Differential Maximum Euclidean Distance Transform Computation in Component Trees

Da Silva, Dennis; Vechiatto Miranda, Paulo André; Alves, Wonder A.L.; Hashimoto, Ronaldo F.; Kosinka, Jiri; Roerdink, Jos B.T.M.

Published in:
Discrete Geometry and Mathematical Morphology

DOI:
10.1007/978-3-031-57793-2_6

Differential Maximum Euclidean Distance Transform Computation in Component Trees

Dennis J. Silva[1,2(✉)], Paulo André Vechiatto Miranda[1], Wonder A. L. Alves[3], Ronaldo F. Hashimoto[1], Jiří Kosinka[2], and Jos B. T. M. Roerdink[2]

[1] Institute of Mathematics and Statistics, University of São Paulo, R. do Matão, 1010, CEP 05508-090, Butantã, São Paulo, SP, Brazil
{dennis,pmiranda,ronaldo}@ime.usp.br
[2] Bernoulli Institute for Mathematics, Computer Science and Artificial Intelligence, University of Groningen, Nijenborgh 9, 9747 AG Groningen, The Netherlands
{d.j.da.silva,j.kosinka,j.b.t.m.roerdink}@rug.nl
[3] Informatics and Knowledge Management Graduate Program, Nove de Julho University, R. Vergueiro, 235/249, CEP 01525-000, Liberdade, São Paulo, SP, Brazil

Abstract. The distance transform is an important binary image transformation that assigns to each foreground pixel the distance to the closest contour pixel. Among other applications, the maximum distance transform (DT) value can describe the thickness of the connected components of the image. In this paper, we propose using the maximum distance transform value as an attribute of component tree nodes. We present a novel algorithm to compute the maximum DT value of all connected components of a greyscale image in a differential way by joining an incremental method for contour extraction in component trees and the Differential Image Foresting Transform (DIFT). We save processing time by reusing the DIFT subtrees rooted at the contour points (DIFT seeds) of a node in its ancestors until those points are not contour points anymore. We experimentally show that we can compute the maximum distance attribute twice as fast as the node-reconstruction approach. Our proposed attribute is increasing and its applicability is exemplified by the design of an extinction value filter. The ability to select thin connected components, like cables, of our filter is compared to filters using other increasing attributes in terms of their parameters and their resulting images.

Keywords: Component tree · Distance transform · Image Foresting Transform

1 Introduction

Component trees are powerful full image representations used in different image processing and analysing tasks, including designing connected operators [14],

S. Brunetti et al. (Eds.): DGMM 2024, LNCS 14605, pp. 67–79, 2024.
https://doi.org/10.1007/978-3-031-57793-2_6

text location [12], eye vessel segmentation [1], interactive image manipulation [19], and many others. In this representation, the connected components (CCs) of the level-sets are the nodes of the tree and the subset relation of these CCs is encoded in the parenthood relationship. An important step in processing images using component trees is describing the nodes through attributes. Different attributes such as area, perimeter, number of holes, height, and volume have been successfully applied in different applications [1,12,14].

The distance transform is an important binary image transformation that assigns for each foreground pixel the distance to the closest contour pixel. Although widely used in image processing [18], the distance transform applied to component trees has not received much attention in the mathematical morphology community. In particular, we note that the maximum distance transform value of the CCs describes the thickness of the CC and could be a useful increasing attribute for the nodes of a component tree. Thus, in this paper, we propose the maximum distance transform value as an attribute of component trees. To make it feasible, we propose a novel algorithm that combines a recent incremental approach to compute the contours of component trees with the computation of the distance transform using the Differential Image Foresting Transform (DIFT), which can reuse a previously computed IFT by only recomputing the IFT on the removed and inserted seeds. We experimentally demonstrate that our algorithm is on average twice as fast as a non-differential approach and that the proposed attribute can be used in extinction value filters to remove thin objects which other increasing attributes cannot easily do.

The remainder of this paper is organised as follows. We recall some useful definitions and notations in Sect. 2. We describe the proposed attribute and the differential algorithm in Sect. 3. In Sect. 4, we report our run-time experiments and present a simple example application of the proposed attribute to extinction filters. We conclude the paper in Sect. 5.

2 Background

2.1 Images

A *greyscale image* is a function $f : D_f \to \mathbb{K}_f$ where $D_f \subseteq \mathbb{Z}^2$ is a regular grid of *pixels* and $\mathbb{K}_f = \{0, 1, \ldots, K_f - 1\} \subset \mathbb{Z}$ is the greylevel set. When $K_f = 2$, f is a *binary image* and we represent it as a set $X = \{p \in D_f : f(p) = 1\}$. We say a pixel p is a *foreground pixel* if $p \in X$ and a *background pixel* if $p \in \mathbb{Z}^2 \setminus X$. We can extract binary images from a greyscale image by selecting the pixels whose grey level is greater or equal to, respectively less than or equal to, a threshold value $\lambda \in \mathbb{Z}$ as its foreground pixels. This *thresholding* operation is denoted by

$$\begin{aligned} [f \geq \lambda] &= \{p \in D_f : f(p) \geq \lambda\}, \\ [f \leq \lambda] &= \{p \in D_f : f(p) \leq \lambda\}. \end{aligned} \tag{1}$$

We call $[f \geq \lambda]$ and $[f \leq \lambda]$), resp.,the *upper* and *lower level-set* of f wrt. λ.

