Journal of Mathematical Imaging and Vision (2025) 67:31
https://doi.org/10.1007/s10851-025-01248-8

Efficient Maximum Euclidean Distance Transform Computation in Component Trees Using the Differential Image Foresting Transform

Dennis J. Silva[1,2] · Paulo A. V. Miranda[1] · Wonder A. L. Alves[3] · Ronaldo F. Hashimoto[1] · Jiří Kosinka[2] · Jos B. T. M. Roerdink[2]

Received: 15 September 2024 / Accepted: 7 April 2025

Abstract
The distance transform is a crucial technique in binary image processing, assigning the distance to the nearest contour to each foreground pixel. In this extended version of our previous work, we enhance our method for computing the maximum distance transform (DT) value, now utilizing the optimized differential image foresting transform (DIFT) and improved contour extraction processes. These advancements enable more efficient computation of the maximum DT value across all connected components of a grayscale image, significantly reducing computational time by intelligently reusing DIFT trees rooted at contour points (DIFT seeds). Our optimized algorithm now achieves processing speeds that are twice as fast as our previous differential method. The proposed attribute, maximum distance, which measures the thickness of objects within the image, has proven pivotal in different image processing approaches. We showcase this through detailed illustrations of attribute opening, extinction value filters, watershed, and ultimate attribute openings.

Keywords Component trees · Distance transform · Image foresting transform · Increasing attribute · Thickness attribute

1 Introduction

Component trees are powerful full image representations used in different image processing and analyzing tasks, including designing connected operators [1], text location [2], eye vessel segmentation [3], interactive image manipulation [4], 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 a common pipeline of image processing using component trees is selecting a subset of nodes. This step depends on the image processing task, but the selection usually depends on describing the nodes by their structural (considering the node relationship inside the tree), texture, and geometry information. The information that describes the nodes is called attributes, and computing them quickly is an important procedure in the component tree image processing framework. Thus, many efficient methods for computing attributes in component trees have been proposed. Silva and collaborators [5] proposed a set of patterns based on [6] which are capable of incrementally counting bit-quads [7] in component trees. Using the bit-quad count, we can quickly compute the area, perimeter, Euler number, and number of holes of the nodes of component trees. Najman and Couprie [8] compute the area and height of the nodes during the construction of the component tree. Using the computed area, they compute the volume of the tree performing a tree traversal. Silva

✉ Dennis J. Silva
dennis@ime.usp.br; d.j.da.silva@rug.nl

Paulo A. V. Miranda
pmiranda@ime.usp.br

Wonder A. L. Alves
wonder@uni9.pro.br

Ronaldo F. Hashimoto
ronaldo@ime.usp.br

Jiří Kosinka
j.kosinka@rug.nl

Jos B. T. M. Roerdink
j.b.t.m.roerdink@rug.nl

[1] Institute of Mathematics and Statistics, University of São Paulo, R. do Matão, 1010, São Paulo, SP 05508-090, Brazil

[2] Bernoulli Institute for Mathematics, Computer Science and Artificial Intelligence, University of Groningen, Nijenborgh 9, 9747 AG Groningen, The Netherlands

[3] Informatics and Knowledge Management Graduate Program, Nove de Julho University, R. Vergueiro, 235/249, São Paulo, SP 01525-000, Brazil

and Lotufo [9] adapted the component tree building algorithm by flooding [10] by computing auxiliary values while scanning the pixels of the image. Then, they compute topological height, number of descendants, width, and height of the bounding box when the child–parent connection is established in the tree building by combining the attribute of the child with the auxiliary values computed at the parent.

