New Algorithms for Multivalued Component Trees

Nicolas Passat[1], Romain Perrin[2], Jimmy Francky Randrianasoa[2,3], Camille Kurtz[4], Benoît Naegel[2]

[1] Université de Reims Champagne Ardenne, CRESTIC, Reims, France
[2] Université de Strasbourg, CNRS, ICube, Strasbourg, France
[3] EPITA Research Laboratory (LRE), Le Kremlin-Bicêtre,France
[4] Université Paris Cité, LIPADE, Paris, France

Motivation

The component tree (CT) [3] can model grey-level images for various image processing / analysis purposes (filtering, segmentation, registration, retrieval…). Its generalized version, the multivalued component tree (MCT) [1] can model images with hierarchically organized values. We provide new tools:

- a new algorithm for the construction of MCTs;
- two strategies for building hierarchical orders on values, required to further build MCTs.

Multivalued Component Tree

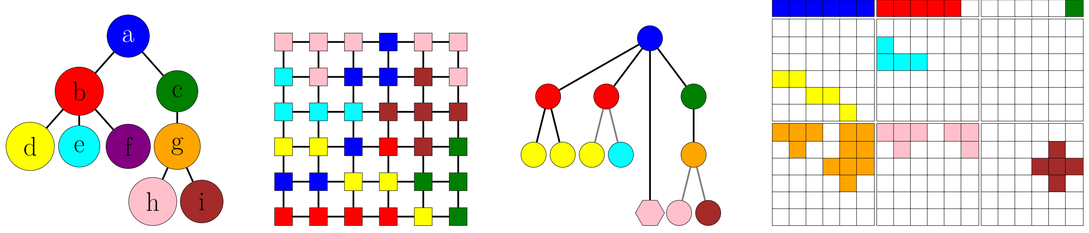
Let $\mathfrak{G} = (\Omega, \frown)$ be a non-directed graph. Let $\mathcal{C}[\mathrm{X}]$ be the set of the connected components of $\mathrm{X} \subseteq \Omega$.
Let $\mathbb{V}$ be a finite set and $\leqslant$ a hierarchical order on $\mathbb{V}$, i.e. an order (1) which admits a minimum (resp. a maximum) and (2) such that for any $\mathrm{v} \in \mathbb{V}$, the subset of the elements lower (resp. greater) than v is totally ordered by $\leqslant$.
Let us consider an image $\mathcal{F} : \Omega \to \mathbb{V}$. The threshold set of $\mathcal{F}$ at value $\mathrm{v} \in \mathbb{V}$ is defined by $\Lambda_{\mathrm{v}}(\mathcal{F}) = \{\mathbf{x} \in \Omega \mid \mathrm{v} \leqslant \mathcal{F}(\mathbf{x})\}$. We define the set of nodes of the MCT as

$$\Theta = \bigcup_{\mathrm{v} \in \mathbb{V}} \mathcal{C}[\Lambda_{\mathrm{v}}(\mathcal{F})] \qquad (1)$$

with $\mathbb{I}(\mathrm{X}) = \{\mathrm{v} \in \mathbb{V} \mid \mathrm{X} \in \mathcal{C}[\Lambda_{\mathrm{v}}(\mathcal{F})]\}$, $\omega(\mathrm{X}) = \bigvee^{\leqslant} \mathbb{I}(\mathrm{X})$ and $\tau(\mathrm{X}) = |\mathbb{I}(\mathrm{X})|$ for any node $\mathrm{X} \in \Theta$.
The inclusion relation $\subseteq$ is a hierarchical order on Θ. Let $\lhd$ be the reflexive-transitive reduction of $\subseteq$.
The Hasse diagram $\mathfrak{T} = (\Theta, \lhd)$ of $(\Theta, \subseteq)$ is the multivalued component tree of the image $\mathcal{F}$.
For any node $\mathrm{X} \in \Theta$, we set $\rho(\mathrm{X}) = \mathrm{X} \setminus \bigcup_{\mathrm{Y} \lhd \mathrm{X}} \mathrm{Y} = \{\mathbf{x} \in \Omega \mid \mathcal{F}(\mathbf{x}) = \omega(\mathrm{X})\}$.

