Efficient component-hypertree construction based on hierarchy of partitions

Alexandre Morimitsu, Nicolas Passat, Wonder Alves, Ronaldo F. Hashimoto

Efficient component-hypertree construction based on hierarchy of partitions

Alexandre Morimitsu[a,*], Nicolas Passat[b], Wonder A. L. Alves[c], Ronaldo F. Hashimoto[a,*]

[a] *Universidade de São Paulo, Instituto de Matemática e Estatística, Rua do Matão, 1010, São Paulo, Brazil*
[b] *Université de Reims Champagne Ardenne, CReSTIC EA 3804, 51097 Reims, France*
[c] *Informatics and Knowledge Management Graduate Program, Universidade Nove de Julho, São Paulo, Brazil*

Abstract

The component-hypertree is a data structure that generalizes the concept of component-tree to multiple (increasing) neighborhoods. However, construction of a component-hypertree is costly because it needs to process a high number of neighbors. In this article, we present some properties used to obtain optimized neighborhoods for component-hypertree computation. Using these properties,we explore a new strategy to obtain neighboring elements based on hierarchy of partitions, leading to a more efficient algorithm with the drawback of a slight loss of precision on the distance of merged nodes.

Keywords:

1. Introduction

In the field of mathematical morphology, there exist different ways of representing digital images. In particular, instead of relying only on the grid of pixels of an image and the values associated with these pixels, alternative representations often aim at organizing the information from spatial and structural points of view, via region-based and connectivity-based paradigms.

Usually, such representations rely on graphs and hierarchies of partitions. A well-known example is the binary partition tree, introduced by Salembier and Garrido (2000). In this data structure, an initial partition of the image is given and neighboring regions are then merged together based on a predefined criterion, progressively defining coarser partitions. This structure can be modeled as a tree, where leaves give the initial partition whereas the intermediate nodes show how the elements of these partitions are merged.

By contrast to the criterion-based binary partition tree, another family of criterion-free hierarchical structures has been proposed. The principal representative of this family is the component-tree, introduced by Salembier et al. (1998) and Jones (1999), and further generalized into variants, e.g. the self-dual tree of shapes (Monasse and Guichard, 2000) or the multivalued component-tree (Kurtz et al., 2014). The component-tree is a lossless image model, that represents a gray-level image as a hierarchy of connected components, based on the inclusion relation of these connected components on the level sets of the image. Component-trees gained widespread adoption, thanks to the efficiency of algorithms dedicated to build (Carlinet and Géraud, 2014) and process them (Jones, 1999; Guigues et al., 2006).

Although being criterion-free, the component-tree presents two meta-parameters: (1) the order on the values of the image, and (2) the connectivity defined for modeling the topological organization of the pixels. Recent researches aimed at exploring the consequences of relaxing the usual constraints on these both meta-parameters. Regarding the order on values, a generalization of component-trees to component-graphs was inves-

*Corresponding authors:
Email addresses: alexandre.morimitsu@usp.br (Alexandre Morimitsu), ronaldo@ime.usp.br (Ronaldo F. Hashimoto)

tigated by turning the total order into partial ones (Passat and Naegel, 2014); this also led to new results on tree of shapes for multivalued images, proposed by Carlinet and Géraud (2015).

Regarding connectivity, the usual way of spatially organizing pixels of an image consists of relying on the standard adjacencies defined on $\mathbb{Z}^d$ in the context of digital topology. Alternative strategies were developed, e.g. by Ronse (1998); Serra (1998); Braga-Neto and Goutsias (2002); Ronse (2014), leading to many types of connectivities, expanding the notion of how regions connect in digital spaces.

In the field of component-trees, two ways were considered for taking into account these new paradigms of connectivity: first, by considering oriented connectivities (Perret et al., 2015); second, by considering families of increasing connectivities, leading to the notion of component-hypertrees (Passat and Naegel, 2011), which are directed acyclic graphs (DAGs).

A component-hypertree is a forest composed of several component-trees built from the same gray-level image, but for many increasing connectivities. In addition to the inclusion links between the nodes, within each component-tree, a second series of inclusion links is defined, between the nodes of the distinct component-trees, leading to the organization of these nodes into a forest of partition trees, namely one partition tree for each level set of values. The set of all the nodes, endowed with these two families of links, leads to a DAG.

In the field of hierarchical models, recent efforts were geared towards the design of strategies that gather information provided by several trees. Two kinds of strategies are developed. The first consists of fusing the information provided by various kinds of trees, leading to a final data structure that remains a tree (Perret et al., 2018; Randrianasoa et al., 2018). The second consists of building more complex data structures that explicitly preserve the information obtained from several trees, leading to final data structures that are no longer trees. This is, for instance, the case for braids of partitions (Kiran and Serra, 2015; Tochon et al., 2019) but also for component-hypertrees.

Perhaps due to the higher complexity of their structure, component-hypertrees are not as widely adopted compared to other tree structures. In recent years, some studies regarding efficient storage of component-hypertrees have been carried out by Morimitsu et al. (2019b). It was shown that some properties of component-hypertrees allow one to take advantage of standard component-tree construction algorithms and to adapt them for performing efficient allocation of nodes and arcs of a component-hypertree (by avoiding redundancies). In addition, attribute computation in such optimized structures was investigated by Morimitsu et al. (2019a).

In terms of component-hypertree construction, the original work proposed by Passat and Naegel (2011) relied on an algorithm for component-tree construction using mask-based connectivity (Ouzounis and Wilkinson, 2007). More recently Morimitsu et al. (2015) proposed a family of dilation-based connectivities that present specific properties well-fitted for accelerating the computation of component-hypertrees compared to the more general mask-based approach.

In this article, we intend to further explore these concepts, by showing how properties of some known algorithms used for component-tree computation can be used to obtain optimized choices of neighborhoods for component-hypertree construction. Based on these properties, we also present a novel strategy for obtaining neighborhoods based on hierarchies of partitions, that allow for the development of even faster algorithms, at the cost of a loss of precision in terms of distance of merged components.

In Section 2, we review the background notions required for understanding the proposed approach. In Sections 3 and 4, we describe component-trees and component-hypertrees. Our main contributions are in Section 5, where, based on properties of an algorithm presented in Section 4, we propose a strategy for obtaining optimized neighborhoods that speed up the hypertree construction algorithm. Then, we explain how the dilation-based strategy presented by Morimitsu et al. (2015) are related to this approach and present the novel strategy for efficiently obtaining neighboring pixels based on a pyramidal hierarchy of partitions. In Section 6, we discuss the advantages and drawbacks of each strategy, based on complexity analysis and experiments. Conclusion and perspectives are presented in Section 7.

2. Background Notions

2.1. Images

Let $\mathcal{D}_f \subset \mathbb{Z}^d$ ($d > 0$) be a nonempty set. Let $\mathbb{K} = \{0, \ldots, K-1\} \subset \mathbb{N}$, with $K > 0$. A gray-level image f is a function $f : \mathcal{D}_f \to \mathbb{K}$. Any element $p \in \mathcal{D}_f$ is called a pixel. For any pixel $p \in \mathcal{D}_f$, the value $f(p)$ denotes the gray-level of p.

If $K = 2$, then f is a binary image. Any binary image can be defined by the set $X = \{p \in \mathcal{D}_f \mid f(p) = 1\}$. Given a value $\lambda \in \mathbb{K}$, the upper level set of f at level λ is defined as $X_\lambda(f) = \{p \in \mathcal{D}_f \mid f(p) \geq \lambda\}$.

