Incremental component tree contour computation

Da Silva, Dennis; Kosinka, Jiri; Hashimoto, Ronaldo F.; Roerdink, J.B.T.M.; Morimitsu, Alexandre; Alves, Wonder A.L.

Published in:
Pattern Recognition Letters

DOI:
10.1016/j.patrec.2024.11.019

Publication date:
2025

Pattern Recognition Letters 187 (2025) 115–121

Contents lists available at ScienceDirect

Pattern Recognition Letters

journal homepage: www.elsevier.com/locate/patrec

Incremental component tree contour computation

Dennis J. Silva [a,c,*], Jiří Kosinka [c], Ronaldo F. Hashimoto [a], Jos B.T.M. Roerdink [c], Alexandre Morimitsu [a], Wonder A.L. Alves [b]

[a] Instituto de Matemática e Estatística, Universidade de São Paulo, Brazil
[b] Informatics and Knowledge Management Graduate Program, Universidade Nove De Julho, Brazil
[c] Bernoulli Institute for Mathematics, Computer Science and Artificial Intelligence, University of Groningen, The Netherlands

ARTICLE INFO

Editor: Song Wang

MSC:
68U10
94A08

Keywords:
Component tree
Contour
Incremental attribute
Mathematical morphology

ABSTRACT

A component tree is a graph representation that encodes the connected components of the upper or lower level sets of a grayscale image. Consequently, the nodes of a component tree represent binary images of the encoded connected components. There exist various algorithms that efficiently extract information and attributes of nodes of a component tree by incrementally exploiting the subset relation encoding in the tree. However, to the best of our knowledge, there is no such incremental approach to extract the contours of the nodes. In this paper, we propose an efficient incremental method to compute the contours of the nodes of a component tree by counting the edges (sides) of contour pixels. In addition, we discuss our method's time complexity. We also experimentally show that our proposed method is faster than the standard approach based on node reconstruction.

1. Introduction

In mathematical morphology, component trees and trees of shapes are image representations that encode grayscale images as trees. These structures represent the connected components or their saturation from the upper and/or lower level sets of the grayscale image into tree nodes, establishing a parenthood relationship based on the set-inclusion of the nodes [1]. These representations are utilized in various image-processing tasks, such as text location [2], car plate location [3], eye vessel segmentation [4], connected operator design [1], and interactive image editing [5].

A common approach to utilizing component trees in image processing or analysis tasks involves computing attributes that describe their nodes, based on the specific application [6]. This approach allows for the selection and filtering of nodes by these attributes, such as selecting or removing tiny, elongated, or circular objects. However, computing attributes for thousands of nodes individually can be costly. Thanks to the set-inclusion parenthood relationship in the tree, we can alleviate these computation costs by incrementally computing the attributes of a node. In this case, we reuse the computed attributes of child nodes to determine the attributes of their parent node. Various attributes can be computed incrementally, including area, height, volume [7], and the number of bit-quads [8,9]. However, to the best of our knowledge, there exists no incremental contour computation algorithm for component trees.

The *contour* (also called (inter-)*boundary* or *border*) of a binary image is the set of foreground pixels that have at least one neighboring background pixel [10], or a *background neighbor* for short. Contour extraction and tracking are important and long-standing problems in the field of image processing, traditionally applied in various areas such as pattern recognition [10], lossy compression, and boundary simplification [11,12]. Contour extraction involves identifying and extraction the boundaries of objects within an image, whereas contour tracking focuses on following these boundaries through a sequence. Recently, new research has emerged with the aim of improving the performance of these contour tracking algorithms [13–15], in particular by taking advantage of advances in parallel computing architectures, such as GPUs and multiprocessor systems [13,14]. These recent research efforts generally take a binary image as input to perform contour extraction. To use these methods on a grayscale image, it would be necessary to perform a threshold decomposition, which would result in many recomputations and a waste of computational resources. In contrast, our work addresses the extraction of contours from grayscale images

* Corresponding author at: Instituto de Matemática e Estatística, Universidade de São Paulo, Brazil.
E-mail addresses: d.j.da.silva@rug.nl, dennis@ime.usp.br (D.J. Silva), j.kosinka@rug.nl (J. Kosinka), ronaldo@ime.usp.br (R.F. Hashimoto), j.b.t.m.roerdink@rug.nl (J.B.T.M. Roerdink), alexandre.morimitsu@alumni.usp.br (A. Morimitsu), wonder@uni9.pro.br (W.A.L. Alves).

https://doi.org/10.1016/j.patrec.2024.11.019
Received 24 April 2024; Received in revised form 5 November 2024; Accepted 14 November 2024
Available online 23 November 2024

using a threshold decomposition encoded in a morphological tree. To avoid wasteful computations, we use component trees and perform the contour extraction incrementally. In this process, computations are reused along the decomposition encoded in the tree, making the method more efficient than traditional methods.

Few works have emerged with respect to contour extraction in morphological trees. Xu et al. [16,17] have used information computed over the contour of shapes, or nodes of a tree of shapes, of a grayscale image for segmentation tasks. They proposed a heuristic to minimize the Mumford-Shah energy functional generated by the nodes of the tree of shapes, employing a greedy algorithm that iteratively identifies and removes nodes to reduce the energy. The image partition generated by the final tree, with the selected nodes removed, produces a segmentation with low energy value. This method requires a regularization term that involves computing information extracted from the contour of the nodes. The authors present an incremental algorithm to compute this information. However, their method requires immersing the input image into the Khalimsky grid domain [18] by materializing the contour pixels , making it unsuitable for extracting the original contour of the nodes of the tree.