The MCT $\mathfrak{T}$ is an image model of the image $\mathcal{F}$:

$$\forall \mathbf{x} \in \Omega, \mathcal{F}(\mathbf{x}) = \bigvee_{\mathrm{X} \in \Theta}^{\leqslant} \mathbf{1}_{(\mathrm{X}, \omega(\mathrm{X}))}(\mathbf{x}) \qquad (2)$$



(a) $(\mathbb{V}, \leqslant)$ (b) $\mathcal{F} : \Omega \to \mathbb{V}$ (c) $\mathfrak{T} = (\Theta, \lhd)$ (d) $\Lambda_{\mathrm{v}}(\mathcal{F})$ for $\mathrm{v} \in \mathbb{V}$

Building the Multivalued Component Tree

This construction algorithm derives from the CT construction of [3].

- `nodes`: stores the nodes of the multivalued component tree.
- `points`: stores the processed points of the image.
- `status`: stores the status of each point of the image.
- `nb_nodes` and `index`: store the number of nodes already fully built and the index of the node currently built at each value of $\mathbb{V}$.
- `progress`: indicates if there exists a node at value v, currently under construction or to be built, which is an ancestor of the node at value u currently being defined.

Algorithm: Build the multivalued component tree

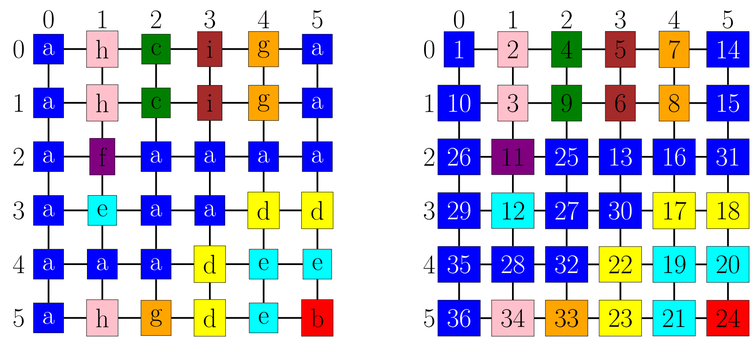
Input: $(\Omega, \frown), (\mathbb{V}, \leqslant), \mathcal{F} : \Omega \to \mathbb{V}$
Output: $\mathfrak{T} = (\Theta, \lhd)$
Build `nodes`, `points`, `status`, `nb_nodes`, `index`, `progress`
$\mathrm{v}_{\min} := \bigwedge^{\leqslant} \mathbb{V}$
Choose $\mathbf{x}_{\min} \in \Omega$ such that $\mathcal{F}(\mathbf{x}_{\min}) = \mathrm{v}_{\min}$
`points[`$\mathrm{v}_{\min}$`].add(`$\mathbf{x}_{\min}$`)`
`progress[`$\mathrm{v}_{\min}$`]` $:=$ true
Flood($\mathrm{v}_{\min}$)

Function: Flood(u $\in \mathbb{V}$)

Input: u $\in \mathbb{V}$: current level
Output: w $\in \mathbb{V}$: value of the parent node of the root of the built (partial) MCT at value u
while `!(points[u].empty())` do
  $\mathbf{x}$:= `points[u].remove()`
  if `index[u]` $>$ `nb_nodes[u]` then
    `nb_nodes[u]` := `index[u]`
    X := `create_node()` // new node in Θ
    `nodes[u].insert`(X)
  if $\mathcal{F}(\mathbf{x}) \neq$ u then
    w := $\mathcal{F}(\mathbf{x})$
    `points[w].add(`$\mathbf{x}$`)`
    `progress[w]` := true
    while u $<$ w do w := Flood(w)
  else
    `status[`$\mathbf{x}$`]` := `index[u]`
    `nodes[u][index[u]].add_to_proper_part(`$\mathbf{x}$`)`
    foreach $\mathbf{y} \frown \mathbf{x}$ do
      w := $\mathcal{F}(\mathbf{y})$
      if `status[`$\mathbf{y}$`]` $= -1$ then
        if u $\leqslant$ w then $\widehat{\mathrm{w}}$:= w
        else $\widehat{\mathrm{w}} := \bigwedge^{\leqslant}\{\mathrm{u}, \mathrm{w}\}$
        `points[`$\widehat{\mathrm{w}}$`].add(`$\mathbf{y}$`)`
        `status[`$\mathbf{y}$`]` := 0
        `progress[`$\widehat{\mathrm{w}}$`]` := true
        while u $<$ $\widehat{\mathrm{w}}$ do $\widehat{\mathrm{w}} :=$ Flood($\widehat{\mathrm{w}}$)
if u $=$ $\mathrm{v}_{\min}$ then
  w := ε
else
  w := $\bigvee^{\leqslant}\{\mathrm{w}' \in \mathbb{V} \mid \mathrm{w}' < \mathrm{u}\}$
  while `progress[w]` $=$ false do w := $\bigvee^{\leqslant}\{\mathrm{w}' \in \mathbb{V} \mid \mathrm{w}' < \mathrm{w}\}$
  `create_edge(nodes[u][index[u]],nodes[w][index[w]])` // new edge in $\lhd$
`progress[u]` = false
`index[u]++`
return w