A neighborhood relation (or simply, neighborhood) is defined as a set $\mathcal{A} \subseteq \mathcal{D}_f \times \mathcal{D}_f$. For any pair of pixels $(p, q) \in \mathcal{A}$, we say that the pixel q is a $\mathcal{A}$-neighbor (or simply, a neighbor) of the pixel p. A neighborhood $\mathcal{A}$ is a symmetric neighborhood if $(p, q) \in \mathcal{A} \Rightarrow (q, p) \in \mathcal{A}$ and vice-versa.

A structuring element (SE) $\mathcal{S} \subset \mathbb{Z}^d$ is a set of offsets. Given a set $P \subset \mathbb{Z}^d$ and an SE $\mathcal{S}$, the dilation of P by $\mathcal{S}$, denoted by $P \oplus \mathcal{S}$, is defined as the Minkowski addition $P \oplus \mathcal{S} = \{p + s \mid p \in P, s \in \mathcal{S}\}$. The reflection of an SE $\mathcal{S}$, denoted by $\check{\mathcal{S}}$, is defined as $\check{\mathcal{S}} = \{-s \mid s \in \mathcal{S}\}$. If $\mathcal{S} = \check{\mathcal{S}}$, then we say that $\mathcal{S}$ is a symmetric SE. SEs can be used for designing neighborhoods; given an SE $\mathcal{S}$, we define the set of neighboring pixels $\mathcal{A}(\mathcal{S})$ as:

$$\mathcal{A}(\mathcal{S}) = \{(p, q) \mid p \in \mathcal{D}_f, q \in (\{p\} \oplus \mathcal{S}) \cap \mathcal{D}_f\} \quad (1)$$

2.2. Graphs

A graph G is a pair $G = (V, E)$, where V defines the set of vertices of G and $E \subseteq V \times V$ is a binary relation that defines the set of arcs of G. We consider directed graphs; in other words, $(v_1, v_2) \in E$ and $(v_2, v_1) \in E$ are considered to be different arcs. If $(v_1, v_2) \in E \Leftrightarrow (v_2, v_1) \in E$, then E is a symmetric relation.

A graph G is a weighted graph if it is composed of a triple $G = (V, E, \omega)$, where $\omega : V \to \mathbb{R}$ is a function that assigns a numerical weight to each vertex.

Given a graph G, a subgraph G' of G is a graph $G' = (V', E')$ such that $V' \subseteq V$ and $E' \subseteq E$. Given G and a set $V' \subset V$, the vertex-induced subgraph $G' = (V', E')$ is the subgraph of G such that $E' \subseteq E$ is composed of all the arcs $(v_1, v_2) \in E$ satisfying $v_1, v_2 \in V'$.

A path in a graph $G = (V, E)$ is a sequence of vertices $(v_1, \ldots, v_{LP})$ such that every $v_j \in V$ ($1 \leq j \leq LP$) and every pair $(v_j, v_{j+1}) \in E$ ($1 \leq j < LP$). If a path starts and ends at the same vertex, this path is called a directed cycle. A graph without directed cycles is called a directed acyclic graph (DAG).

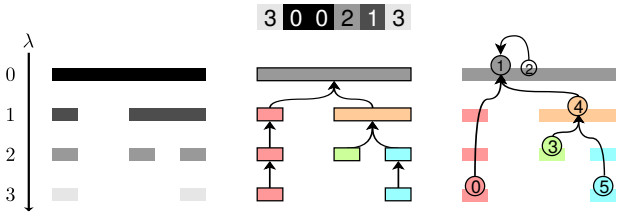

Figure 1: At the top: a 1-dimensional gray-level image f composed of six pixels, with a set of values $\mathbb{K} = \{0, 1, 2, 3\}$. Below, from left to right: the four level sets of f; the component-tree of f; and a graphical representation of parent, which stores the max-tree of f. Each element $p \in \mathcal{D}_f$ is depicted as a circle (where larger circles indicate canonical elements) and arcs represent the parenthood relation. The CCs (colored rectangles) can be reconstructed from the subtrees rooted in the canonical elements.

From the notion of path, connectedness in graphs can be defined. Let $G = (V, E)$ be a graph. A vertex $v \in V$ is said to be connected to $v' \in V$ iff there exists a path from v to v' in G. If E is a symmetric relation, then v connected to v' implies that v' is also connected to v. Given a symmetric relation E, a connected component (CC) in a graph is defined as a maximal set of connected vertices and we say a graph is connected if it has only one connected component. (In this work, it suffices to define CCs only when E is symmetric.)

2.3. Images as Graphs

Any gray-level image f can be represented by a weighted graph $(\mathcal{D}_f, \mathcal{A}, f)$. In the sequel, we assume that a gray-level image is defined as a connected weighted graph with a symmetric neighborhood.

Let $X \subseteq \mathcal{D}_f$ be (the foreground of) a binary image and $\mathcal{A} \subseteq \mathcal{D}_f \times \mathcal{D}_f$ a symmetric neighborhood. The set of CCs of X induced by $\mathcal{A}$ (further called the $\mathcal{A}$-CCs of X), denoted by $CC(X, \mathcal{A})$, is the set of the CCs of the graph $G = (X, \mathcal{A})$. For a gray-level image f, we define its set of $\mathcal{A}$-CCs as:

$$CC(f, \mathcal{A}) = \bigcup_{\lambda=0}^{K-1} CC(X_\lambda(f), \mathcal{A}) \quad (2)$$

namely as the union of the $\mathcal{A}$-CCs for all the binary images obtained from the successive thresholds of f.

2.4. Trees

If a graph $G = (V, E)$ is such that $|V| = |E| + 1$ and if there is a vertex $v_r \in V$ such that there exists a (unique) path from v to v_r, for every $v \in V$, then G is called a (rooted) tree. Vertices of trees are also called nodes. In particular, the node v_r is called the root of the tree.

If $(v_1, v_2) \in E$, then v_1 is a child of v_2 and v_2 is the parent of v_1. Given a node $v \in V$, we denote its parent as $par(v)$. If v is the root, then $par(v)$ if undefined and, if v has no children, then v is called a leaf. If two nodes $v_1, v_2 \in V$ are such that v_1 is connected to v_2, then v_1 is a descendant of v_2 and v_2 in an ancestor of v_1. Given a node $v \in V$, its set of descendants is denoted by $desc(v)$. The depth of a node v is the number of arcs existing in the path linking v to the root v_r (where the depth of v_r is 0).

Given a tree $G = (V, E)$ and a node $v \in V$, the subtree (of G) rooted in v consists of the subgraph G' induced by the set $V' = \{v\} \cup desc(v)$. A graph defined as the disjoint union of one or more trees is called a forest.

2.5. Hypertrees

Let $G = (V, E)$ be a DAG. We assume that E can be partitioned into $\eta > 0$ nonempty subsets E_k $(1 \leq k \leq \eta)$ (i.e. with $E_a \cap E_b = \emptyset$ for $1 \leq a < b \leq \eta$ and $\bigcup_{k=1}^{\eta} E_k = E$), and that the graphs $G_k = (V, E_k)$ are forests.

In particular, for each node $v \in V$, there exists at most one node $v_k \in V$ such that $(v, v_k) \in E_k$. In this way, v_k is the parent of v with respect to E_k and we denote $par_k(v) = v_k$.

We say that G is a hypertree[1] if, for any $v \in V$ such that $par_k(v)$ exists for all $1 \leq k \leq \eta$, we have, for all $a \neq b$:

$$par_a(par_b(v)) = par_b(par_a(v)) \tag{3}$$

Note that, by setting $\eta = 1$, any tree is indeed a hypertree.