The distance transform is an important binary image transformation widely used in image processing [11] that assigns to each foreground pixel the distance to the closest contour pixel. In particular, in our previous work [12] we noted 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. We also proposed an algorithm for quickly computing this attribute by combining an incremental contour approach and differential image foresting transform (DIFT). While dynamic shortest path problems in the Many-Sources Many-Destinations have been extensively studied in networking, particularly in scenarios where resources become unavailable [13–16], our approach builds upon the DIFT framework to efficiently handle evolving structures. The method exploits the connected component subset relationship encoding in component trees to incrementally compute sets of removed and inserted seeds (contour pixels) required by the DIFT approach to compute the distance transform. By doing so, we experimentally showed that the approach is twice faster than a non-differential node-reconstruction approach. Since the maximum distance transform value is an increasing attribute, we implemented an extinction value filter to filter out thin objects in grayscale images. This paper extends [12] by including the following contributions:

- *Execution time optimization*: We propose an optimization on the incremental contour computation using grayscale erosion and on the DIFT by including an adaptive adjacency relation.
- *IFT and DIFT description expansion*: We expand the discussion on the IFT and DIFT by describing the changes required in the DIFT for performing the differential approach in the component tree in detail.
- *Detailed execution time experiments on a robust dataset*: We adopted a robust full-HD photograph dataset for measuring the execution time of four different implementations including two non-differential and two differential approaches.
- *New applications of the maximum distance attribute*: The proposed attribute was applied in four different ways: two attribute filtering applications, including attribute opening to enhance larger structures by removing thin ones and extinction value filtering to simplify images based on thin structures; and two object segmentation applications, using watershed segmentation with markers derived from maximum distance attribute to isolate thin structures and ultimate attribute opening to detect high-contrast thin objects.

The remainder of the paper is organized as follows. We recall some useful definitions and notations related to component trees, distance transform, IFT, and DIFT in Sect. 2. We describe the proposed attribute and the optimized differential algorithm in Sect. 3. In Sect. 4, we report our execution time experiments and present different applications of the proposed attribute. We conclude the paper in Sect. 5.

2 Background

In this section, we review key definitions and properties that are essential for this work.

2.1 Images

A *grayscale 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 $\mathbb{K}_f = \{0, 1, \dots, K_f - 1\} \subset \mathbb{Z}$ is the gray-level 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 grayscale image by selecting the pixels whose gray 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 $\mathbb{N}_4 = \{(-1, 0), (0, -1), (1, 0), (0, 1)\}$ and $\mathbb{N}_8 = \mathbb{N}_4 \cup \{(-1, -1), (1, -1), (1, 1), (-1, 1)\}$ as the 4- and 8-*connected adjacency relation*. These adjacency relations generate the 4− or 8-*connected neighborhood* of a pixel p:

$$\mathcal{N}_4(p) = \{p + q : q \in \mathbb{N}_4\} \text{ and } \mathcal{N}_8(p) = \{p + q : q \in \mathbb{N}_8\}. \tag{2}$$

We denote an arbitrary neighborhood of pixel p by $\mathcal{N}(p)$. If $q \in \mathcal{N}(p)$, we say q is a *neighbor* of (or adjacent to) p. In this work, we assume that the neighborhood is symmetric. Specifically, if $q \in \mathcal{N}(p)$, then $p \in \mathcal{N}(q)$ as well. Given a set of pixels $X \subseteq \mathbb{Z}^2$, we denote the set of pairs of adjacent pixels by

$$\mathcal{A}(X) = \{(p, q) \in X^2 : q \in \mathcal{N}(p)\}. \tag{3}$$

Given a grayscale 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, the adjacency relation defines its edges, and the weights of the vertices are the pixel gray 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

$$\begin{aligned} \mathcal{U}(f, \mathcal{A}) &= \{C \in CC(G_{[f \geq \lambda],\mathcal{A}}) : \lambda \in \mathbb{Z}\}, \\ \mathcal{L}(f, \mathcal{A}) &= \{C \in CC(G_{[f \leq \lambda],\mathcal{A}}) : \lambda \in \mathbb{Z}\}. \end{aligned} \tag{4}$$