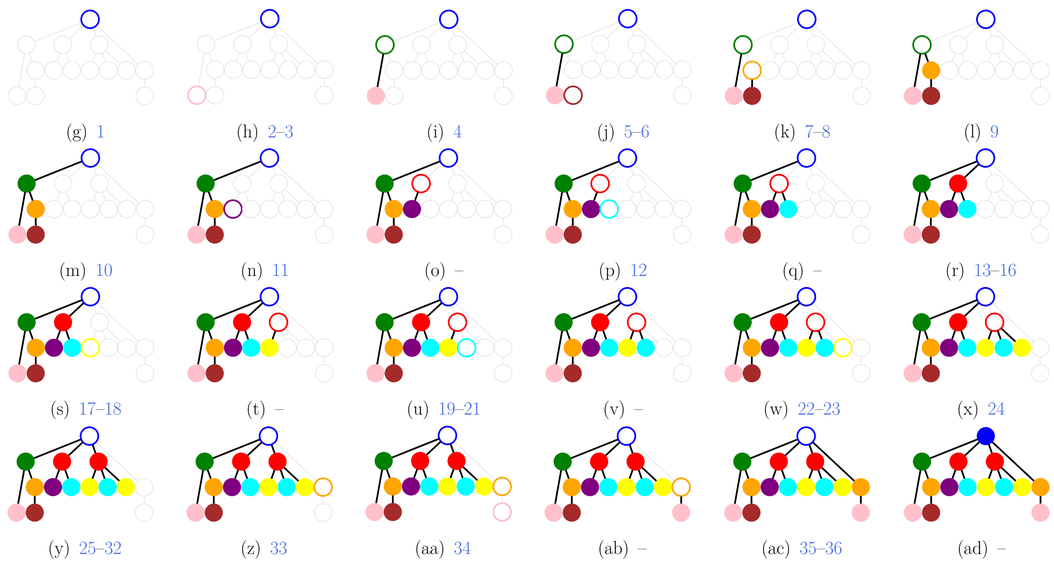
Running the Construction Algorithm

Construction of the MCT of an image $\mathcal{F} : \Omega \to \mathbb{V}$. At a current stage: a plain coloured node is fully built; a contour-colored node is under construction; a non-colored node has not been considered yet; a black edge is built; a light gray edge is not built.



(e) $\mathcal{F} : \Omega \to \mathbb{V}$

(f) Order of processing of Ω



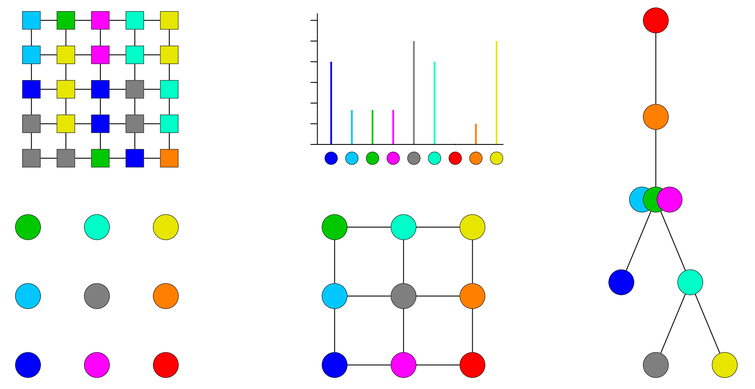
(g) 1 (h) 2–3 (i) 4 (j) 5–6 (k) 7–8 (l) 9
(m) 10 (n) 11 (o) – (p) 12 (q) – (r) 13–16
(s) 17–18 (t) – (u) 19–21 (v) – (w) 22–23 (x) 24
(y) 25–32 (z) 33 (aa) 34 (ab) – (ac) 35–36 (ad) –