[1]The defined notion of hypertree may be different from the notion commonly used in graph theory (namely, a hypergraph that admits a host graph which is a tree). Here, the terminology of hypertree denotes the definition of many sets of edges E_k, where the subgraph induced by each E_k is a forest. In other words, a hypertree could be seen as a specific collection of graphs.

3. Component-Trees and Component-Hypertrees

3.1. Component-Trees

Let f be a gray-level image, seen as a weighted graph $G(f) = (V, E, \omega) = (\mathcal{D}_f, \mathcal{A}, f)$. The component-tree of f is the tree $CT = (V_{CT}, E_{CT})$ where $V_{CT} = \{(C, \lambda) \mid C \in CC(X_\lambda(f), \mathcal{A}), \lambda \in \mathbb{K}\}$ (see Eq. (2)), whereas E_{CT} is defined by the following relation: $((C_1, \lambda_1), (C_2, \lambda_2)) \in E_{CT}$ iff $C_1 \subseteq C_2$ and $\lambda_1 = \lambda_2 + 1$.

Component-trees can be efficiently stored using max-trees, which are based on the union-find structure introduced by Tarjan (1975). A max-tree can be stored in an array, usually called parent, which consists of a mapping $\mathcal{D}_f \to \mathcal{D}_f$ that can be viewed as a tree $T_{\mathsf{parent}} = (\mathcal{D}_f, E)$, where $(p, q) \in E \Leftrightarrow q = \mathsf{parent}[p]$. Since there is a direct correspondence between these two notions, from now on, terminology of trees are applied directly to the parent arrays.

Given an array parent, a pixel $p \in \mathcal{D}_f$ is canonical if p is the root node p_r or $f(p) > f(\mathsf{parent}[p])$. The representative of $p \in \mathcal{D}_f$ is the canonical element returned by the function $rep : \mathcal{D}_f \to \mathcal{D}_f$, defined as follows: $rep(p) = p$, if p is canonical; or $rep(p) = rep(\mathsf{parent}[p])$, otherwise. Two pixels $p, q \in \mathcal{D}_f$ are comparable in parent if and only if one of these conditions is true: (1) $rep(p) = rep(q)$; or (2) $rep(p) \in desc(rep(q))$; or (3) $rep(q) \in desc(rep(p))$.

Let parent be an array satisfying the following properties: (1) for any $p \in \mathcal{D}_f$ with $p \neq p_r$, $f(p) \geq f(\mathsf{parent}[p])$; and (2) for any pair $(p, q) \in \mathcal{A}$, p and q are comparable. Then, we say that parent stores the max-tree of f (using $\mathcal{A}$) and any $\mathcal{A}$-CC C of f is represented by a canonical element r, where $C = \{r\} \cup desc(r)$. Conversely, if parent stores the max-tree of f (using $\mathcal{A}$), then properties (1) and (2) above are true. See Fig. 1.

3.2. Component-Hypertrees

Let us assume that we no longer consider only one neighborhood $\mathcal{A}$, but a set of many increasing neighborhoods $\{\mathcal{A}_1, \ldots, \mathcal{A}_n\}$ $(n \geq 1)$. We can build, for each neighborhood $\mathcal{A}_i$ $(1 \leq i \leq n)$, a specific component-tree $CT_i = (V_{CT_i}, E_{CT_i})$. The graph $G = (\bigcup_{i=1}^{n} V_{CT_i}, \bigcup_{i=1}^{n} E_{CT_i})$ obtained by the union of these n component-trees is, by construction, a forest.

We denote $V_{HT} = \bigcup_{i=1}^{n} V_{CT_i}$ and $E_{HT}^{\uparrow} = \bigcup_{i=1}^{n} E_{CT_i}$. For the sake of readability, a node of G is denoted as a triplet

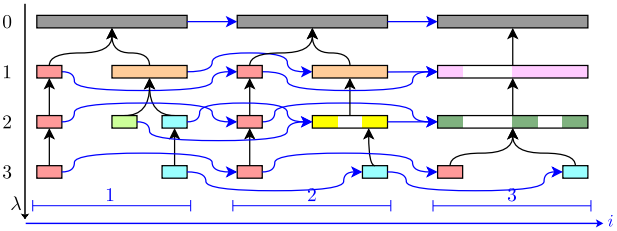

Figure 2: An example of a complete component-hypertree of the gray-level 1-dimensional image from Fig. 1. Each node $N = (C, \lambda, i)$ is represented by a rectangle, with C drawn inside each node. The vertical disposition of the nodes indicates their gray-level λ whereas neighborhood indices i give their horizontal disposition. Nodes with the same color correspond to the same CC. Arcs from $E^{\uparrow}_{HT}$ are in black, whereas arcs are from $E^{\rightarrow}_{HT}$ are in blue.

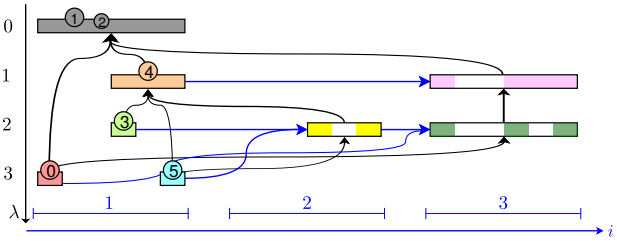

Figure 3: The component-hypertree obtained using an optimized version of Alg. 1, which is a compact version of the complete component-hypertree of Fig. 2. Numbered circles inside nodes indicate the stored pixels.

(C, λ, i), which means that it corresponds to a binary connected component C of the image f obtained at the level set λ with a neighborhood $\mathcal{A}_i$; in other words, we have:

$$V_{HT} = \{(C, \lambda, i) \mid C \in CC(X_\lambda(f), \mathcal{A}_i)), 0 \leq \lambda < K, 1 \leq i \leq n\} \quad (4)$$

Let us now assume that the neighborhoods $\mathcal{A}_i$ are increasing, i.e., $i \leq j \Rightarrow \mathcal{A}_i \subseteq \mathcal{A}_j$. and let (C, λ, i) be a node of V_{HT}, where $\lambda \in \mathbb{K}$. It is plain that there exists a unique node $(C', \lambda, i+1)$ of V_{HT} such that $C \subseteq C'$.

Based on this fact, one can build a second family of edges, namely $E^{\rightarrow}_{HT}$ defined as:

$$E^{\rightarrow}_{HT} = \{((C, \lambda, i), (C', \lambda, i{+}1)) \mid C \subseteq C', 0 \leq \lambda < K, 1 \leq i < n\} \quad (5)$$

which enriches the first family of edges:

$$E^{\uparrow}_{HT} = \{((C, \lambda, i), (C', \lambda{-}1, i)) \mid C \subseteq C', 0 < \lambda < K, 1 \leq i \leq n\} \quad (6)$$

In particular, the graph $(V_{HT}, E^{\rightarrow}_{HT})$ is a forest. In this forest, each node (C, λ, n) $(0 \leq \lambda < K)$ is the root of a specific tree, and the nodes of this tree are all the nodes corresponding to connected components included in C. More precisely, at each gray-level λ, these nodes form a partition of C, and these partitions are increasingly refined from n down to 1.

We say that the component-hypertree is complete to explicitly mention that it models all the nodes of V_{HT}, via a multiset of connected components (which will no longer be the case for optimized, compact variants of component-hypertrees); see Fig. 2. It is plain that a complete component-hypertree is a hypertree, with respect to the characterization given in Eq. (3).