In this paper, we propose a new incremental algorithm to compute the contour of the nodes of a component tree (max-tree or min-tree) within the original input image domain. Our method keeps track of the number of background neighbors for each pixel in the image. When processing a node, any pixel with at least one background neighbor is included in the contour of the node, while pixels with no remaining background neighbors are removed from the contour. We incrementally reuse the contour computation of the child nodes by including or removing contour pixels as we scan the pixels of their parent node. Consequently, we obtain the contour pixels for each node of the tree. We experimentally demonstrate that our method is faster in extracting the contour of component tree nodes compared to a naive node reconstruction-based approach. Additionally, we discuss the asymptotic and worst-case scenario analysis of our method.

The rest of the paper is organized as follows. In Section 2, we present the definition of component trees and contours, and the naive node reconstruction-based approach to extract the contour. In Section 3, we present our proposed algorithm and discuss how it works and its asymptotic analysis. Section 4 describes the experiments we performed to evaluate our algorithm and the obtained results. We discuss applications that could be improved by using our proposed method in Section 5. Finally, the paper is concluded in Section 6.

2. Background

2.1. Image definition

A *grayscale image* is a function $f : D_f \to \mathbb{K}_f$, where its domain $D_f \subset \mathbb{Z}^2$ is a finite rectangular grid in $\mathbb{Z}^2$ and its codomain $\mathbb{K}_f = \{0, 1, \dots, K_f - 1\} \subset \mathbb{Z}$ is a set of integers. The elements of the image domain are referred to as *pixels*, and the elements of the image codomain are called *gray levels*. When $K_f = 2$, the image is *binary* and can be represented by the set $X = \{p \in D_f : f(p) = 1\}$. A pixel p in a binary image X is called a *foreground pixel* when $p \in X$ and a *background pixel* when $p \in (D_f \setminus X)$. We can threshold a grayscale image at $\lambda \in \mathbb{Z}$ to form binary images

$$X_\lambda(f) = \{p \in D_f : f(p) \geq \lambda\} \text{ and } X^\lambda(f) = \{p \in D_f : f(p) \leq \lambda\}. \quad (1)$$

This method of extracting binary images is called *thresholding* and the produced binary image is called a *level set*. $X_\lambda(f)$ is called the *upper level set of value* λ and $X^\lambda(f)$ is called the *lower level set of value* λ. We can relate groups of pixels by defining a *neighborhood* or *adjacency relation*. In particular, the adjacency relation $\mathcal{A}_4(p) = \{p + q : q \in \{(-1,0),(0,-1),(1,0),(0,1)\}\}$ that relates a pixel p to its 4 closest pixels is called 4-*connected adjacency*. The adjacency relation $\mathcal{A}_8(p) = \mathcal{A}_4(p) \cup \{p+q : q \in \{(-1,-1),(1,-1),(1,1),(-1,1)\}\}$ that relates a pixel p to its 8 closest pixels is called 8-*connected adjacency*. We denote by $\mathcal{A}$ either of the two adjacency relations. We say a pixel p is $\mathcal{A}$-adjacent to a pixel q when $p \in \mathcal{A}(q)$. In this case, we also say that p and q are *neighbors* when we do not care about the adjacency. A sequence of $\mathcal{A}-$adjacent pixels in X form an $\mathcal{A}-$path. When there exists an $\mathcal{A}-$path between two pixels they are *connected* and a group of maximally connected pixels within a binary image X is called a *connected component* (CC) of X. The set of all connected components of all upper level sets and lower level sets are respectively denoted by

$$\begin{aligned} &\mathcal{U}(f,\mathcal{A}) = \{C \in CC(\mathcal{A}(X_\lambda(f))) : \lambda \in \mathbb{Z}\} \text{ and} \\ &\mathcal{L}(f,\mathcal{A}) = \{C \in CC(\mathcal{A}(X^\lambda(f))) : \lambda \in \mathbb{Z}\}. \end{aligned} \quad (2)$$

Letting $\mathcal{S}(f,\mathcal{A})$ be either $\mathcal{U}(f,\mathcal{A})$ or $\mathcal{L}(f,\mathcal{A})$, we define the set to select the CCs between $C_1, C_2 \in \mathcal{S}(f,\mathcal{A})$ as follows

$$\texttt{between}(\mathcal{S}(f,\mathcal{A}), C_1, C_2) = \{C \in \mathcal{S}(f,\mathcal{A}) : C_1 \subset C \subset C_2\}. \quad (3)$$

2.2. Component tree

A *component tree* of an image f is defined as the tree $T_f = \big(V(T_f), E(T_f)\big)$ such that

$$\begin{aligned} &V(T_f) \text{ is either } \mathcal{U}(f,\mathcal{A}) \text{ or } \mathcal{L}(f,\mathcal{A}) \\ &E(T_f) = \{(C,N) \in V(T_f)^2 : C \subset N, \nexists C' \in \texttt{between}(V(T_f), C, N)\}. \end{aligned} \quad (4)$$

Exploiting the inclusion relation encoded in component trees, we can compactly represent them (or store in computer memory) using *compact nodes* $\widehat{N} = N \setminus \bigcup_{C \in \texttt{children}(T_f,N)} C$. The pixels of a compact node $\widehat{N}$ are called the *compact node pixels* (or CNPs for short) of the node N, and N is called the *small component* of its CNPs. Given a pixel $p \in D_f$, we denote its small component by $SC(T_f, p)$. Fig. 1 shows examples of component trees.

The compact representations of the component trees are denoted with hat symbols. Thus, the compact representation of the tree $T_f = \big(V(T_f), E(T_f)\big)$ is denoted by $\widehat{T}_f = \big(\widehat{V}(T_f), \widehat{E}(T_f)\big)$. The compact representation of the component trees of upper and lower level sets are called *max-tree* and *min-tree*, respectively. Using the compact representation of the tree, we can get the full node using the reconstruction operation $\texttt{rec\_node}(\widehat{T}_f, \widehat{N})$ which recursively unites the CNPs of N and the reconstruction of its children.