We relate spatially close pixels by an adjacency relation. In particular, we define $\mathcal{N}_4(p) = \{p + q : q \in \{(-1,0),(0,-1),(1,0),(0,1)\}\}$ and $\mathcal{N}_8(p) = \mathcal{N}_4(p) \cup \{p + q : q \in \{(-1,-1),(1,-1),(1,1),(-1,1)\}\}$ as the 4-*connected* and 8-*connected adjacency relation* (or 4- and 8-connected neighbourhood) of p, resp. We denote an arbitrary neighbourhood of pixel p by $\mathcal{N}(p)$. If $q \in \mathcal{N}(p)$, we say q is a *neighbour* of (or adjacent to) p, and that p and q are *neighbours* (adjacent pixels). Given a set of pixels $X \subseteq \mathbb{Z}^2$, we denote the set of pairs of adjacent pixels by

$$\mathcal{A}(X) = \{(p,q) \in X^2 : q \in \mathcal{N}(p)\}. \tag{2}$$

Given a greyscale image $f : D_f \to \mathbb{K}_f$ and an adjacency relation $\mathcal{A}$, we can enrich the image by representing it by a *vertex-weighted graph* $\mathcal{G}_{f,\mathcal{A}} = (f, D_f, \mathcal{A}(D_f))$, where its vertices are pixels, its edges are defined by the adjacency relation, and the weights of the vertices are the pixel grey levels. Using the set representation of the binary image, we can also create the graph $G_{X,\mathcal{A}} = (X, \mathcal{A}(X))$ that enriches the binary image representation. Using the graph representation of binary images, we define a *path* $\pi(p,q)$ between two pixels $p, q \in X$ as the sequence $(r_0, r_1, \ldots, r_n)$ of pixels in X with $r_i \in \mathcal{N}(r_{i+1})$ for $0 \leq i < n$, $r_0 = p$ and $r_n = q$. If there exists such a path $\pi(p,q)$ in X, we say p and q are *connected*, otherwise, we say they are *disconnected*. A subset of pixels $S \subseteq X$ is called *connected* if all pairs of pixels $p, q \in S$ are connected in S. If S is maximally connected, it is called a *connected component* of X. We denote the set of connected components of a graph $G_{X,\mathcal{A}}$ by $CC(G_{X,\mathcal{A}})$, and the connected component containing pixel $p \in \mathbb{Z}^2$ by $CC(G_{X,\mathcal{A}}, p)$, with $CC(G_{X,\mathcal{A}}, p) = \emptyset$ if $p \notin X$. Further, we define the family of connected components of the upper and lower level-sets by

$$\begin{aligned} \mathcal{U}(f,\mathcal{A}) &= \{C \in CC(G_{[f \geq \lambda],\mathcal{A}}) : \lambda \in \mathbb{Z}\}, \\ \mathcal{L}(f,\mathcal{A}) &= \{C \in CC(G_{[f \leq \lambda],\mathcal{A}}) : \lambda \in \mathbb{Z}\}. \end{aligned} \tag{3}$$

2.2 Component Trees

A (rooted directed) *tree* T is an acyclic (directed) graph $T = (V(T), E(T))$, where $V(T)$ is the set of vertices/nodes and $E(T)$ is the set of (directed) edges. Given a node $N \in V(T)$, a tree T supports the following operations:

$$\begin{aligned} \texttt{parent}(N,T) = P &\Leftrightarrow (N,P) \in E(T), \\ \texttt{children}(N,T) &= \{C \in V(T) : (C,N) \in E(T)\}, \\ \texttt{rootTree}(T) = N &\Leftrightarrow \nexists P \in V(T) : P = \texttt{parent}(N,T). \end{aligned} \tag{4}$$

A *component tree* $T_f = (V(T_f), E(T_f))$ is a tree such that

$$\begin{aligned} V(T_f) &\text{ is either } \mathcal{U}(f,\mathcal{A}) \text{ or } \mathcal{L}(f,\mathcal{A}), \\ E(T_f) &= \{(N,P) : N, P \in V(T_f), N \subset P, \texttt{between}(N,P,T_f) = \emptyset\}, \end{aligned}$$

where

$$\texttt{between}(N,P,T_f) = \{P' \in V(T_f) : N \subset P' \subset P\}. \tag{5}$$