4. Algorithmics for Component-Hypertrees

Following the definitions and notations stated above, an algorithm for (minimal) component-hypertree construction from f using $\mathbb{A}$ is given in Alg. 1. It iterates on the neighborhood index i and allocates new nodes and arcs at the end of each step $1 \leq i \leq n$.

The CONNECT procedure called in Alg. 1 is presented in Alg. 2. It consists of a variation of the algorithm presented by Ouzounis and Wilkinson (2007) for a parallel implementation of the max-tree. It receives two $\mathcal{A}_i$-neighboring pixels p, q and an array parent. Then, the array is updated by modifying parenthood relations to make p and q comparable (no changes are performed if p and q were already comparable) without invalidating previously existing comparability relationships. More specifically, the updated parent satisfies the properties presented in Prop. 1:

Proposition 1. *Let p, q be two non-comparable pixels in parent. If parent′ is the result of the* CONNECT *procedure when applied to the pair (p, q), then, the following properties are valid:*

1. *For any $p' \in \mathcal{D}_f$ that is not the root node (of parent′), $f(p') \geq f(\text{parent}'[p'])$.*
2. *Comparability relationships are preserved, i.e., given two pixels $p', q' \in \mathcal{D}_f$ such that p' and q' are comparable in parent, then p' and q' are also comparable in parent′.*
3. *Suppose $p' = p$ or p' is an ancestor of p in parent and suppose $q' = q$ or q' is an ancestor of q in parent. Then, p' and q' are comparable in parent′.*

Algorithm 1 Component-hypertree construction.

```
1: procedure buildHypertree(f, 𝔸 = (𝒜_1, ..., 𝒜_n))
2:     for p ∈ 𝒟_f do parent[p] ← ∅;  ▷ i.e., not defined
3:     for 1 ≤ i ≤ n do
4:         for (p, q) ∈ 𝒜_i do
5:             rP ← rep(p), rQ ← rep(q);
6:             if f(rP) ≥ f(rQ) then
7:                 connect(parent, f, rP, rQ);
8:             else connect(parent, f, rQ, rP);
9:         Allocate nodes and arcs of the hypertree.
```

Algorithm 2 connect procedure

```
1: procedure connect(parent, f, p_1, p_2)
2:     if p_1 ≠ p_2 then
3:         parP_1 ← rep(parent[p_1]);        ▷ rep(∅) = ∅;
4:         if parP_1 ≠ p_2 then
5:             if f(parP_1) ≥ f(p_2) then    ▷ f(∅) = −∞;
6:                 connect(parent, f, parP_1, p_2);
7:             else
8:                 parent[p_1] ← p_2;
9:                 connect(parent, f, p_2, parP_1);
```

Let parent_i denote the array parent at the end of step i ($1 \leq i \leq n$) in Alg. 1. Thanks to the properties of Prop. 1, to compute parent_i, it suffices to call the connect procedure for all pairs $(p, q) \in \mathcal{A}_i$. When Line 9 is reached at step i, parent_i stores the max-tree of f (using $\mathcal{A}_i$) and the nodes and arcs of the hypertree can be allocated. In summary, for each canonical element p, a new node $N = (p, f(p), i)$ is allocated. Then, for every node $N_1 = (q, f(q), i-1)$ with neighborhood index $i-1$, we allocate an arc from N_1 to $N_2 = (rep(q), f(q), i)$ and, for every canonical element $p \in \mathcal{D}_f$, we allocate the arc $((p, f(p), i), (\mathsf{parent}[p], f(\mathsf{parent}[p]), i))$.

This step can be optimized if redundant nodes and arcs are not allocated. This can be done by keeping track of changes in the array at each step $1 \leq i \leq n$: if p is canonical in parent_i and $desc(p)$ is the same in parent_{i-1} and parent_i, then p represents a $\mathcal{A}_{i-1}$-CC and need not be allocated at step i, since a node representing it was already allocated at step $i-1$. Analogously, any arc $(p, \mathsf{parent}_i[p])$ where both $rep(p)$ and $rep(\mathsf{parent}_i[p])$ represent $\mathcal{A}_{i-1}$-CCs need not be allocated at step i. Efficient ways of implementing these optimizations can be found in (Morimitsu et al., 2019b). An example of an optimized component-hypertree is given in Fig. 3.

5. Choice of Sets of Neighboring Pixels

The optimality of Alg. 1 depends on the choice of the set of neighboring pixels. In particular, its complexity is linked to the number of calls of Line 4. In other words, Alg. 1 can be optimized if the number of calls of connect is reduced.

For this purpose, given a gray-level image f and a increasing sequence $\mathbb{A} = (\mathcal{A}_1, \ldots, \mathcal{A}_n)$, we say that two neighborhoods $\mathcal{A}_i$ and a $\mathcal{A}'_i$ are equivalent iff, given parent_{i-1}, running Alg. 1 at step i with $\mathcal{A}_i$ or $\mathcal{A}'_i$ both produce an array parent_i that stores the same max-tree. Given a sequence $\mathbb{A}' = (\mathcal{A}'_1, \ldots, \mathcal{A}'_n)$, $\mathbb{A}$ and $\mathbb{A}'$ are equivalent iff, for any $1 \leq i \leq n$, $\mathcal{A}_i$ and $\mathcal{A}'_i$ are equivalent, and this implies that running Alg. 1 with $\mathbb{A}'$ and $\mathbb{A}$ produces the same hypertrees. For some well-chosen types of neighborhoods, it is possible to obtain optimized equivalent sequences $\mathbb{A}'$ that significantly reduce the complexity of Alg. 1 (compared with $\mathbb{A}$).

A possible strategy for obtaining those neighborhoods is to build a neighborhood $\mathcal{A}'_i \subset \mathcal{A}_i$ without including arcs $(p, q) \in \mathcal{A}_i$ when p and q are comparable in $parent_{i-1}$ (since they do not change the array when connect is called), or when it is known that there exists another pair $(p', q') \in \mathcal{A}_i$ that will make p and q become comparable as a side effect of calling connect for (p', q').

More formally, suppose parent_{i-1} is given and there exist pixels p, p', q and $q' \in \mathcal{D}_f$ such that, in parent_{i-1}, the following conditions are valid: (1) p and q are not comparable; (2) $p' \in desc(rep(p))$; and (3) $q' \in desc(rep(q))$ (note that this implies that p' and q' are also not comparable in parent_{i-1}). Additionally, suppose (p, q) and $(p', q') \in \mathcal{A}_i$. Thanks to Prop. 1, calling connect to (p', q') at step i of Alg. 1 also makes p and q comparable in parent_i. Hence, if connect is called for (p', q') (i.e., the pair with greater depths in parent_{i-1}, which is the pair with higher gray-levels in a max-tree), then calling connect for (p, q) is not needed, implying that $\mathcal{A}_i$ and $\mathcal{A}'_i = \mathcal{A}_i \setminus \{(p, q)\}$ are equivalent. Besides, all pairs in $P \times Q \subseteq \mathcal{A}_i$, where P (resp., Q) consists of pixels in the

path from p' to the root node (resp., q' to the root) are not needed, with the exception of pair (p', q'). Formally, this idea is presented in Prop. 2.

Proposition 2. *Suppose* (1) $\mathcal{A}_i$ *is a neighborhood;* (2) parent_{i-1} *is given, with* $i > 1$*;* (3) $P \times Q \subseteq \mathcal{A}_i$ *such that* P *(resp.,* Q*) is a subset of a path in* parent_{i-1}*. If* $(h_p, h_q) \in P \times Q$ *such that* h_p *(resp.* h_q*) is the element of* P *(resp.* Q*) with the greatest depth in* parent_{i-1}*, then* $\mathcal{A}_i$ *and* $\mathcal{A}'_i = (\mathcal{A}_i \setminus (P \times Q)) \cup \{(h_p, h_q)\}$ *are equivalent.*

Note that, for any pair of sets (P, Q) satisfying the conditions of Prop. 2, $\mathcal{A}'_i$ has up to $|P \times Q| - 1$ fewer elements than $\mathcal{A}_i$.