Hierarchical order construction (1/2): (Pre)ordering the value set

Building the MCT requires a hierarchical order on $\mathbb{V}$.
First, we can build a hierarchical preorder $\leqslant_{\mathbb{V}}$ on $\mathbb{V}$. This can be done by building the CT of an "image" composed by the value set. It is only required that $\mathbb{V}$ be endowed with:

- an adjacency $\frown_{\mathbb{V}}$, allowing to map a graph structure on $\mathbb{V}$;
- a function $\delta_{\mathbb{V}} : \mathbb{V} \to \mathbb{N}$, allowing to associate to each element of $\mathbb{V}$ a value within the totally ordered set $(\mathbb{N}, \leq)$.

the CT of $\delta_{\mathbb{V}}$ is a hierarchical preorder $\leqslant_{\mathbb{V}}$ on $\mathbb{V}$.

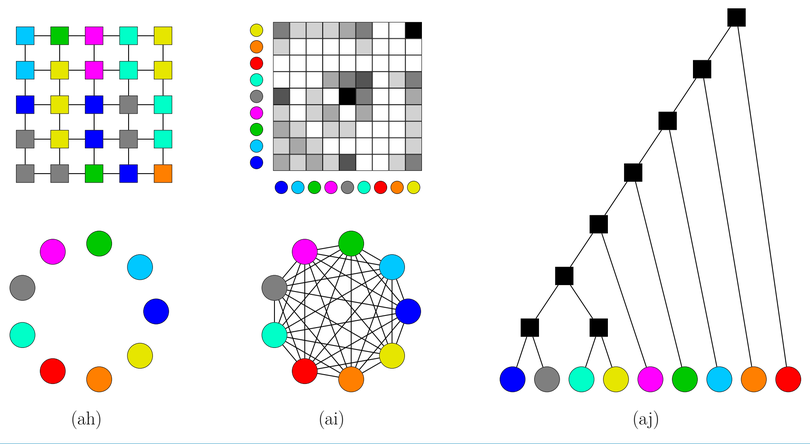


(ae) (af) (ag)

Hierarchical order construction (2/2): Ordering the enriched value set

Second, we can build hierarchical order $\leqslant_{\mathbb{W}}$ on $\mathbb{W} \supset \mathbb{V}$ so that the elements of $\mathbb{V}$ are the maximal elements with respect to $\leqslant_{\mathbb{W}}$.
This can be done by building the binary partition tree (BPT) [2] of an "image" composed by the value set $\leqslant_{\mathbb{V}}$. It is only required that $\mathbb{V}$ be endowed with:

- an adjacency $\frown_{\mathbb{V}}$, allowing to map a graph structure on $\mathbb{V}$;
- a priority function $\delta_{\frown_{\mathbb{V}}} : \frown_{\mathbb{V}} \to \mathbb{N}$, allowing to determine the couples of nodes to be merged in priority.



(ah) (ai) (aj)

References

[1] C. Kurtz, B. Naegel, and N. Passat. connected filtering based on multivalued component-trees. IEEE Transactions on Image Processing, 23:5152–5164, 2014.

[2] P. Salembier and L. Garrido. binary partition tree as an efficient representation for image processing, segmentation, and information retrieval. IEEE Transactions on Image Processing, 9:561–576, 2000.

[3] P. Salembier, A. Oliveras, and L. Garrido. anti-extensive connected operators for image and sequence processing. IEEE Transactions on Image Processing, 7:555–570, 1998.

Funding

This work was supported by the French Agence Nationale de la Recherche (grants ANR-20-THIA-0006, ANR-20-CE45-0011, ANR-22-CE45-0034 and ANR-23-CE45-0015).