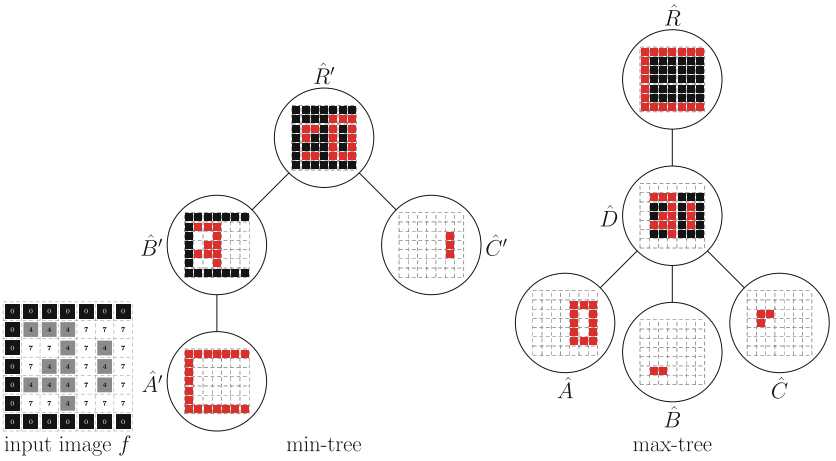


Fig. 1. Max-tree and min-tree of the input image f. The red pixels are the CNPs of the nodes, the black pixels are pixels of the node that are stored in a descendant node and the white pixels are the background pixels. (Color figure online)

We can associate the grey level at which a node N first appears during greylevel decomposition of f with the node in the component tree as follows:

$$\texttt{level}(N, T_f) = \begin{cases} \inf_{p \in N} f(p), & \text{if } V(T_f) = \mathcal{U}(f, \mathcal{A}), \\ \sup_{p \in N} f(p), & \text{if } V(T_f) = \mathcal{L}(f, \mathcal{A}). \end{cases} \tag{6}$$

Naive handling of component trees by computers can be costly due to the pixels that belong to many nodes. To alleviate this, each pixel is stored only at the first node where it becomes a foreground pixel. The full node is then formed by the pixels it stores and the pixels stored by its descendants. Such nodes are called *compact nodes* and are formally defined by $\hat{N} = N \setminus \bigcup_{C \in \texttt{children}(T_f, N)} \cdot$ We call a pixel $p \in \hat{N}$ a *compact node pixel* or CNP for short. Since the compact node that stores a pixel p is the smallest CC in the tree containing p, we call it the *small component* of p and denote it by $\mathcal{SC}(T_f, p) = N \Leftrightarrow p \in \hat{N}$. This leads to the definition of the *compact component tree* $\hat{T}_f = (\hat{V}(T_f), \hat{E}(T_f))$:

$$\hat{V}(T_f) = \{\hat{N} : N \in V(T_f)\},$$
$$\hat{E}(T_f) = \{(\hat{A}, \hat{B}) : \hat{A}, \hat{B} \in \hat{V}(T_f), (A, B) \in E(T_f)\}.$$

The compact representations of component trees built using the upper and lower level-sets are called *max-tree* and *min-tree*, respectively; see Fig. 1.

Component trees are useful tools for image processing because we can associate *attributes* to their nodes. Formally, an attribute is a function that maps component tree nodes to a set that describes some feature of the nodes. The function `level` is an example of an attribute that maps nodes to their associated grey level. Other common examples are area, volume, and perimeter [11].

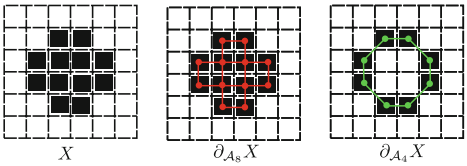


Fig. 2. Contour definition and the impact of the adopted adjacency relation. X is a binary image, $\partial_{\mathcal{A}_8}(X)$ is the contour of X extracted using $\mathcal{A}_8$ and $\partial_{\mathcal{A}_4}(X)$ is the contour of X extracted using $\mathcal{A}_4$. The graph in red (middle) shows that $\partial_{\mathcal{A}_8}(X)$ produces an $\mathcal{A}_4$-contour and the graph in green (right) shows that $\partial_{\mathcal{A}_4}(X)$ produces an $\mathcal{A}_8$-contour. (Color figure online)

Attributes that increase as we move from the leaves to the root are called *increasing*. Formally, an attribute $\texttt{attr}$ is increasing if for $A, B \in V(T_f) : A \subset B$ implies $\texttt{attr}(A) \leq \texttt{attr}(B)$; e.g. area. Otherwise, the attribute is called *non-increasing*; e.g. perimeter. An *incremental algorithm* is an algorithm that computes an attribute of a node using the node's CNPs and the attribute value of its children [17]. When there exists an incremental algorithm that computes an attribute, we say that the attribute is *incremental*.

2.3 Contours

Given a binary image X and an adjacency relation $\mathcal{A}$, we say that a pixel $p \in X$ is a contour pixel if it is adjacent to a background pixel $q \in \mathbb{Z}^2 \setminus X$. The set of contour pixels of X is the *contour* of X [10]:

$$\partial_{\mathcal{A}}(X) = \{p \in X : \exists q \in \mathcal{N}(p), q \in \mathbb{Z}^2 \setminus X\}. \tag{7}$$

When extracting contours, it is important to note that the connectivity of the adjacency relation used to compute the contour is dual to the connectivity of the resulting contour. That is, $\partial_{\mathcal{A}_8}(X)$ produces $\mathcal{A}_4$-connected contours and $\partial_{\mathcal{A}_4}(X)$ results in $\mathcal{A}_8$-connected contours; see Fig. 2.