Although Prop. 2 seems to suggest a reduction of $\mathcal{A}_i$ to $\mathcal{A}'_i$, some choices of neighborhoods provide efficient ways of computing $\mathcal{A}'_i$ directly. In the context of Prop. 2, for any pair (P, Q) satisfying the specified conditions, only the pair (h_p, h_q) needs to be added to $\mathcal{A}'$. In the following, we present some sequence of neighborhoods that can benefit from this strategy.

5.1. Dilation-Generated Neighborhoods

In this section, we show how sequences of dilation-generated neighborhoods (Morimitsu et al., 2015) are particular cases of neighborhoods that use the properties of Prop. 2 to speed up hypertree construction. In essence, these sequences can merge pixels at increasing distances, using incremental dilations by small SEs to grow the neighborhoods.

To explain it in more detail, a dilation-generated neighborhood sequence $\mathbb{A} = (\mathcal{A}_1, \ldots, \mathcal{A}_n)$ is built from a sequence of increasing SEs $\mathbb{S} = (\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_n)$, where $\mathcal{A}_i = \mathcal{A}(\mathcal{S}_i)$ (see Eq. (1)). The sequence $\mathbb{S}$, in turn, is built from another sequence of SEs $\mathbb{B} = (\mathcal{B}_1, \ldots, \mathcal{B}_n)$ in the following way: $\mathcal{S}_1 = \mathcal{B}_1 \oplus \check{\mathcal{B}}_1$ and, for any $1 < i \leq n$, $\mathcal{S}_i = \mathcal{S}_{i-1} \oplus \mathcal{B}_i \oplus \check{\mathcal{B}}_i$ or equivalently, $\mathcal{S}_i = \mathcal{W}_i \oplus \check{\mathcal{W}}_i$, where $\mathcal{W}_i = \mathcal{B}_1 \oplus \ldots \oplus \mathcal{B}_i$.

The main idea consists of building mappings, for $2 \leq i \leq n$, $L_{i-1} : \mathcal{D}_f \oplus \mathcal{W}_{i-1} \to \mathcal{P}(\mathcal{D}_f)$ (where $\mathcal{P}(\cdot)$ denotes power set) in such a way that, for each $r \in \mathcal{D}_f \oplus \mathcal{W}_{i-1}$, any two distinct pixels in $L_{i-1}(r)$ are comparable in parent_{i-1} (i.e., $L_{i-1}(r)$ is a subset of a path in parent_{i-1}). Supposing that a pair $(p, q) \in \mathcal{A}_i$ with $L_{i-1}(p) \times L_{i-1}(q) \subseteq \mathcal{A}_i$ is given, based on Prop. 2, we define:

$$\mathcal{A}'_i = (\mathcal{A}_i \setminus (L_{i-1}(p) \times L_{i-1}(q))) \cup \{h_p^{i-1}, h_q^{i-1}\}, \quad (7)$$

where $h_p^{i-1} \in L_{i-1}(p)$ and $h_q^{i-1} \in L_{i-1}(q)$ have the greatest depth in parent_{i-1}. Then $\mathcal{A}_i$ and $\mathcal{A}'_i$ are equivalent.

To build these mappings we note that, using dilation properties, any two pixels $p, q \in \mathcal{D}_f$ are $\mathcal{A}(\mathcal{S}_i)$-neighbors iff $(\{p\} \oplus \mathcal{W}_i) \cap (\{q\} \oplus \mathcal{W}_i) \neq \emptyset$. Thanks to this property, if we define $L_i(r) = \{x \in \mathcal{D}_f \mid x \in \{r\} \oplus \check{\mathcal{W}}_i\}$, for $r \in \mathcal{D}_f \oplus \mathcal{W}_i$, we have $(\{p\} \oplus \mathcal{W}_i) \cap (\{q\} \oplus \mathcal{W}_i) \neq \emptyset$ iff $\exists r \in \mathcal{D}_f \oplus \mathcal{W}_i : p, q \in L_i(r)$. Thus, any pair of distinct pixels in $L_i(r)$ are $\mathcal{A}(\mathcal{S}_i)$-neighbors and for any $(p, q) \in \mathcal{A}(\mathcal{S}_i)$, $\exists r \in \mathcal{D}_f \oplus \mathcal{W}_i$ with $p, q \in L_i(r)$. Additionally, given any $r \in \mathcal{D}_f \oplus \mathcal{W}_i$, $L_i(r) = \bigcup_{x \in \{r\} \oplus \check{\mathcal{B}}_i} L_{i-1}(x)$. This implies that $p, q \in L_i(r)$ iff $\exists p', q' \in \{r\} \oplus \check{\mathcal{B}}_i$ such that $p \in L_{i-1}(p')$ and $q \in L_{i-1}(q')$.

With the mappings L_{i-1} built, we now analyze this problem in the context of Alg. 1 and Prop. 2. Assuming Alg. 1 is running and step i has just started, our goal is to compute $\mathcal{A}'_i$ equivalent to $\mathcal{A}_i$ to speed up the i-th step, as follows: for all $r \in \mathcal{D}_f \oplus \mathcal{W}_i$, we compute all combinations of $p, q \in r \oplus \check{\mathcal{B}}_i$ with $p \neq q$, and add the pair (h_p^{i-1}, h_q^{i-1}) to $\mathcal{A}'_i$ (see Eq.(7)). In this way, the optimization of Alg. 1 is performed at Line 4 by replacing the neighborhood $\mathcal{A}_i$ with $\mathcal{A}'_i$.

In general, this strategy of building $\mathcal{A}'_i$ and call CONNECT to its pairs is faster than processing all $\mathcal{A}_i$-neighboring pixels. A more detailed complexity analysis is provided in Sec. 6. Furthermore, there are other optimizations and technical details that were omitted here, since they are more closely related to the choice of neighborhoods than the properties of Prop. 2. For more details, the reader is referred to (Morimitsu et al., 2015).

5.2. Neighborhoods Based on Hierarchies of Partitions

From now on, we present a novel strategy to obtain a sequence of neighboring pixels $\mathbb{A}'$ based on hierarchy of partitions. This strategy can significantly reduce the complexity of the hypertree building algorithm by employing Prop. 2.

Let $\mathbb{H} = (\mathcal{H}_1, \ldots, \mathcal{H}_n)$ be a hierarchy of partitions of $\mathcal{D}_f$, i.e., each $\mathcal{H}_i$ is a partition of $\mathcal{D}_f$ for $1 \leq i \leq n$ and, for every element R of $\mathcal{H}_i$, there exists an element R' of $\mathcal{H}_{i+1}$ such that $R \subseteq R'$, i.e., the partition $\mathcal{H}_i$ refines the partition $\mathcal{H}_{i+1}$ for $1 \leq i < n$. In addition, we consider,

for each partition $\mathcal{H}_i$, a region adjacency graph (RAG) $G_i = (\mathcal{H}_i, E_i)$, where we say $R, R' \in \mathcal{H}_i$ are adjacent iff $(R, R'), (R', R) \in E_i$ (where E_i is a symmetric relation). Thus, a RAG $G_i = (\mathcal{H}_i, E_i)$ induces a neighborhood $\mathcal{A}_i$ in such way that, for any two distinct pixels $p, p' \in \mathcal{D}_f$, we have $(p, p') \in \mathcal{A}_i$ iff there exist $R, R' \in \mathcal{H}_i$ such that $p \in R$, $p' \in R'$ and either $R = R'$ or $(R, R') \in E_i$.