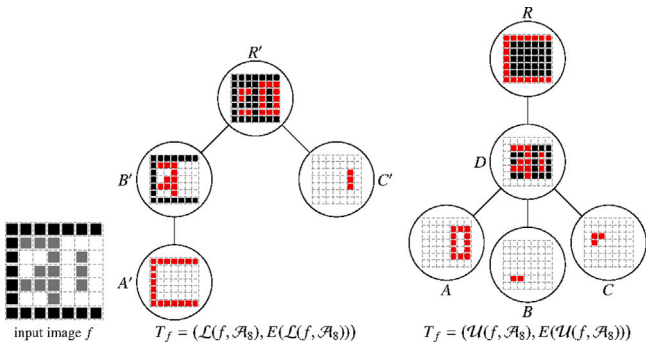

Fig. 1. A grayscale image and its compact component trees of the lower and upper level-sets. The pixels in red are the CNPs of the node, the pixels in black are pixels of the nodes that are not CNPs, and the pixels in white are the background pixels of the node.

2.3. Contour definition and a naive algorithm

Given a binary image X and an adjacency relation $\mathcal{A}$, a foreground pixel p is considered a contour pixel if it is $\mathcal{A}$-adjacent to a background pixel. The contour $\partial_\mathcal{A} X$ of X is the set of all contour pixels of X (see Fig. 2), defined as

$$\partial_\mathcal{A} X = \{p \in X : \exists q \in \mathcal{A}(p) : q \in \mathbb{Z}^2 \setminus X\}. \quad (5)$$

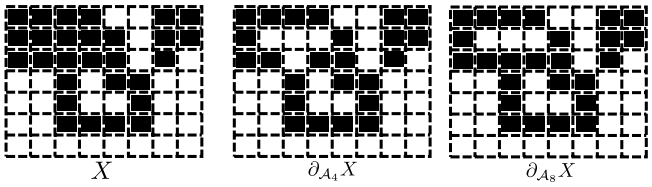

Fig. 2. Binary image X, the contour of X considering $\mathcal{A}_4$, and the contour of X considering $\mathcal{A}_8$. Black squares are foreground pixels and white squares are background pixels.

A simple approach to compute the contour of a binary image X is: (i) initialize an empty contour set and (ii) scan the foreground pixels of X and for each foreground pixel p, if there exists a background pixel adjacent to p, then add p to the contour. To compute the contour of all nodes of a component tree (min-tree or max-tree), we need to (a) reconstruct each node, and (b) apply the contour extraction to the reconstructed node. We call this approach non-incremental and node reconstruction-based. In the node reconstruction-based method, we consider that the pixels $p \notin \mathcal{D}_f$ are background pixels and any of its foreground neighbors is a contour pixel. In this approach, we call `rec−node` and the contour extraction method for each node of the tree. The contour extraction method costs $\mathcal{O}(|\mathcal{D}_f|)$ and `rec−node`$(\hat{T}_f, \hat{N})$ costs $\mathcal{O}(|\mathcal{D}_f|)$. To simplify notation, we write simply V instead of $V(T_f)$ when no confusion is likely to arise. The time complexity of contour extraction using this node-reconstruction approach is

$$\mathcal{O}(|V|)\left(\mathcal{O}(|\mathcal{D}_f|) + \mathcal{O}(|\mathcal{D}_f|)\right) = \mathcal{O}(|V| \cdot |\mathcal{D}_f|).$$

3. Incremental contour computation algorithm

We now present our incremental algorithm to compute the contours, exactly as defined in Eq. (5), of all max-tree nodes and discuss a slightly modified version of the algorithm for min-tree nodes. The main idea is to keep track of how many *background neighbors* each pixel has. At the start of processing a node, the contour pixels of its children have more than one background neighbor, and the CNPs of the node being processed have none. After processing, the contour pixels of the children with no background neighbors are removed from the current node contour and the CNPs with at least one background neighbor are included in the node contour. This process is described in Algorithm 1 and illustrated by Fig. 3 for the component tree in Fig. 1.

Initially, in lines 2–3, Algorithm 1 starts by setting up a map `contours` : $V(T_f) \to \mathcal{P}(\mathcal{D}_f)$ with empty sets for each node $N \in V(T_f)$ and a map `ncount` : $\mathcal{D}_f \to \mathbb{Z}$ with a zero for each pixel $p \in \mathcal{D}_f$. As is common in incremental algorithms, our proposed algorithm computes the contours from leaves to the root (in arbitrary order) in the loop of lines 4–16 (illustrated in the orientation top to bottom in Fig. 3). For each node N, the algorithm sets up the contour `Ncontour` as the union of the contours of its children according to the current values in `ncount` in line 5 (see `ncount` and `Ncontour` in column "before node processing" in Fig. 3). Then, it scans the node CNPs $p \in \hat{N}$ in the loop of lines 6–15 (red pixels in Fig. 3). For each p, the algorithm counts how many background neighbors p has (see lines 8–9, and green arrows in Fig. 3) and decrements the number of background neighbors of its neighbors from its children (see lines 10–11 and red arrows in Fig. 3). After the count update (see `ncount` in column "after node processing" in Fig. 3), all neighbors of p for which the number of background neighbors reaches zero are removed from N's contour (lines 12–13). Also, if p has at least one background neighbor, then it becomes a contour pixel of N (lines 14–15). In line 16, the algorithm registers the `Ncontour` (see `Ncontour` in column "after node processing" in Fig. 3) as the contour of node N in the map `contours` and in line 17 returns the contours of the max-tree nodes. Fig. 4 shows a subset of the contours computed for a photograph by Algorithm 1.

Algorithm 1: Incremental computation of the contour of component tree nodes.