2.4 Differential Image Foresting Transform

The *Image Foresting Transform* (IFT) algorithm is a generalization of Dijkstra's algorithm for multiple sources (seeds) and more general connectivity functions. It can be applied to an image graph to develop image operators based on optimum connectivity [7]. In its seeded version, as used in this work, the search for optimal paths is restricted to paths starting in a set of seeds $\mathcal{S} \subset D_f$.

For a given image graph $G_{X,\mathcal{A}}$ and seed set $\mathcal{S} \subset X \subseteq D_f$, let $\pi^*(s, p)$ with $s \in \mathcal{S}$ denote the path computed by IFT ending at pixel $p \in X \subseteq D_f$. Computed paths are stored in a predecessor map $\texttt{pred} : D_f \rightarrow D_f^* = D_f \cup \{\textbf{NIL}\}$, such that for each pixel $p \in X \setminus \mathcal{S}$, $q = \texttt{pred}(p)$ indicates the path's predecessor node of p in the computed path $\pi^*(s, p)$ and $\texttt{pred}(s) = \textbf{NIL}$ indicates that s is the

origin of the path (root node). A root map $\texttt{root} : D_f \to D_f$ is used to explicitly store the origin of paths $\pi^*(s,p)$, such that $\texttt{root}(p) = s$ and $\texttt{pred}(s) = \textbf{NIL}$.

Let $\Pi_q(G_{X,\mathcal{A}})$ be the set of all possible paths in graph $G_{X,\mathcal{A}}$ ending at pixel $q \in X$ and $\Pi(G_{X,\mathcal{A}}) = \bigcup_{q \in X} \Pi_q(G_{X,\mathcal{A}})$ indicate the set of all possible paths in $G_{X,\mathcal{A}}$. A *connectivity function* $\Psi : \Pi(G_{X,\mathcal{A}}) \to \mathbb{R}^+$ computes a cost $\Psi(\pi(p,q))$ for any path $\pi(p,q)$. A path $\pi(p,q)$ is *optimal* if $\Psi(\pi(p,q)) \leq \Psi(\pi'(x,q))$ for any other path $\pi'(x,q) \in \Pi_q(G_{X,\mathcal{A}})$, irrespective of its starting point x. During the IFT calculation, a cost map $\texttt{cost} : D_f \to \mathbb{R}^+$ is used to store the costs of the computed paths $\pi^*(s,p)$, such that $\texttt{cost}(p) = \Psi(\pi^*(s,p))$ for all $p \in X$.

In the case of a sequence of IFT applications for different sets of seeds modified by insertion and/or removal of seeds and the same connectivity function, the *Differential Image Foresting Transform* (DIFT) allows the updating of paths stored in `pred` and other maps in a time proportional to the size of the modified regions in the image (i.e., in sublinear time) [6]. Let a sequence of IFTs be represented as $\langle IFT_{(\mathcal{S}^1)}, IFT_{(\mathcal{S}^2)}, \ldots, IFT_{(\mathcal{S}^n)} \rangle$, where n is the total number of IFT executions on the image. At each execution, the seed set $\mathcal{S}^i$ is modified by adding and/or removing seeds to obtain a new set $\mathcal{S}^{i+1}$. We define a *scene* $\mathcal{G}^i$ as the set of maps $\mathcal{G}^i = \{\texttt{pred}^i, \texttt{root}^i, \texttt{cost}^i\}$, resulting from the *i-th* iteration in a sequence of IFTs. DIFT allows to efficiently compute a scene $\mathcal{G}^i$ from the previous scene $\mathcal{G}^{i-1}$, a set $\Delta^+_{\mathcal{S}^i} = \mathcal{S}^i \setminus \mathcal{S}^{i-1}$ of new seeds for addition, and a set $\Delta^-_{\mathcal{S}^i} = \mathcal{S}^{i-1} \setminus \mathcal{S}^i$ of seeds marked for removal, by reusing the part of the previous calculation that remains unchanged.

2.5 Distance Transform

The Distance Transform (DT) is a well-known binary image transformation that assigns to each foreground pixel the distance to the closest background/contour pixel. Formally, given a binary image X the DT at a pixel $p \in X$ is defined as

$$\texttt{edt}(X,p) = \min_{q \in \partial_{\mathcal{A}}(X)} \|q - p\|, \tag{8}$$

where $\|\cdot\|$ denotes the standard Euclidean L_2 norm on $D_f \subset \mathbb{Z}^2$.