Depending on the choices of $\mathbb{H}$ and $\mathbb{E} = (E_1, \ldots, E_n)$, it is possible to design efficient algorithms for hypertree construction. In particular, let $\mathbb{H}$ and $\mathbb{E}$ be defined in such a way that the following condition holds:

$$\forall R \in \mathcal{H}_i, \forall R_1, R_2 \in \mathcal{H}_{i-1} \text{ with } R_1 \neq R_2, R_1 \cup R_2 \subseteq R \Rightarrow (R_1, R_2) \in E_{i-1} \quad (8)$$

Suppose that $\mathbb{H}$ and $\mathbb{E}$ are given and they satisfy the conditions in Eq. (8). Since any region $R \in \mathcal{H}_i$ is a merge of regions of $\mathcal{H}_{i-1}$, then for any $(p, q) \in \mathcal{A}_i$ with $p, q \in R$, either p and q both belong to the same region of $\mathcal{H}_{i-1}$ or they belong to adjacent regions of G_{i-1}. In both cases, by definition, this implies that p and q are $\mathcal{A}_{i-1}$-neighbors.

Hence, for any pair $(p, q) \in (\mathcal{A}_i \setminus \mathcal{A}_{i-1})$, p and q are in adjacent regions of G_i. Thus, there exist two adjacent regions $R_p, R_q \in \mathcal{H}_i$ such that $p \in R_p$, $q \in R_q$. As explained above, any two distinct elements of R_p (resp. R_q) are $\mathcal{A}_{i-1}$-neighbors, which means, at the start of step i in Alg. 1, they are comparable in parent_{i-1}. Then, Prop 2 applies and we can define:

$$\mathcal{A}'_i = (\mathcal{A}_i \setminus (R_p \times R_q)) \cup \{h^i_p, h^i_q\}, \quad (9)$$

where $h^i_p \in R_p$ and $h^i_q \in R_q$ have the greatest depth in parent_{i-1}. From Prop 2, we conclude that $\mathcal{A}_i$ and $\mathcal{A}'_i$ are equivalent.

5.3. Pyramidal Hierarchy

One particular strategy that can be used to efficiently build a hierarchy of partitions is to design a pyramidal hierarchy. For that, consider a sequence $(\mathcal{D}_0, \mathcal{D}_1 \ldots, \mathcal{D}_n)$ such that $\mathcal{D}_f = \mathcal{D}_0 \supset \mathcal{D}_1 \supset \ldots, \supset \mathcal{D}_n$ and, for $i = 1, \ldots, n$, let $\rho_i : \mathcal{D}_{i-1} \to \mathcal{D}_i$ be a mapping, called downsampling, that assigns each pixel $p \in \mathcal{D}_{i-1}$ to a pixel $u \in \mathcal{D}_i$. In this way, we define a pyramidal hierarchy as a sequence of downsamplings $(\rho_1, \ldots, \rho_n)$. An interesting property is that the composition of i downsamplings $\theta_i = \rho_i \rho_{i-1} \cdots \rho_1 : \mathcal{D}_f \to \mathcal{D}_i$ induces an equivalence relation on $\mathcal{D}_f$, denoted by $\equiv_i$, as follows:

$$\forall p, q \in \mathcal{D}_f, (p \equiv_i q) \Leftrightarrow \theta_i(p) = \theta_i(q) \quad (10)$$

This equivalence relation leads us to a partition $\mathcal{H}_i$ of $\mathcal{D}_f$ in a such way that two distinct pixels $p, q \in \mathcal{D}_f$ are in the same region in $\mathcal{H}_i$ iff $\theta_i(p) = \theta_i(q) = u \in \mathcal{D}_i$. More formally, $\mathcal{H}_i = \{R^u_i : u \in \mathcal{D}_i\}$ where $R^u_i = \{x \in \mathcal{D}_f : u = \theta_i(x)\}$. In this way, a pyramidal hierarchy $(\rho_1, \ldots, \rho_n)$ defines a hierarchy of partitions $\mathbb{H} = (\mathcal{H}_1, \ldots, \mathcal{H}_n)$.

Given these definitions, we now focus on obtaining a specialized and suitable pyramidal hierarchy for efficient component-hypertree construction. One possible choice consists of the following: given $(t_1, \ldots, t_d) \in \mathbb{Z}^d$ with $t_j > 0$, we define the downsampling $\rho_i : \mathcal{D}_{i-1} \to \mathcal{D}_i$ as, for any $x = (x_1, \ldots, x_d) \in \mathcal{D}_{i-1}$,

$$\rho_i(x = (x_1, \ldots, x_d)) = \begin{cases} (x_1, \ldots, x_d), & i = 1; \\ (\lfloor x_1/t_1 \rfloor, \ldots, \lfloor x_d/t_d \rfloor), & i > 1, \end{cases} \quad (11)$$

where $\lfloor a \rfloor$ indicates the greatest integer less than or equal to a.

In this way, $\mathcal{H}_i$ consists of hyper-rectangles of $\mathcal{D}_f$ of size $((t_1)^{i-1}, (t_2)^{i-1}, \ldots, (t_d)^{i-1})$.

Now, we define the sequence $\mathbb{E} = (E_1, \ldots, E_n)$. In order to ensure the validity of condition given in (8), we make use of the following RAG $G_i = (\mathcal{H}_i, E_i)$, where

$$E_i = \{(R^v_i, R^u_i) \in \mathcal{H}_i \times \mathcal{H}_i : v, u \in \mathcal{D}_i, |v_j - u_j| < t_j, 1 \leq j \leq d\} \quad (12)$$

Finally, we analyze this problem in the context of Alg. 1 and Prop. 2. Suppose that Alg. 1 is running at the beginning of step i. Then, since the choice of $\mathbb{E}$ satisfies the conditions given in Eq. (8), we can make use of Eq. (9) to compute a neighborhood $\mathcal{A}'_i$ equivalent to $\mathcal{A}_i$, using the following strategy: for all $p \in \mathcal{D}_i$, find all $q \in \mathcal{D}_i$ such that $p \neq q$ and $(R^p_i, R^q_i) \in E_i$; and, then, add the pair (h^i_p, h^i_q) into $\mathcal{A}'_i$ (see Eq. (9)). An example is provided in Fig. 4. An analysis showing the efficiency of this strategy applied to the pyramidal hierarchy of hyper-rectangles is given in the next section.

6. Analysis

6.1. Complexity Analysis

Let $\mathbb{A} = (\mathcal{A}_1, \ldots, \mathcal{A}_n)$ be a sequence of increasing sets of neighboring pixels. When a sequence $\mathbb{A}'$ is built

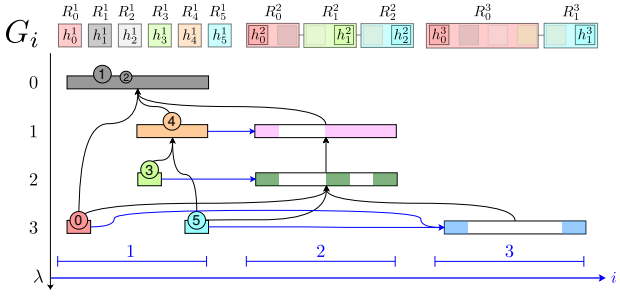

Figure 4: Top: the sequence of graphs of the hierarchy of partitions $\mathbb{G} = (G_1, G_2, G_3)$ from the image f from Fig. 1 with $t = (t_1) = (2)$. Bottom: the resulting component-hypertree of f using $\mathbb{A}' = (\mathcal{A}'_1, \mathcal{A}'_2, \mathcal{A}'_3)$.

