

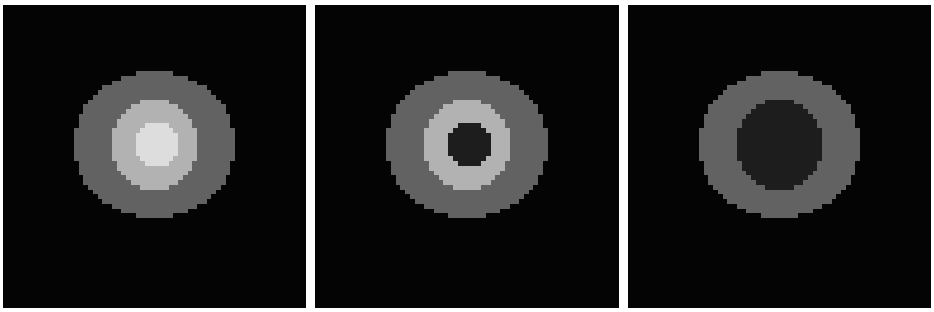
FIGURE 5 – (a-h) Chaque image est le résultat d'un filtrage dont les nœuds ne respectant pas la description sont supprimés (la reconstruction est une reconstruction moyenne).

6.3.4 Cas particulier

Il existe un cas particulier que l'on peut mettre en lumière. Avec les autres structures permettant une représentation hiérarchique d'une image (voir la section 3 sur les structures) il n'est pas possible d'extraire une forme intermédiaire par un seul filtrage ou du moins par une seule passe.

Dans la mesure où la méthode proposée, de par sa construction basée sur l'histogramme et les relation de voisinage, permet de placer dans les feuilles de l'arbre des coupes multivalué, des formes dont le label associé est un pic dans ce même histogramme. De fait, des formes qui, classiquement dans d'autres structures seraient placées dans des nœuds intermédiaires en lien avec les relations de voisinage ou de lignes de niveaux, sont ici placées dans des feuilles de l'arbre. Ceci permet de les extraire comme l'on peut extraire par exemple une forme de taille minimale.

L'image (f) montre l'extraction du disque intermédiaire. Il y a trois disques imbriqués les uns dans les autres. Avec une structure classique, l'on pourrait extraire le disque central ou bien le disque extérieur mais on ne pourrait dissocier le disque intermédiaire de ces deux-là en un seul filtrage. C'est une possibilité supplémentaire offerte par l'utilisation de cette méthode.



(a) *original* (b) $A \leq 70$ (c) $A \leq 206$

(d) $A \leq 557$ (e) $A \leq 3267$ (f) $200 \leq A \leq 210$

FIGURE 6 – (a) image originale, (b-e) filtrage successif par suppression des nœuds dont la valeur est inférieure à la légende, (f) extraction d'une forme intermédiaire