DT-based image operators are obtained in the IFT framework by considering the connectivity function

$$\Psi_{\text{Euc}}(\pi(p,q)) = \begin{cases} \|q - p\| & \text{if } p \in \mathcal{S}; \\ +\infty & \text{otherwise.} \end{cases} \tag{9}$$

Its applications involve its use in multi-scale shape skeletonization [8,9] and shape descriptors (e.g., fractal dimensions [5], contour saliences [13], and shape descriptors based on tensor scale [2]).

The distance transform computation in a differential mode requires an updated version of the DIFT algorithm as proposed in [4]. In this work, we adopt this algorithm with some extra modifications, since we also have here the expansion of the graph $G_{X,\mathcal{A}}$ and not just the modification of the set of seeds from $\mathcal{S}^{i-1}$ to $\mathcal{S}^i$, as explained in Sect. 3, given that X grows as we move towards

the root of the component tree. Therefore, we have a different X_i for each execution of DIFT with $X_{i-1} \subset X_i$. For each new node of the expanded graph in $X_i \setminus X_{i-1}$, its neighbours in X_{i-1} that already have computed paths, rooted in seeds not marked for removal, must also be inserted into the priority queue used by DIFT to allow their future path extensions.

3 Proposed Method

3.1 Proposed Attribute

We define the *maximum distance transform value* (or maximum distance for short) $\texttt{maxDist} : V(T_f) \to \mathbb{R}$ as an attribute of component trees for a node $N \in V(T_f)$ as

$$\texttt{maxDist}(N) = \max_{p \in N} \texttt{edt}(N, p). \tag{10}$$

Since a node N can never lose pixels when we move to its ancestors, the `edt` of the pixels never decreases, making the maximum distance an increasing attribute (see Sect. 2.2). Consequently, it can be used in an extinction value filter (see Sect. 4.2). It describes the thickness of the node in such a way that a thin object independent of its size (area) has a low value of maximum distance. We propose an efficient differential algorithm to compute it.

3.2 Differential Algorithm

Our proposed algorithm processes the max-tree from its leaves to the root. For each level-set, we collect the new seeds by finding new contour pixels and the removed seeds by finding the contour pixels from the previously processed level-set that are not contour pixels on the level-set which is currently being processed. Then, we use DIFT for distance transform computation using the seeds from the previously computed level-set (maintained seeds), the found new seeds (inserted seeds), and removing the seeds (removed seeds) that are not contour pixels in the level-set being processed. The maximum distance transform value for each DIFT root is mapped to a contour pixel of the node. Finally, we scan the contour pixels of the node (computed incrementally) to find the maximum distance transform value mapped to the DIFT root in the previous step.

We summarise these steps in Algorithm 1. In the algorithm, we denote by $\mathbb{Z}^+$ the set of positive integers, $\mathbb{R}^+$ the set of positive real numbers, and $D_f^* = D_f \cup \{\textbf{NIL}\} : \textbf{NIL} \notin D_f$, where **NIL** denotes an invalid pixel.

Algorithm 1 starts by initialising the `maxDist` map, the DIFT variables, and the incremental contour variables in lines 2–7. In line 8, we create a map `levelToNodes` that maps to λ all nodes associated with grey level λ such that we can quickly access all nodes associated with λ. Then, we loop over all levels λ of the input image in lines 9–35 and skip processing levels that are not in the image using the **if** statement in line 10. Then, we loop over all nodes N associated with λ in lines 11–32. For each node N, we (i) remove the background neighbours by scanning the contour of the children and checking if the contour pixel is a

Algorithm 1: Differential algorithm

Input: A greyscale image $f : D_f \to \mathbb{K}_f$ and its max-tree $\hat{T}_f = (\hat{V}(T_f), \hat{E}(T_f))$
Output: A map $\texttt{maxDist} : V(T_f) \to \mathbb{R}^+$