(where $\mathbb{A}$ and $\mathbb{A}'$ are equivalent), then time to build the neighborhoods must be taken into consideration. In the case of Sec. 5.1, the construction strategy has complexity $\Theta(|\mathcal{D}_f \oplus \mathcal{W}_n| \cdot |\mathcal{B}_i|^2)$.

For the case of the pyramidal hierarchy of hyper-rectangles, $|\mathcal{D}_i|$ is divided by $\Pi_{j=1}^{d} t^j$ for every increment of i (for $i > 1$), implying that $|\mathcal{D}_i|$ decreases geometrically. When using Eq. (12) to define E_i, each region $R_i^p \in \mathcal{H}_i$ has at most $m = \Pi_{j=1}^{d}(2t_j - 1)$ adjacent regions. Hence, m is a fixed value that does not depend on i and construction of $\mathcal{A}'_i$ is $\Theta(|\mathcal{D}_i| \cdot m)$.

In Alg. 1, complexity is directly related to the choice of $\mathbb{A}$ and the complexity of the CONNECT procedure. In the worst case, CONNECT has a complexity of $O(K)$, meaning that Alg. 1 has a complexity of $O(\sum_{i=1}^{n} |\mathcal{A}_i| \cdot K)$. When $\mathbb{A}$ is swapped for $\mathbb{A}'$, than $|\mathcal{A}'_i|$ is bounded by the complexity of the building process since, for each combination tested (when building $\mathcal{A}'_i$), at most one pair is added to $\mathcal{A}'_i$. Hence, for dilation-generated neighborhoods, using the approach explained in Sec. 5.1, $\sum_{i=1}^{n} |\mathcal{A}'_i| = O(\sum_{i=1}^{n} |\mathcal{D}_f \oplus \mathcal{W}_i| \cdot |\mathcal{B}_i|^2)$. In general, this is efficient because $|\mathcal{B}_i|$ is usually very small. For example, to build SEs $\mathcal{S}_i$ where $\mathcal{S}_i$ is the d-dimensional cube with sides $2i+1$, then $|\mathcal{B}_i| = 2^d$. In most practical cases, $d = 2$ or 3, so this is a small constant. This would imply that $\sum_{i=1}^{n} |\mathcal{A}'_i| = O(|\mathcal{D}_f \oplus \mathcal{W}_n| \cdot n)$.

For this particular choice of $\mathcal{S}_i$, this is better than running Alg. 1 directly in $\mathcal{A}(\mathcal{S}_i)$, since $|\mathcal{S}_i| = (2i+1)^d$, and $\sum_{i=1}^{n} |\mathcal{A}(\mathcal{S}_i)| = \Theta(|\mathcal{D}_f| \cdot n^{d+1})$. However, the complexity of the dilation-based implementation can be further improved using other optimizations described in Morimitsu et al. (2015). In the most optimized version, $\sum_{i=1}^{n} |\mathcal{A}'_i|$ is expected to be $\Theta(|\mathcal{D}_f| \cdot 2^d \cdot d \cdot \log(n))$, or $\Theta(|\mathcal{D}_f| \cdot \log(n))$ if we assume d is a constant. This complexity is obtained using a probabilistic approach and tests carried out on the ICDAR database (Nayef et al., 2017) corroborate this expected complexity.

Finally, for the pyramidal hierarchy, we have $|\mathcal{A}'_i| = O(|\mathcal{D}_i| \cdot \Pi_{j=1}^{d}(2t_j - 1))$. In the particular case where $t = (2, \ldots, 2)$, we have $\sum_{i=1}^{n} |\mathcal{A}'_i| = O(|\mathcal{D}_f| \cdot (1 + \frac{1}{2^d - 1}) \cdot 3^d) = O(|\mathcal{D}_f| \cdot 3^d)$, i.e., if d is a constant, then $\sum_{i=1}^{n} |\mathcal{A}'_i| = O(|\mathcal{D}_f|)$.

6.2. Critical Analysis

In the most optimized version of the dilation-based strategy, the size of the neighborhoods can grow at most linearly for each dimension. This implies that a large number of SEs is required to connect distant nodes. By contrast, the pyramidal approach reduces the domain in each dimension geometrically, which means that a much lower n can be used.

In order to experimentally corroborate these observations, a set of 500 random images ranging from 1 to 10 megapixels (MP) was chosen from the ICDAR 2017 Robust Reading Challenge Dataset (Nayef et al., 2017). For each image, component-hypertrees using both strategies were computed. For pyramidal hierarchies, $t = (2, 2)$ was used and, for dilation-based neighborhoods, $\mathcal{B}_i$ was the 2×2 SE for all $1 \leq i \leq n$.

The obtained results are presented in Fig. 5. From the graph, it is possible to see that the pyramidal strategy is about 2.5 times faster than the dilation-based approach, for the fixed $n = 11$, and more than 14 times faster when $n = 512$ was applied to try to extract nodes with the same distance of the pyramidal approach.

A drawback of geometric growth in the size of the neighborhoods is that many neighborhoods are skipped, which induces a loss of precision in terms of distance of merged nodes.

Even so, it is still possible to obtain an estimate of the (Chebyshev) distance between merged nodes. Experiments performed in 15 000 nodes extracted from images of the same ICDAR database showed that, for $t = (2, 2)$, the value $2^{(i-1)}$ is a good estimate of the distance of nodes merged at step i ($2 \leq i \leq n$). Compared to the exact values, the estimated distances differed by about 8% on average, with more of half of the nodes differing by less

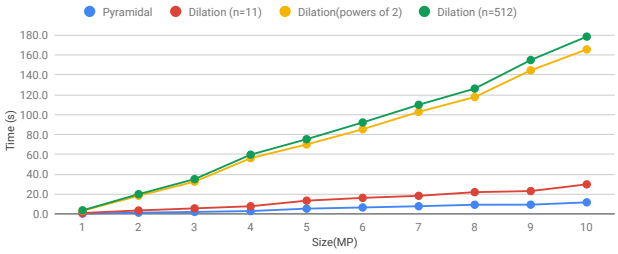

Figure 5: Comparison of average times between the pyramidal hierarchy strategy (blue line, $n = 11$) and the dilation-based strategy using: $n = 11$ (red); $n = 512$ allocating updating the hypertree at powers of 2 (yellow), to try to match the graph allocated using the pyramidal approach; and $n = 512$ allocating all nodes (green). The implementation was written in Java and was tested using a i7 2.6GHz processor with 16GB of RAM.

than 5%, more than 90% of the nodes differing by less than 25% and about 0.1% of the nodes with a difference of more than 50%.

Even though the computed distance is not exact, this degree of precision can still be useful in some applications. For example, in text extraction applications, one common assumption is that space between letters of a single word is somewhat constant and smaller than spaces between words. Then, we can extract different scales of objects from textual images (letters, words, and lines of texts) by selecting nodes in which the variance of the horizontal spacing of the merged parts are within a certain threshold. To computing statistics of attributes in component-hypertree, the approach in (Morimitsu et al., 2019b)) was used. An example is given in Fig. 6.

7. Conclusion

In this article, we reviewed the theory and algorithms used for component-hypertree computation, highlighting how some properties of an algorithm used for max-trees can be used to optimize component-hypertree construction. Specifically, this led to a proposition that was used as a foundation of a novel strategy for efficient component-hypertree construction based on a pyramidal hierarchy of partitions. Complexity analysis and experimental results were provided to demonstrate that the proposed method computes component-hypertree more efficiently than existing approaches without a significant loss of precision regarding the distance of merged nodes.