Input: A grayscale image f, its max-tree $\hat{T}_f$, and an adjacency relation $\mathcal{A}$.
Output: A list of contours of each node of the tree T_f.

```
1  Function tree-extract-contour-incremental (f,
     T̂_f = (V̂(T_f), Ê(V(T_f))), A)
2    Initialize contours[N] with ∅ for each N ∈ V(T_f);
3    Initialize ncount[p] with 0 for each p ∈ D_f;
4    foreach N̂ ∈ V̂(T) from leaves to root do
5      Ncontour ← ⋃_{C∈children(T_f,N)} contours[C];
6      foreach p ∈ N̂ do
7        foreach q ∈ A(p) do
8          if q ∉ D_f or f(p) > f(q) then
9            ncount[p] ← ncount[p] + 1;
10         else if q ∈ D_f and f(p) < f(q) then
11           ncount[q] ← ncount[q] − 1;
12           if ncount[q] = 0 then
13             Ncontour ← Ncontour \ {q};
14       if ncount[p] > 0 then
15         Ncontour ← Ncontour ∪ {p};
16     contours[N] ← Ncontour;
17   return contours;
```

Algorithm 1 relies on the following max-tree property: for any pixel $p \in \mathcal{D}_f$, it holds

$$SC(T_f, p) \subseteq X_{f(p)}(f).$$

Also, it is true that for any pixel $q \in \mathcal{D}_f$ with $f(q) < f(p)$ (or $q \notin \mathcal{D}_f$), we have

$$q \notin X_{f(p)}(f) \Rightarrow q \notin SC(T_f, p).$$

Algorithm 1 uses these properties in lines 8–9. Since $p \in \hat{N}$ (line 6), then $N = SC(T_f, p)$. Also, we have $q \in \mathcal{A}(p)$ (line 7) and when $q \notin \mathcal{D}_f$ or$f(p) > f(q)$ is true, this implies that $q \notin N$ and q is a background neighbor of p. Our proposed algorithm also relies on the max-tree property that relates the small components of a pair (p, q) of $\mathcal{A}$-adjacent pixels as follows:

$$q \in \mathcal{A}(p) \text{ and } f(p) < f(q) \Rightarrow SC(T_f, p) \supset SC(T_f, q). \tag{6}$$

In particular, we apply this property in lines 10–11 of Algorithm 1 to subtract p from the background neighbor count of q. First, note that N is the small component of p (line 6) and q is $\mathcal{A}$-adjacent to p (line 7). In line 10, the condition $f(p) < f(q)$ is checked. When it is true, we have $SC(T_f, p) \supset SC(T_f, q)$, that is $SC(T_f, q)$ is a descendant of N. Thanks to the leaves-to-root order of the outermost loop of the algorithm, we have that $N' = SC(T_f, q)$ has already been processed and since $f(p) < f(q)$ holds, p was counted as a background neighbor of q in `ncount`$[q]$ when N' was being processed. However, p becomes a foreground pixel in N, and consequently it is not a background neighbor of q anymore. Thus, we decrement `ncount`$[q]$.

Also, our algorithm only increments the number of background neighbors of a pixel p in line 9. When the loop over its neighborhood is finished (lines 7–13) and it has at least one background neighbor (line 14), the algorithm includes it in the contour of N. Similarly, the background neighbor number for a pixel q, `ncount`$[q]$, is decremented only when a pixel neighbor of q is being processed in one of the ancestors of $SC(T_f, q)$. When q is a contour pixel of a child of N and `ncount`$[q]$ reaches zero during the algorithm execution for