Function computeMaximumDistanceDifferential *(f, $\hat{T}_f$)*
  Let $\texttt{maxDist} : V(T_f) \to \mathbb{R}^+$ with $\texttt{maxDist}[N] = 0, \forall N \in V(T_f)$;
  Let $\texttt{bin} = \emptyset$ and $\texttt{cost} : D_f \to \mathbb{R}^+$ be the cost image;
  Let $\texttt{pred} : D_f \to D_f^*$ with $\texttt{pred}[p] = \textbf{NIL}, \forall p \in D_f$;
  Let $\texttt{root} : D_f \to D_f$ with $\texttt{root}[p] = p, \forall p \in D_f$ and Q be a cost queue;
  Let $\texttt{contours} : V(T_f) \to \mathcal{P}(D_f)$ with $\texttt{contours}[N] = \emptyset, \forall N \in V(T_f)$;
  Let $\texttt{ncount} : D_f \to \mathbb{Z}^+$ with $\texttt{ncount}[p] = 0, \forall p \in D_f$;
  Let $\texttt{levelToNodes} : \mathbb{K}_f \to V(T_f)$ with $\texttt{levelToNodes}[\lambda] = \{N \in V(T_f) : \texttt{level}(T_f, N) = \lambda\}, \forall \lambda \in \mathbb{K}_f$;
  foreach $\lambda \in \max(\mathbb{K}_f)$ *down to* $\min(\mathbb{K}_f)$ **do**
    if $\texttt{levelToNodes}[\lambda] = \emptyset$ **then continue**;
    foreach $N \in \texttt{levelToNodes}[\lambda]$ **do**
      Let $\texttt{toRemove} = \emptyset$ and $\texttt{Ncontour} = \emptyset$ be two sets;
      foreach $C \in \texttt{children}(T_f, N)$ **do**
        foreach $p \in \texttt{contours}[C]$ **do**
          foreach $q \in \mathcal{N}_4(p)$ **do**
            if $q \in D_f$ *and* $f(q) = \texttt{level}(T_f, N)$ **then**
              $\texttt{ncount}[p]$--;
          if $\texttt{ncount}[p] = 0$ **then** $\texttt{toRemove} \leftarrow \texttt{toRemove} \cup \{p\}$;
          else $\texttt{Ncontour} \leftarrow \texttt{Ncontour} \cup \{p\}$;
      if $\texttt{toRemove} \neq \emptyset$ **then** $\texttt{treeRemoval}(\texttt{toRemove}, \texttt{bin}, Q, \texttt{root}, \texttt{pred}, \texttt{cost})$;
      foreach $p \in \hat{N}$ **do**
        $\texttt{bin} \leftarrow \texttt{bin} \cup \{p\}$;
        foreach $q \in \mathcal{N}_4(p)$ **do**
          if $q \notin D_f$ *or* $f(p) > f(q)$ **then** $\texttt{ncount}[p]$++;
        if $\texttt{ncount}[p] > 0$ **then**
          $\texttt{Ncontour} \leftarrow \texttt{Ncontour} \cup \{p\}$;
          $\texttt{root}[p] \leftarrow p, \texttt{pred}[p] \leftarrow \textbf{NIL}, \texttt{cost}[p] \leftarrow 0$;
          $\texttt{insert}(Q, \texttt{cost}, p)$;
        else
          $\texttt{cost}[p] \leftarrow +\infty$;
          foreach $q \in \mathcal{N}_4(p) : q \in \texttt{bin}$ **do**
            if $q \notin Q$ *and* $\texttt{cost}[q] \neq +\infty$ **then** $\texttt{insert}(Q, \texttt{cost}, q)$;
    $\texttt{Bedt} \leftarrow \texttt{EDTDiff}(Q, \mathcal{A}_8, \texttt{bin}, \texttt{root}, \texttt{pred}, \texttt{cost})$
    foreach $N \in \texttt{levelToNodes}[\lambda]$ **do**
      $\texttt{maxDist}[N] \leftarrow \max_{p \in \texttt{contours}[N]} \texttt{Bedt}[p]$
  return maxDist;

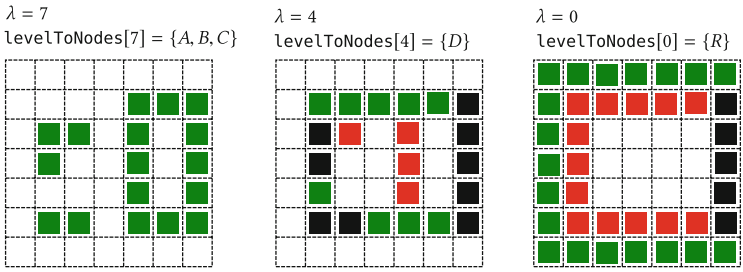


Fig. 3. MIR sets for each level-set of the image in Fig. 1. The pixels in black represent the seeds which are maintained from the previous computed λ, the green pixels represent the inserted seeds at the current level λ, and the red pixels represent the seeds removed from the previous processed λ. (Color figure online)

neighbour of a CNP of N (lines 15–17), (*ii*) if the analysed contour pixel has no background neighbour anymore it is included in the sets of seeds to be removed, otherwise, it is a maintained seed and is included in the contour of N (lines 18 and 19). In line 20, we remove the collected seeds by calling `treeRemoval` (function as described in [4,6]). Next, we scan the CNPs of N (lines 21–32). For each CNP p, we (*i*) include it in the current level-set (line 22) such that when finishing the loop of lines 11–32, we have `bin` $= [f \geq \lambda]$, (*ii*) we count the background neighbours of p (line 24), (*iii*) if p has at least one background neighbour, we include it in the contour of N and update DIFT variables to include it as a new seed (lines 25–28), otherwise we set the variables of the DIFT to make it a non-seed pixel which needs to be processed (lines 30–32). In line 33, all nodes associated with λ have been processed, the DIFT variables are set and we can run the DIFT to compute the distance transform of the level-set $[f \geq \lambda]$. Then, we call `EDTDiff` in line 33 which computes the distance transform using a DIFT and returns an image `Bedt` containing the maximum distance transform valued mapped to a DIFT's seed (contour pixel) of the node. Finally, we extract the maximum DT value from this boundary image `Bedt` for all nodes associated with λ by scanning their contour pixels in lines 34–35.