In future works, we intend to investigate other hierarchies that could be used for efficient component-hypertree construction. Also, we aim to analyze the impact of using non-integer values for rescaling the domain in the pyramidal approach. Although this approach loses some properties (since it does not generate a hierarchy of partitions), it can also benefit from the fact that it keeps the same computation complexity and skips fewer neighborhoods, reducing the loss of precision in terms of distance computation.

Acknowledgements

This study was funded in part by the CAPES – Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (Finance Code 001); FAPESP – Fundação de Amparo a Pesquisa do Estado de São Paulo (Proc. 2015/01587-0 and 2018/15652-7); CNPq – Conselho Nacional de Desenvolvimento Científico e Tecnológico (Proc. 428720/2018-8).

References

Braga-Neto, U., Goutsias, J., 2002. Connectivity on complete lattices: New results. Computer Vision and Image Understanding 85, 22–53.

Carlinet, E., Géraud, T., 2014. A comparative review of component tree computation algorithms. IEEE Trans. Image Processing 23, 3885–3895.

Carlinet, E., Géraud, T., 2015. MToS: A tree of shapes for multivariate images. IEEE Trans. Image Processing 24, 5330–5342.

Guigues, L., Cocquerez, J.P., Men, H.L., 2006. Scale-sets image analysis. Int. J. Computer Vision 68, 289–317.

Jones, R., 1999. Connected filtering and segmentation using component trees. Computer Vision and Image Understanding 75, 215–228.

Kiran, B.R., Serra, J., 2015. Braids of partitions, in: ISMM, pp. 217–228.

Kurtz, C., Naegel, B., Passat, N., 2014. Connected filtering based on multivalued component-trees. IEEE Trans. Image Processing 23, 5152–5164.

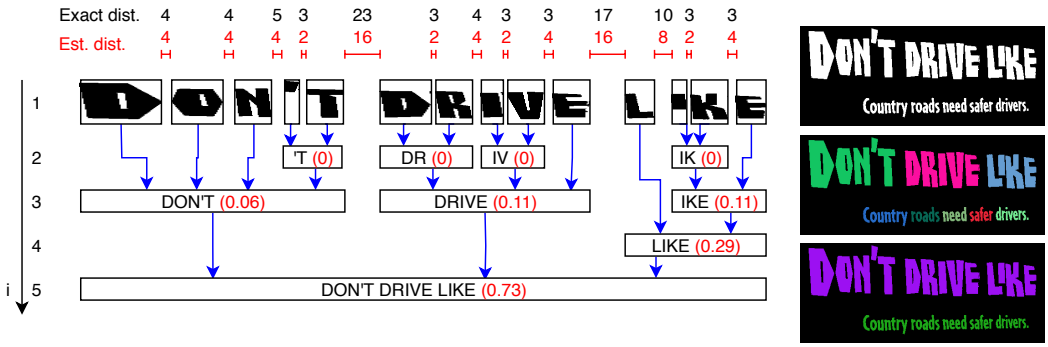

Figure 6: Computation of horizontal spacing and text extraction using the pyramidal approach, where the input is a binary image (image on the top right). The original image is from the ICDAR database (Nayef et al., 2017). In the top row, the exact horizontal distance between CCs, in pixels, is shown. In the row below, in red, the estimated values are displayed. For this example, $t = (2, 1)$ is used to merge nodes aligned horizontally, so the estimated distance is 2^{i-1}. At the bottom, some nodes of the hypertree are depicted, showing how letters are merged as i increases. Each node is indicated by its content (in black) and its variance of the horizontal spacing between merged nodes (in red). The displayed value is obtained by dividing the variance of the horizontal spacing by the average height of the merged nodes, to add scale invariance. By selecting the nodes with the highest i with variance below a given threshold, it is possible to extract different scales of objects. For example, by selecting nodes with variance less than 0.3, words can be extracted (middle right, where elements with the same color belong to the same node) and, if no threshold is given, then entire lines of text can be obtained (bottom right).

Monasse, P., Guichard, F., 2000. Scale-space from a level lines tree. J. Visual Communication and Image Representation 11, 224–236.

Morimitsu, A., Alves, W.A.L., Hashimoto, R.F., 2015. Incremental and efficient computation of families of component trees, in: ISMM, pp. 681–692.

Morimitsu, A., Alves, W.A.L., Silva, D.J., Gobber, C.F., Hashimoto, R.F., 2019a. Incremental attribute computation in component-hypertrees, in: ISMM, pp. 150–161.

Morimitsu, A., Alves, W.A.L., Silva, D.J., Gobber, C.F., Hashimoto, R.F., 2019b. Minimal component-hypertrees, in: DGCI, pp. 276–287.

Nayef, N., Yin, F., Bizid, I., Choi, H., Feng, Y., Karatzas, D., Luo, Z., Pal, U., Rigaud, C., Chazalon, J., Khlif, W., Luqman, M.M., Burie, J., Liu, C., Ogier, J., 2017. ICDAR 2017 RRC-MLT challenge, in: ICDAR, pp. 1454–1459.

Ouzounis, G.K., Wilkinson, M.H., 2007. A parallel implementation of the dual-input max-tree algorithm for attribute filtering, in: ISMM, pp. 449–460.

Ouzounis, G.K., Wilkinson, M.H.F., 2007. Mask-based second-generation connectivity and attribute filters. IEEE Trans. Pattern Analysis and Machine Intelligence 29, 990–1004.

Passat, N., Naegel, B., 2011. Component-hypertrees for image segmentation, in: ISMM, pp. 284–295.

Passat, N., Naegel, B., 2014. Component-trees and multivalued images: Structural properties. J. Mathematical Imaging and Vision 49, 37–50.

Perret, B., Cousty, J., Guimarães, S.J.F., Santana Maia, D., 2018. Evaluation of hierarchical watersheds. IEEE Trans. Image Processing 27, 1676–1688.

Perret, B., Cousty, J., Tankyevych, O., Talbot, H., Passat, N., 2015. Directed connected operators: Asymmetric hierarchies for image filtering and segmentation. IEEE Trans. Pattern Analysis and Machine Intelligence 37, 1162–1176.

Randrianasoa, J.F., Kurtz, C., Desjardin, E., Passat, N., 2018. Binary partition tree construction from multiple features for image segmentation. Pattern Recognition 84, 237–250.

Ronse, C., 1998. Set-theoretical algebraic approaches to connectivity in continuous or digital spaces. J. Mathematical Imaging and Vision 8, 41–58.

Ronse, C., 2014. Axiomatics for oriented connectivity. Pattern Recognition Letters 47, 120–128.

Salembier, P., Garrido, L., 2000. Binary partition tree as an efficient representation for image processing, segmentation, and information retrieval. IEEE Trans. Image Processing 9, 561–576.

Salembier, P., Oliveras, A., Garrido, L., 1998. Antiextensive connected operators for image and sequence processing. IEEE Trans. Image Processing 7, 555–570.

Serra, J., 1998. Connectivity on complete lattices. J. Mathematical Imaging and Vision 9, 231–251.

Tarjan, R.E., 1975. Efficiency of a good but not linear set union algorithm. J. ACM 22, 215–225.

Tochon, G., Dalla Mura, M., Veganzones, M.A., Chanussot, J., 2019. Braids of partitions for the hierarchical representation and segmentation of multimodal images. Pattern Recognition 95, 162–172.