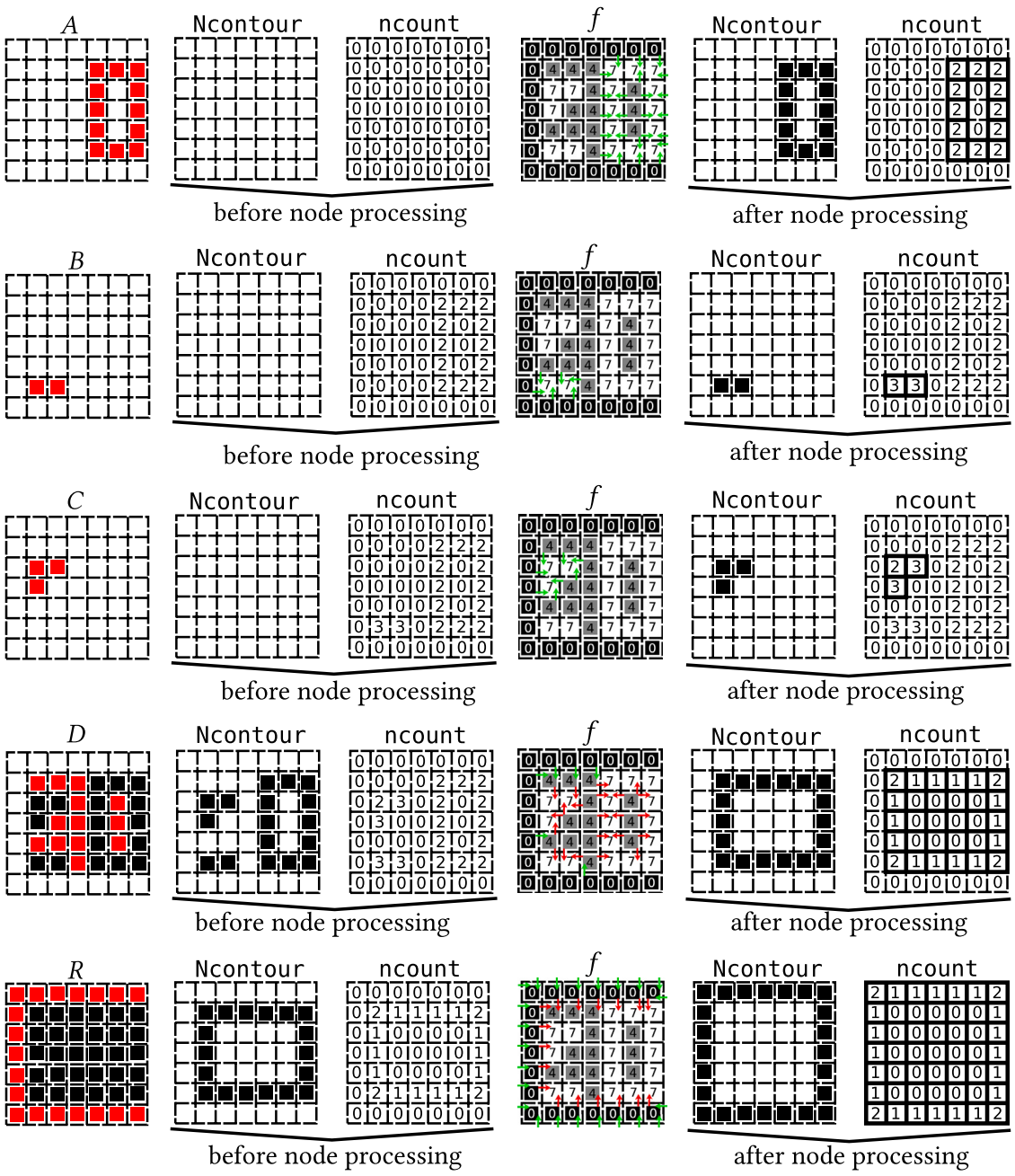

Fig. 3. Graphical representation of the execution of Algorithm 1 for the component tree depicted in Fig. 1. Each row represents the processing of a node, which is divided into "before node processing" and "after node processing". The green arrows point to the pixels in which the number of background neighbors has been incremented. The red arrows point to the pixels in which the number of background neighbors has been decremented. The red pixels represent the CNPs of the nodes and the black pixels represent the foreground pixels that are not CNPs.

N, q is removed from the contour of N and it will not be a contour pixel again because from N up to the root, it is surrounded by foreground pixels. Consequently, at the end of the loop in lines 4–16, we have `Ncontour` $= \partial_{\mathcal{A}} N$ and `ncount`$[p]$ to correspond to the number of background neighbors of p considering node N, for any $p \in N$.

The reasoning of our algorithm is slightly different when the input parameters are $\mathcal{A}_8$-contour and $\mathcal{A}_4$-component tree because the property described in Eq. (6) does not hold for diagonally connected pixels of $\mathcal{A}_8$. However, our proposed algorithm, still, correctly computes the contours as discussed in Appendix A.

Algorithm 1 can be adapted to work with min-trees instead of max-trees by inverting the order relation in lines 8 and 10, that is changing '>' to '<' in line 8 and '<' to '>' in line 10.

3.1. Complexity analysis

The performance of our proposed incremental algorithm heavily depends on the structure of the input morphological tree. To effectively describe the complexity, let c_N denote the number of children of a node $N \in V$ and δ indicate the length of the longest contour between all nodes of the input tree T_f.

For managing set operations necessary for storing contours, we utilize red–black trees and hash maps from the C++ STL. In this context, n specifically denotes the current number of elements in the set, affecting the complexity of operations. Insertion and removal in a red–black tree are executed in $\mathcal{O}(\log n)$, while a hash map performs these operations in $\mathcal{O}(n)$ in the worst-case scenario and $\mathcal{O}(1)$ on average.

(a) Markers for selecting nodes (b) The contour of selected nodes

Fig. 4. (a) 15 markers (green circles) for node selection overlayed over a grayscale image and (b) the grayscale image overlayed with the contours of the selected nodes. The contours are computed using our proposed approach (Algorithm 1). We selected the nodes by finding the largest small component of the marker's pixels. Note that this is only a small subset of the complete set of contours our proposed method computes. The input image is taken from [19], filtered by a height opening, and cropped for visualization purposes.

Focusing on the red–black tree, the complexity of key operations in the algorithm is detailed as follows:

- Line 5 handles the union of all children's contours for a node N, inserting each pixel into the set individually. This union operation is therefore computed in $\mathcal{O}(c_N \delta \log \delta)$ for a single node $N \in V$ and in $\mathcal{O}(\sum_{N \in V} c_N \delta \log \delta) = \mathcal{O}(\delta \log \delta \sum_{N \in V} c_N)$ when applied to all nodes. Since T_f is a tree, $\sum_{N \in V} c_N = |V| - 1$, which results in a comprehensive complexity of $\mathcal{O}(|V| \delta \log \delta)$.
- Lines 8–13 entail nested loops over the tree nodes, their CNPs, and pixel neighborhoods. Since each pixel is processed only once due to the partitioning properties of CNPs, the removal of contour pixels incurs a cost of $\mathcal{O}(\log \delta)$. Hence, the total complexity for this segment is $\mathcal{O}\left(|D_f| \log \delta\right)$.
- Line 15 involves adding a pixel to the set, which is performed once for each pixel during each iteration of the loop, resulting in a complexity of $\mathcal{O}(|D_f| \log \delta)$ for the entire image domain.

The final result for the time complexity is:

$$\mathcal{O}(|V| \delta \log \delta) + \mathcal{O}(|D_f| \log \delta).$$

Similarly, the worst-case time complexity of our proposed method using hash maps is:

$$\mathcal{O}(|V| \delta^2) + \mathcal{O}(|D_f| \delta).$$

Compared to the simple approach presented in Section 2.3, with complexity $\mathcal{O}(|V| \cdot |D_f|)$, our method is expected to be significantly faster since δ is usually much smaller than $|D_f|$. Regarding the choice of the data structure, it is worth noting that due to the average cost of using hash maps being lower than the average cost of using red–black trees, the hash map approach is expected to run faster on average. To confirm the accuracy of our analysis, we conducted experiments that are presented in Section 4.