The main idea of Algorithm 1 is to incrementally keep the sets of maintained, inserted, and removed seeds (MIR sets for short). It keeps these sets implicitly in lines 18 (removed seeds), 19 (maintained seeds), and 26 (inserted seeds). Using MIR sets we can quickly set up a DIFT for distance transform computation and find its maximum value for each CC. Figure 3 depicts the MIR sets for each level-set of the image in Fig. 1.

4 Experimental Results

4.1 Run Time Analysis

We have performed experiments to compare Algorithm 1 to a non-differential approach. In the non-differential approach, we decompose the input image f

into all its level-sets, and run an IFT (non-differential) that computes EDT for each one of them. Then, we reconstruct each node associated with the threshold value and compute the maximum value of the EDT. We implemented both approaches in single-thread C++14 programs, and ran the experiments on the dataset available at [3]. This dataset contains 18 greyscale images varying on content and with dimensions varying from 256×256 to 8218×2700. We ran the experiments on a laptop computer with Ubuntu 19.10, 16 GiB of RAM, and an Intel®Core^{TM}i7-8750H CPU (2.20 Ghz $\times$12) processor. The experiments were run 10 times, alternating between starting running the non-differential and differential approach. Then, we took the average run-time of the 10 runs for each image. The details of the experiments, including source code, scripts, and results, are available on GitHub [16]. In summary, our differential approach was on average 2.01 times faster than the non-differential approach. In addition, the differential approach was at least 1.04 times faster and was maximally 2.87 times faster than the non-differential approach.

4.2 Extinction Value Filters

If we have an increasing attribute and consider a greyscale image as a topographic surface in which the ground is at level zero and the grey-level of each pixel denotes its altitude, we can associate to the peaks the lowest attribute value such that peak will be extinguished. These attribute values are called *extinction values* and can be associated with leaves of a component tree [15]. Then, we can apply a connected filter to the image by sorting the leaves by their extinction value, selecting a desired number L of leaves, and pruning the branches associated with L lowest extinction values. Height, area, volume and many other increasing attributes are applied in different applications. In particular, moment of inertia is an increasing attribute commonly used in applications that rely on object thickness [20]. Similarly, our proposed attribute is increasing and describes the

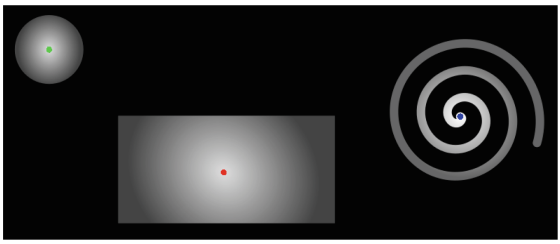
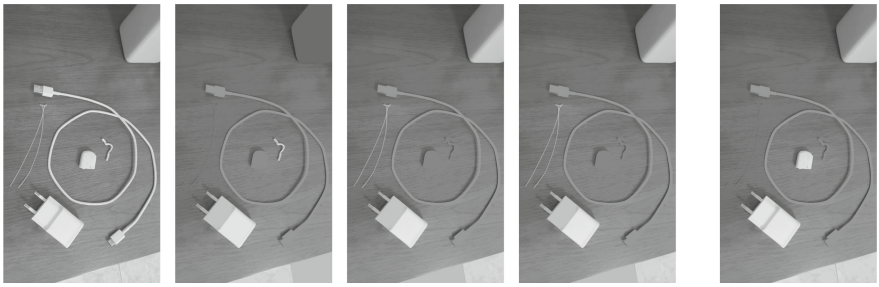


Fig. 4. Synthetic figure for attribute illustrations. The max-tree of the image contains three leaves highlighted in red, green, and blue. Their respective moment of inertia extinction values are $+\infty$, 9.51×10^6, and 9.59×10^7, and maximum distance extinction values are $+\infty$, 2350, and 53. Unlike the moment of inertia, the maximum distance extinction value of the spiral is lower than that of the circle. (Color figure online)

thickness of the object. However, our proposed attribute can better describe the thickness of elongated objects as shown in Fig. 4.



input height ($L = 3$) area ($L = 4$) inertia ($L = 4$) max EDT ($L = 5$)

Fig. 5. Extinction value filter for different attributes. Each image corresponds to the input image after an extinction value filter applied to keep L leaves for each attribute. The number L was chosen as the highest L-value for which the cable is filtered out.

We demonstrate an extinction value filter to remove a cable on a desk with different objects in Fig. 5. The figure shows that our attribute correctly filters thin objects out before filtering out other thick objects. In this example, only our proposed algorithm kept the semi-rounded eraser.

5 Conclusion

The distance transform of a binary image is an important operator used in many applications. In particular, it can be used to describe the thickness of connected components. In this work, we have proposed using the maximum distance transform value as an attribute of component trees. To do so, we have introduced a novel algorithm that adapts the incremental contour computation and the Differential Image Foresting Transform to quickly compute this attribute. We experimentally show that our algorithm is on average twice as fast as the non-differential approach. We have shown that this attribute is increasing and demonstrated its usage in an extinction value filter to remove thin objects of a greyscale image, also comparing it to other increasing attributes.