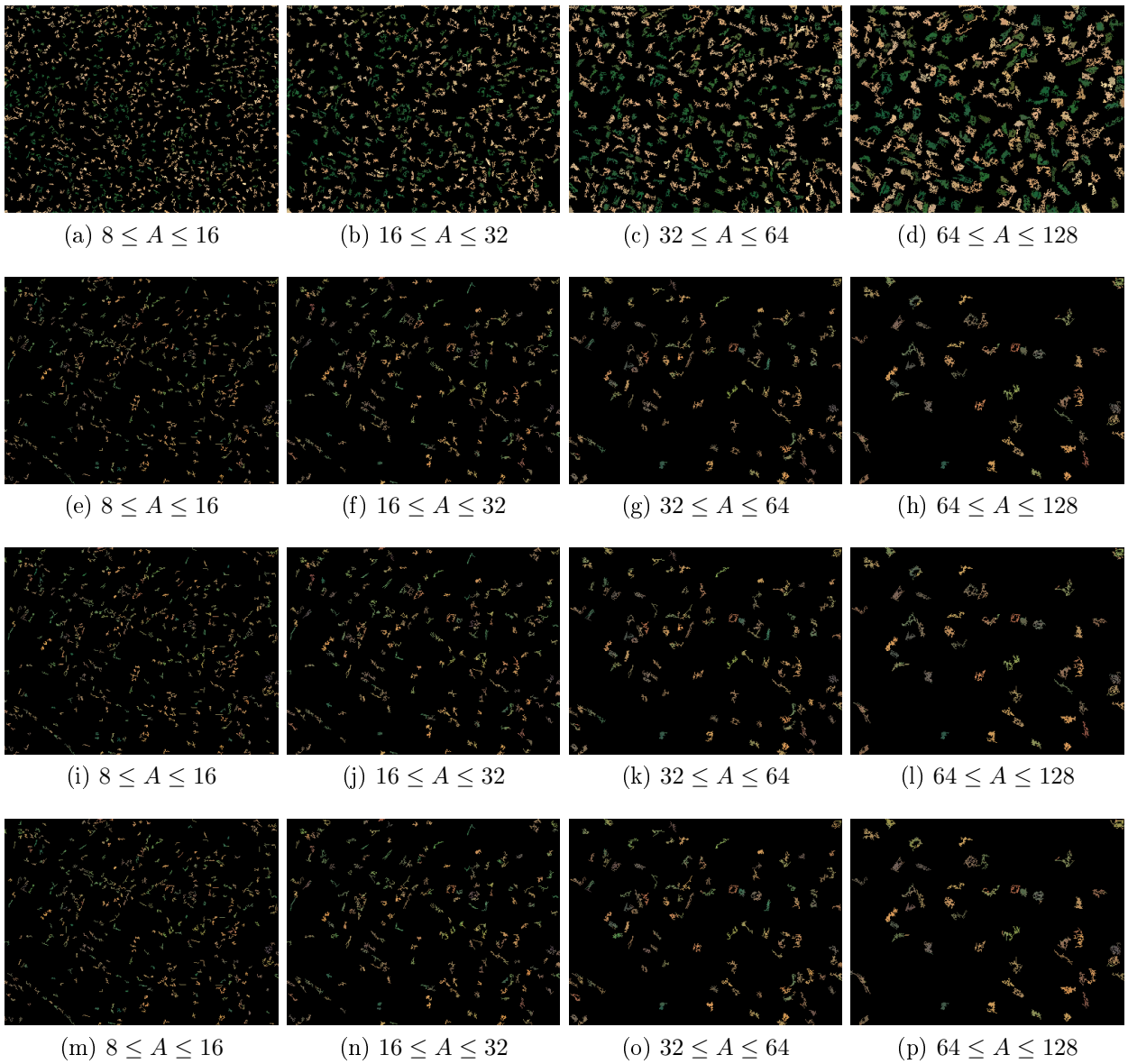
6.4 Influence des paramètres

Le principal argument influant sur la création de l'arbre des coupes multivalué est le paramétrage de la fenêtre de Parzen. L'application de cette fenêtre permet de *lisser* l'histogramme en recalculant un estimation de probabilité de densité. La taille de la fenêtre de Parzen ici appliquée sur l'espace RVB est en réalité un cube de dimensions $n*n*n$ où n est la *largeur* de la fenêtre.

La première ligne (figure 7) contient des coupes sans application de fenêtre de Parzen (histogramme brut). La seconde ligne contient des coupes avec application d'une fenêtre de Parzen de taille $3*3*3$. Il faut noter que ces images sont obtenues avec un facteur de subdivision de 8 donc la taille de la fenêtre est de $24*24*24$. La troisième ligne contient des coupes avec application d'une fenêtre de Parzen de taille $5*5*5$ donc de taille réelle $40*40*40$. La quatrième ligne contient des coupes avec application d'une fenêtre de Parzen de taille $7*7*7$ donc de taille réelle $56*56*56$.

On peut observer que plus la fenêtre est large, plus l'histogramme est lissé, ce qui a pour conséquence d'affiner la coupe obtenue. C'est particulièrement visible sur la colonne la plus à droite où la fenêtre de grande taille (p), filtrée avec le même critère, affine considérablement la précision des zones extraites par rapport à la même coupe sans fenêtre (d).