4. Run-time experiments

We have conducted run-time experiments using the validation dataset of the Occluded RoadText Challenge from the Robust Reading Competition portal [19]. The dataset consists of 200 color images with scenes taken while driving. The images are 1920 × 1080 large and contain natural elements, urban infrastructure, text, crowed movement and other elements. We converted the images to grayscale and extended the dataset by scaling down the images to $960 \times 540, 480 \times 270, 240 \times 135$, and 120×68 using the `rescale` method from the module `transform` of scikit-image [20]. Our setup was a laptop with Ubuntu 19.10, memory of 16GiB, and a processor Intel® Core™ i7-8750H CPU (2.20 GHz ×12). The implementation is single-core and was developed in C++ on top of the *morphotree* library [21].

Table 1
Runtime experimental results grouped by image resolution for the validation dataset of the Occluded RoadText Challenge with rescaled images.

Resolution	Non-incr.	incr. RB-trees		incr. HM	
	Time (ms)	Time (ms)	Speed-up	Time (ms)	Speed-up
1920 × 1080	26 008.0	1619.8	16.9	1365.5	19.8
960 × 540	6 475.3	447.4	15.0	405.8	16.5
480 × 270	1 567.0	125.3	12.8	120.2	13.4
240 × 135	401.2	38.3	10.6	37.6	10.9
120 × 68	105.8	11.7	9.2	11.8	9.1
Mean	6 911.4	448.5	12.9	338.1	14.0

We run our experiments for the node-reconstruction approach described in Section 2.3 and Algorithm 1 using red–black trees and hash maps. We run the experiment 12 times, each time the order of the methods' execution was changed. The experiment produced a table with the average runtime of the used methods for each image in the dataset. The table was used to compute the speed-up of our incremental algorithm over the non-incremental approach. The algorithm's implementations, the source codes, the full table, and the data analysis process are available in [22].

Table 1 presents the average runtime and speed-up grouped by image resolution. Our incremental approaches are faster than the non-incremental approach for all image scales and the speed-up of our method increases as the image scale increases.

We visually present the runtime of the contour extraction method relationship to the resolution of the images in the dataset in Fig. 5.

The experimental results show that our incremental approach is faster than the non-incremental approach. Table 1 shows that our proposed method runs 12.9 or 14.0 times faster than the node-reconstruction approach depending on the data structure used for storing the contours. Fig. 5 shows that the run-time of our approach grows approximately linearly with the image resolution. Appendix B presents a table and a plot comparing the run-time of our approach and the node-reconstruction approach for trees of different sizes.

The first step of our algorithm is to push the contour pixels of all children into the contour of the node being processed. Some of these contour pixels are later removed when the algorithm identifies the former contour pixels that have no background neighbors anymore. This process of including and then removing the contour pixels does not happen in the non-incremental approach. Thus, the worst-case scenario of our method in relation to the non-incremental approach would be when every contour pixel of the children pushed into the node being processed is removed. On the other hand, the best scenario for the

Table 2
Chessboard pattern experiment. N^2 determines image size. The speed-ups are computed by dividing the non-incremental method run-time over the incremental method run-time.

N^2	Non-incr.	incr. RB-trees		incr. HM	
	Time (ms)	Time (ms)	Speed-up	Time (ms)	Speed-up
32 × 32	<1	<1	Undefined	<1	Undefined
64 × 64	1	<1	Undefined	<1	Undefined
128 × 128	4.1	2.2	1.9	2.8	1.5
256 × 256	19.2	13.7	1.4	12.3	1.6
512 × 512	71.5	59.2	1.2	48.7	1.5
1024 × 1024	285.1	298.3	1.0	195.9	1.5
2048 × 2048	1139.7	1437.7	0.8	750.4	1.5
4096 × 4096	4584.6	6297.0	0.8	3103.8	1.5
Mean	678.43	901.0	Undefined	458.2	Undefined

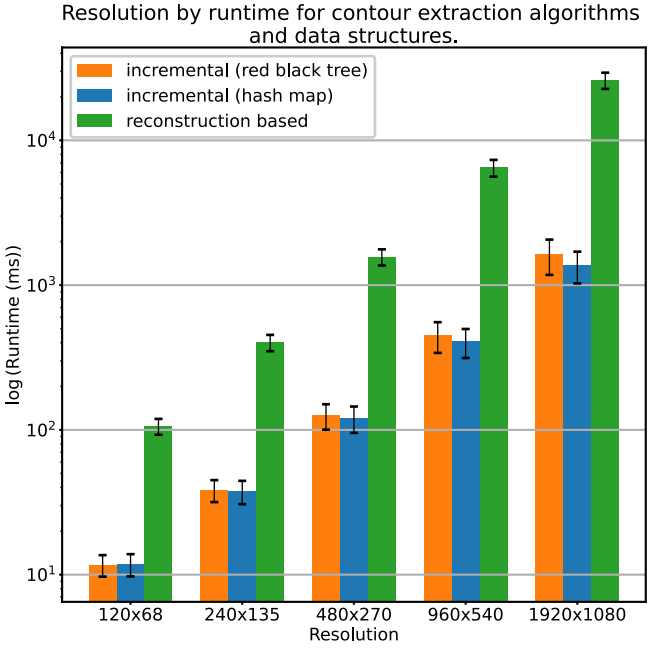

Fig. 5. Bar plot of the image resolutions and runtime of the incremental (using red–black trees and hash maps) and non-incremental approaches for computing node contours. The black vertical bars represent the standard deviation.

non-incremental approach in relation to our incremental method is a tree with few nested nodes To analyze our proposed algorithm in the worst-case scenario compared to the non-incremental approach, we perform experiments on synthesized chessboard pattern images. These images produce 4-connected adjacency max-trees with the root node containing a child for each bright pixel.