Acknowledgment. Dennis J da Silva, Ronaldo F. Hashimoto, Paulo A.V. Miranda and Wonder A.L. Alves acknowledge CNPq - Conselho Nacional de Desenvolvimento Científico e Tecnológico (Proc. 141422/2018-1, Proc. 428720/2018-7, Proc. 407242/2021-0, and Proc. 313087/2021-0) 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.

References

1. Alves, W.A.L., Gobber, C.F., Araújo, S.A., Hashimoto, R.F.: Segmentation of retinal blood vessels based on ultimate elongation opening. In: Campilho, A., Karray, F. (eds.) ICIAR 2016. LNCS, vol. 9730, pp. 727–733. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-41501-7_81
2. Andalo, F.A., Miranda, P.A.V., da S. Torres, R., Falcão, A.X.: A new shape descriptor based on tensor scale. In: International Symposium on Mathematical Morphology and Its Application to Signal and Image Processing, pp. 141–152 (2007)
3. Carlinet, E., Géraud, T.: A comparison of many max-tree computation algorithms. In: Hendriks, C.L.L., Borgefors, G., Strand, R. (eds.) ISMM 2013. LNCS, vol. 7883, pp. 73–85. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38294-9_7
4. Condori, M.A., Cappabianco, F.A., Falcão, A.X., Miranda, P.A.: An extension of the differential image foresting transform and its application to superpixel generation. J. Vis. Commun. Image Representation **71**, 102748 (2020)
5. da S. Torres, R., Falcão, A., da F. Costa, L.: A graph-based approach for multiscale shape analysis. Pattern Recogn. **37**(6), 1163–1174 (2004)
6. Falcão, A.X., Bergo, F.P.: Interactive volume segmentation with differential image foresting transforms. IEEE Trans. Med. Imaging **23**(9), 1100–1108 (2004)
7. Falcão, A.X., Stolfi, J., de Alencar Lotufo, R.: The image foresting transform: theory, algorithms, and applications. IEEE Trans. Pattern Anal. Mach. Intell. **26**(1), 19–29 (2004)
8. Falcão, A., da F. Costa, L., da Cunha, B.: Multiscale skeletons by image foresting transform and its application to neuromorphometry. Pattern Recogn. **35**(7), 1571–1582 (2002)
9. Falcão, A., Feng, C., Kustra, J., Telea, A.: Chapter 2 - multiscale 2D medial axes and 3D surface skeletons by the image foresting transform. In: Saha, P.K., Borgefors, G., Baja, G.S.D. (eds.) Skeletonization, pp. 43–70. Academic Press (2017)
10. Gonzalez, R.C., Woods, R.E.: Digital Image Processing, 4th edn. Pearson, London (2018)
11. Najman, L., Couprie, M.: Building the component tree in quasi-linear time. IEEE Trans. Image Process. **15**(11), 3531–3539 (2006)
12. Neumann, L., Matas, J.: Real-time scene text localization and recognition. In: 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 3538–3545. IEEE (2012)
13. da S. Torres, R., Falcão, A.: Contour salience descriptors for effective image retrieval and analysis. Image Vision Comput. **25**(1), 3–13 (2007). SIBGRAPI
14. Salembier, P., Wilkinson, M.H.: Connected operators. IEEE Signal Process. Mag. **26**(6), 136–157 (2009)
15. Silva, A.G., Lotufo, R.d.A.: New extinction values from efficient construction and analysis of extended attribute component tree. In: SIBGRAPI, pp. 204–211 (2008)
16. Silva, D.J., Miranda, P.A.V., Alves, W.A.L., Hashimoto, R.F., Kosinka, J., Roerdink, J.B.T.M.: Differential maximum Euclidean distance transform value computation in component trees - experiments analysis webpage and source code (2023). https://github.com/dennisjosesilva/max_dist_diff. Accessed 10 Oct 2023
17. Silva, D.J., Alves, W.A., Hashimoto, R.F.: Incremental bit-quads count in component trees: theory, algorithms, and optimization. Pattern Recogn. Lett. **129**, 33–40 (2020)

18. Telea, A.C.: Data Visualization: Principles and Practice, 2 edn., pp. 363–379. CRC Press (2014). Chap. Image visualization
19. Wang, J., Silva, D.J., Kosinka, J., Telea, A., Hashimoto, R.F., Roerdink, J.B.T.M.: Interactive image manipulation using morphological trees and spline-based skeletons. Comput. Graph. **108**, 61–73 (2022)
20. Wilkinson, M.H.F., Roerdink, J.B.T.M.: Fast morphological attribute operations using Tarjan's union-find algorithm. In: Goutsias, J., Vincent, L., Bloomberg, D.S. (eds.) Mathematical Morphology and its Applications to Image and Signal Processing. Computational Imaging and Vision, vol. 18, pp. 311–320. Springer, Boston (2002). https://doi.org/10.1007/0-306-47025-X_34