En revanche, on peut observer que la taille de la fenêtre ne semble pas produire de résultats fondamentalement différents si l'on compare les trois dernières lignes. Cela pourrait s'expliquer par le fait que le le facteur de subdivision provoque un agrandissement automatique de la fenêtre de Parzen. Il faudrait alors regarder les résultats obtenus avec un facteur de subdivision de 1. Malheureusement, et ce pour des raisons purement liées au programme, l'obtention de ces résultats n'est pas possible.



(a) $8 \leq A \leq 16$ (b) $16 \leq A \leq 32$ (c) $32 \leq A \leq 64$ (d) $64 \leq A \leq 128$

(e) $8 \leq A \leq 16$ (f) $16 \leq A \leq 32$ (g) $32 \leq A \leq 64$ (h) $64 \leq A \leq 128$

(i) $8 \leq A \leq 16$ (j) $16 \leq A \leq 32$ (k) $32 \leq A \leq 64$ (l) $64 \leq A \leq 128$

(m) $8 \leq A \leq 16$ (n) $16 \leq A \leq 32$ (o) $32 \leq A \leq 64$ (p) $64 \leq A \leq 128$

FIGURE 7 – (a-d) coupes successives sans fenêtre de Parzen, (e-h) coupes successives avec une fenêtre de Parzen de $3*3$, (i-l) coupes successives avec une fenêtre de Parzen de $5*5$, (m-p) coupes successives avec une fenêtre de Parzen de $7*7$

7 Conclusion

Pour faire un tour d'horizon du travail accompli lors de ce stage de Master II, récapitulons ici les principales étapes et leurs dénouements.

L'on a proposé méthode de construction d'un ordre partiel basée sur les infirmations intrinsèques de l'image à savoir son histogramme. Partant d'un code existant permettant la génération de cet ordre ainsi qu'une reconstruction naïve de l'image résultat, deux nouveaux espaces de valeurs à savoir l'espace RVB et l'espace HSV ont été ajoutés avec succès avec tous les développements et adaptations nécessaires.

Diverses modifications améliorant le processus ont été ajoutées avec succès notamment l'ajout de classes d'équivalence, la non création des labels inutilisés... Trois nouvelles méthodes de reconstruction de l'image résultat ont également été intégrées au code : une reconstruction par représentant de classes d'équivalence, une reconstruction par moyenne des pixels originaux et une reconstruction par médiane des pixels originaux. Ces méthodes présentent les résultats sous diverses formes.

Afin de faciliter l'utilisation de cette méthode à des fins scientifiques, une interface regroupant sous une seule fenêtre les trois précédents programmes, clarifiant et automatisant les options disponibles a été développée avec succès.

Enfin un état de l'art des structures de représentations hiérarchiques des images avec ordre totaux et partiels a été effectuée afin de pouvoir comparer cette méthode aux autres (arbre et graphe des coupes, arbre de partitionnement binaire et arbre de partitionnement binaire multi-critères, arbre et graphe des formes...).

Des améliorations et travaux subséquents incluent la possibilité d'utilisation de critères de filtrage multi-attributs sous forme d'un arbre de critères, la définition formelle de la méthode sous forme d'équation ou bien l'introduction d'une base de données d'apprentissage visant à apprendre les critères photométriques destinant à venir se coupler à l'utilisation seule de l'histogramme de l'optique d'améliorer la qualité et la fiabilité de l'ordre partiel construit...

Sur une note plus personnelle, je tiens à remercier Benoît Naegel responsable du stage et Fabrice Heitz responsable de l'équipe IMAGeS dans laquelle ce stage a été effectué. J'aimerais également remercier Adrien Krähenbühl pour son aide et ses conseils apportés tout au long du stage en sa qualité de co-responsable.

À cet égard, j'en profite pour remercier l'ensemble du personnel du laboratoire ICube et de l'UFR de Mathématique-Informatique pour m'avoir permis de vivre cette aventure riche qui, je l'espère aura contribuer à me faire grandir.