In addition, every bright pixel is a contour pixel in its small component that should be removed (except by the ones with neighbors outside image domain) at the root node contour pixel set.

For this experiment, we generated a dataset of chessboard pattern images of size $N \times N$ for $N \in \{32, 64, 128, 256, 512, 1024, 2048, 4096\}$ and computed the time elapsed for the execution of the incremental (Algorithm 1 using red–black trees and hash maps) and non-incremental (node-reconstruction) methods (see Section 2.3). We compute the speed-up of the incremental methods over the non-incremental method. The registered data is shown in Table 2.

Table 2 shows that our algorithm with the red–black tree implementation has a similar run-time to the non-incremental method up to the image of size 512 × 512. It performed slower than the non-incremental method for images larger than 512 × 512. Our incremental approach using the hash map was faster than the non-incremental approach for all tested images. It is worth noting that the worst-case scenario for our algorithm is unlikely when dealing with natural and synthesized images.

5. Proposed applications

In this section, we propose several applications which could benefit from our contour computation method.

Interactive morphological tree manipulation: There are different approaches to interactive morphological tree manipulation for various image processing and analysis tasks [5,23–25]. A common interactive morphological tree application requirement is visually selecting tree nodes by either clicking on regions of interest in the image or by clicking on the nodes in a graphical representation of the tree. In both cases, it is beneficial to highlight the region by painting the selected nodes' pixels. A similar approach could paint the contour of the selected nodes so the user could see the contour of the selected nodes and the region of the image that the node contains. Our proposed algorithm could be used to pre-compute the contours and when the user selects the nodes, paint the stored contours of the selected nodes.

Distance transform and skeleton computation: Distance Transform (DT) and Medial Axis Transform (or skeleton) computation are important binary image transformations that can be used in many different applications. The Dense Medial Descriptor (DMD) uses those transformations in a grayscale image by computing them level set by level set. DMD and its variants have been successfully applied to image segmentation, artistic editing, image manipulation, and compression [5,26,27]. Although the DMD approach uses a parallel GPU-based method to compute the transformations, they are performed a single level set at a time, i.e., the computation of a level set is not re-used in the level sets that contain it. Many DT and skeleton computation algorithms are initialized by assigning special values to the contour pixels [28,29]. So, our incremental contour computation algorithm could be used to save time in the contour pixel computation step of DT and skeleton algorithms. Indeed, Silva et al. [30] have adapted our proposed algorithm to quickly compute DT using Differential Image Foresting Transform.

Computing contour information: We can extract different information from the contours of morphological tree nodes such as perimeter and difference between the contours and their non-contour neighbors. The contours of the nodes would enable to compute the Mumford-Shah energy functional [16,17] on component trees.

6. Conclusion

The paper has presented a novel method to incrementally compute the contour of max-trees or min-trees by keeping track of the number of background neighbors for each pixel.

We have analyzed the time complexity of our method and experimentally showed that it is faster than a naive, non-incremental approach depending on the underlying data structure adopted. We have discussed how our method could be applied to interactive morphological tree manipulation, distance transform and skeleton computation, and contour information computation. Based on the results presented, we believe that our method represents a valuable contribution to the field of image analysis and holds the potential to be applied in a

wide range of practical applications. As future work, we will work on adapting our proposed algorithm for incremental contour tracking in component trees.

CRediT authorship contribution statement

Dennis J. Silva: Writing – review & editing, Writing – original draft, Software, Methodology, Investigation, Conceptualization. **Jiří Kosinka:** Writing – review & editing, Writing – original draft, Supervision, Project administration, Methodology, Investigation, Conceptualization. **Ronaldo F. Hashimoto:** Writing – review & editing, Supervision, Project administration, Methodology, Investigation, Conceptualization. **Jos B.T.M. Roerdink:** Writing – review & editing, Supervision, Project administration, Conceptualization. **Alexandre Morimitsu:** Writing – review & editing, Methodology. **Wonder A.L. Alves:** Writing – review & editing, Methodology.

Declaration of Generative AI and AI-assisted technologies in the writing process

During the preparation of this work, the author(s) used ChatGPT in order to improve readability and language. After using this tool/service, the author(s) reviewed and edited the content as needed and take(s) full responsibility for the content of the publication.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

Dennis J. da Silva and Wonder A.L. Alves acknowledge CNPq - Conselho Nacional de Desenvolvimento Científico e Tecnológico (Proc. 141422/2018-1 and Proc. 428720/2018-7) for financial support. Ronaldo F. Hashimoto and Wonder A.L. Alves acknowledge FAPESP - Fundação de Amparo a Pesquisa do Estado de São Paulo (Proc. 2015/22308-2 and 2018/15652-7) for financial support.

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.patrec.2024.11.019.

Data availability

We also cite our Github repository with the source code developed for the paper and experiments.

References

[1] P. Salembier, M.H.F. Wilkinson, Connected operators, IEEE Signal Process. Mag. 26 (6) (2009) 136–157.
[2] L. Neumann, J. Matas, Real-time scene text localization and recognition, in: Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on, IEEE, 2012, pp. 3538–3545.
[3] J. Climent, L.S. Oliveira, A new algorithm for number of holes attribute filtering of grey-level images, Pattern Recognit. Lett. 53 (2015) 24–30.
[4] W.A. Alves, C.F. Gobber, S.A. Araújo, R.F. Hashimoto, Segmentation of retinal blood vessels based on ultimate elongation opening, in: International Conference Image Analysis and Recognition, Springer, 2016, pp. 727–733.
[5] J. Wang, D.J. Silva, J. Kosinka, A. Telea, R.F. Hashimoto, J.B. Roerdink, Interactive image manipulation using morphological trees and spline-based skeletons, Comput. Graph. 108 (2022) 61–73.
[6] R. Souza, L. Tavares, L. Rittner, R. Lotufo, An overview of max-tree principles, algorithms and applications, in: 2016 29th SIBGRAPI Conference on Graphics, Patterns and Images Tutorials (SIBGRAPI-T), 2016, pp. 15–23.
[7] L. Najman, M. Couprie, Building the component tree in quasi-linear time, IEEE Trans. Image Process. 15 (11) (2006) 3531–3539.
[8] D.J. Silva, W.A.L. Alves, A. Morimitsu, C.F. Gobber, R.F. Hashimoto, Incremental bit-quads count in tree of shapes, in: B. Burgeth, A. Kleefeld, B. Naegel, N. Passat, B. Perret (Eds.), Mathematical Morphology and Its Applications To Signal and Image Processing, Springer International Publishing, Cham, 2019, pp. 162–173.
[9] D.J. Silva, W.A. Alves, R.F. Hashimoto, Incremental bit-quads count in component trees: Theory, algorithms, and optimization, Pattern Recognit. Lett. 129 (2020) 33–40.
[10] R.C. Gonzalez, R.E. Woods, Digital Image Processing, fourth ed., Pearson, 2018.
[11] O.A. Tapia-Dueñas, H. Sánchez-Cruz, Context-free grammars to detect straight segments and a novel polygonal approximation method, Signal Process., Image Commun. 91 (2021) 116080.
[12] H. Sánchez-Cruz, O.A. Tapia-Dueñas, F. Cuevas, Polygonal approximation using a multiresolution method and a context-free grammar, in: J.A. Carrasco-Ochoa, J.F. Martínez-Trinidad, J.A. Olvera-López, J. Salas (Eds.), Pattern Recognition, Springer International Publishing, Cham, 2019, pp. 261–270.
[13] V.M. Garcia-Molla, P. Alonso-Jordá, R. García-Laguía, Parallel border tracking in binary images using GPUs, J. Supercomput. 78 (7) (2022) 9817–9839.
[14] S. Gupta, S. Kar, Algorithms to speed up contour tracing in real time image processing systems, IEEE Access 10 (2022) 127365–127376.
[15] J. Seo, S. Chae, J. Shim, D. Kim, C. Cheong, T.-D. Han, Fast contour-tracing algorithm based on a pixel-following method for image sensors, Sensors 16 (3) (2016) http://dx.doi.org/10.3390/s16030353.
[16] Y. Xu, T. Géraud, L. Najman, Salient level lines selection using the mumford-shah functional, in: 2013 IEEE International Conference on Image Processing, 2013, pp. 1227–1231.
[17] Y. Xu, T. Géraud, L. Najman, Hierarchical image simplification and segmentation based on Mumford–Shah-salient level line selection, Pattern Recognit. Lett. 83 (2016) 278–286, Efficient Shape Representation, Matching, Ranking, and its Applications.
[18] L. Najman, T. Géraud, Discrete set-valued continuity and interpolation, in: C.L.L. Hendriks, G. Borgefors, R. Strand (Eds.), Mathematical Morphology and Its Applications To Signal and Image Processing, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 37–48.
[19] G. Tom, M. Mathew, A. Mondal, J. Weinman, D. Karatzas, C.V. Jawahar, Robust reading competition - occluded RoadText, 2024, https://rrc.cvc.uab.es/?ch=29, (Accessed 23 July 2024).
[20] Scikit-image, API reference, 2024, https://scikit-image.org/docs/stable/api/skimage.transform.html#skimage.transform.rescale. (accessed 23 July 2024).
[21] D.J. Silva, Morphotree, 2022, https://github.com/dennisjosesilva/morphotree. (Accessed 1 December 2022).
[22] D.J. Silva, R.F. Hashimoto, J. Kosinka, J.B.T.M. Roerdink, W.A.L. Alves, A. Mormitsu, Incremental component tree contour computation - Experiments analysis webpage and source code, 2023, https://github.com/dennisjosesilva/incremental-contour-experiments. (Accessed 29 June 2023).
[23] M.A. Westenberg, J.B.T.M. Roerdink, M.H.F. Wilkinson, Volumetric attribute filtering and interactive visualization using the max-tree representation, IEEE Trans. Image Process. 16 (12) (2007) 2943–2952.
[24] L.A. Tavares, R.M. Souza, L. Rittner, R.C. Machado, R.A. Lotufo, Interactive max-tree visualization tool for image processing and analysis, in: 2015 International Conference on Image Processing Theory, Tools and Applications, IPTA, 2015, pp. 119–124.
[25] L.A. Tavares, R.M. Souza, L. Rittner, R.C. Machado, R.A. Lotufo, A max-tree simplification proposal and applications for the interactive max-tree visualization tool, in: 2016 29th SIBGRAPI Conference on Graphics, Patterns and Images, SIBGRAPI, 2016, pp. 313–320.
[26] M. Van Der Zwan, Y. Meiburg, A. Telea, A dense medial descriptor for image analysis, in: Proc. VISAPP, 2013, pp. 285–293.
[27] J. Wang, J. Kosinka, A. Telea, Spline-based dense medial descriptors for lossy image compression, J. Imaging 7 (8) (2021).
[28] A.C. Telea, Data Visualization: Principles and Practice, CRC Press, 2014, pp. 368–372.
[29] A. Falcão, L. da Fontoura Costa, B. da Cunha, Multiscale skeletons by image foresting transform and its application to neuromorphometry, Pattern Recognit. 35 (7) (2002) 1571–1582.
[30] D.J. Silva, P.A.V. Miranda, W.A.L. Alves, R.F. Hashimoto, J. Kosinka, J.B.T.M. Roerdink, Differential maximum euclidean distance transform computation in component trees, in: S. Brunetti, A. Frosini, S. Rinaldi (Eds.), Discrete Geometry and Mathematical Morphology, Springer Nature Switzerland, Cham, 2024, pp. 67–79.