8 Bibliographie

[1] P.Salembier, J.Serra, **Flat Zones Filtering, Connected Operators and Filters by Reconstruction**, *IEEE Transactions on Image Processing* 4(8) : 1153-1160, 1995

[2] P.Salembier, M.H.F. Wilkinson, **Connected Operators : A review of region-based morphological image processing techniques**, *IEEE Signal Processing Magazine* 6 : 136-157, 2009

[3] P.Salembier, A.Oliveras, L.Garrido, **Antiextensive Connected Operators for Image and Sequence Processing**, *IEEE Transactions on Image Processing* 7(4) : 555-570, 1998

[4] P.Salembier, P.Brigger, J.M.Casas, M.Cardàs, **Morphological Operators for Image and Video Compression**, *IEEE Transactions on Image Processing* 5(6) : 881-898, 1996

[5] R.Jones, **Connected Filtering and Segmentation using component trees**, *Computer Vision and Image Understanding* 75(3) : 215-228, 1999

[6] E.J.Breen, R.Jones, H.Talbot, **Mathematical Morphology : a useful set of tools for image analysis**, *Statistics and Computing* 10(2) : 105-120, 2000

[7] B.Naegel, N.Passat, **Component-trees and Multivalued Images : A comparative Study**, *Mathematical Morphology and Its Application to Signal and Image Processing* pp.261-271, 2009

[8] L.Najman, M.Couprie, **Building the Component-Tree in Quasi-Linear Time**, *IEEE Transactions on Image Processing* 15(11) : 3531-3539, 2006

[9] N.Passat, B.Naegel, **An extension of component-trees to partial orders**, *Proceedings/ ICIP...International Conference on Image Processing*, 2009

[10] C.Kurtz, B.Naegel, N.Passat, **Multivalued component-tree filtering**, *IEEE Transactions on Image Processing* 23(12) : 5152-5164, 2014

[11] E.Carlinet, T.Géraud, **A fair comparison of many max-tree algorithm**, *Mathematical Morphology and Its Applications to Signal and Image Processing* pp. 73-85, 2013

[12] P.Salembier, L.Garrido, **Binary Partition Tree as an Efficient Representation for Image Processing, Segmentation and Information Retrieval**, *IEEE Transactions on Image Processing* 9(4) : 561-576, 2000

[13] J.F.Randrianasoa, C.Kurtz, E.Desjardin, N.Passat, **Binary Partition Tree Construction from Multiple Features for Image Segmentation**, *Pattern Recognition* 84 : 237-250, 2018

[14] T.Géraud, E.Carlinet, S.Crozet, L.Najman, **A quasi-linear algorithm to compute the tree of shapes of a nD image**, *Mathematical Morphology and Its Applications to Signal and Image Processing* pp. 98-110, 2013

[15] Y.Pan, J.Douglas Birdwell, S.M.Djouadi, **Preferential Image Segmentation using Trees**

of Shapes, *IEEE Transactions on Image Processing* 18(4) : 854-866, 2009

[16] E.Carlinet, T.Géraud, **A comparative review of component tree computation algorithms**, textitIEEE Transactions on Image Processing, 23(9) : 3885-3895, 2014

[17] E.Carlinet, T.Géraud, **A Morphological Tree of Shapes for color images**, *2014 22nd International Conference on Pattern Recognition*, 2014

[18] E.Parzen, **On estimation of a probability function and mode**, *The Annals of Mathematical Statistics*, Vol. 33, No. 3, pp 1065-1076, 1962

[19] K.Benkrid, D.Crookes, **Design and FGPA implementation of Perimeter Estimator**, 2000

[20] R.Perrin, **Graphes de coupes sur images de labels**, *rapport de Travail d'Étude et de Recherche*, dépôt gitlab, Université de Strasbourg, 2018

[21] B.Naegel, **Projet 2015-CGraph-Shaping (branche TER_2018)**, *cgraph-shaping*, dépôt gitlab du laboratoire ICube

[22] B.Naegel, **Bibliothèque de traitement d'images LibTim**, *dépôt de projet Github*, Github

[22] B.Naegel, **multivaluedcomponenttree (branche stage)**, *dépôt de projet gitlab*, Laboratoire ICube