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:

$$\begin{aligned} \texttt{parent}(N, T) = P &\Leftrightarrow (N, P) \in E(T), \\ \texttt{children}(N, T) &= \{C \in V(T) : (C, N) \in E(T)\}, \\ \texttt{rootTree}(T) = N &\Leftrightarrow \nexists P \in V(T) : P = \texttt{parent}(N, T). \end{aligned} \tag{5}$$

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

$$\texttt{between}(N, P, T_f) = \{P' \in V(T_f) : N \subset P' \subset P\}. \tag{6}$$

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

$$\texttt{level}(N, T_f) = \begin{cases} \inf_{p \in N} f(p), & \text{if } V(T_f) = \mathcal{U}(f, \mathcal{A}), \\ \sup_{p \in N} f(p), & \text{if } V(T_f) = \mathcal{L}(f, \mathcal{A}). \end{cases} \tag{7}$$

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 $\hat{N} = N \setminus \bigcup_{C \in \texttt{children}(T_f, N)}$. 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* $\hat{T}_f = (\hat{V}(T_f), \hat{E}(T_f))$:

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

The compact representations of component trees constructed from upper and lower level sets are known as the *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 gray level. Other common examples are area, volume, and perimeter [17]. Attributes that increase as we move from the leaves to the root are called *increasing*. Formally, an attribute `attr` is increasing if $A, B \in V(T_f) : A \subset B$ implies $\texttt{attr}(A) \leq \texttt{attr}(B)$; an example is area. Otherwise, the attribute is called *non-increasing*, such as the perimeter. An *incremental algorithm* computes a node's attribute based on its own CNPs and the attribute values of its children [5]. An attribute is considered *incremental* if there exists an incremental algorithm that can compute it.

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 [18]:

$$\partial_{\mathcal{A}}(X) = \{p \in X : \exists q \in \mathcal{N}(p), q \in \mathbb{Z}^2 \setminus X\}. \tag{8}$$

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

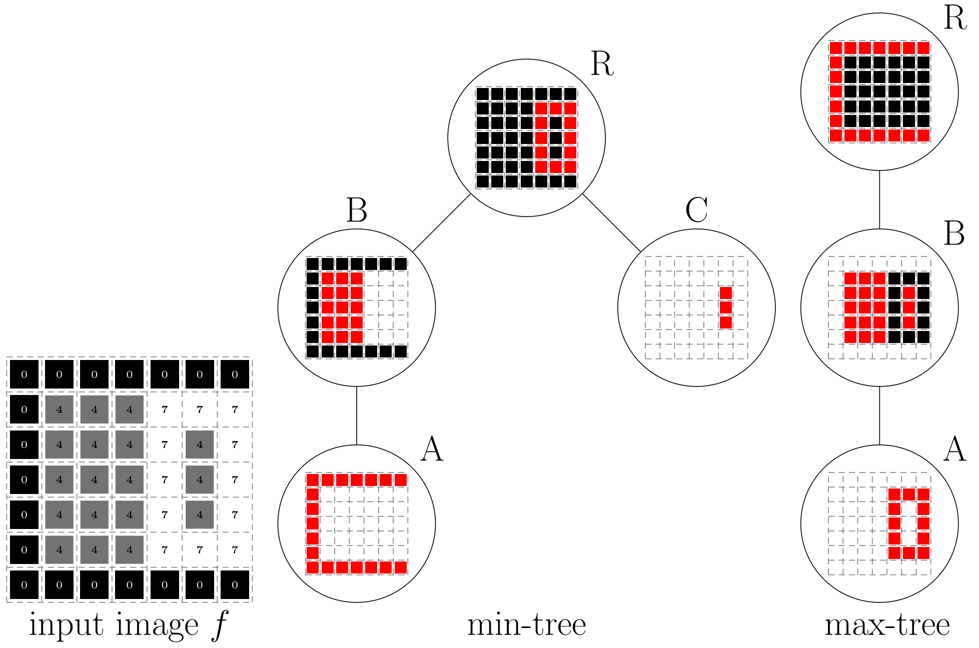


Fig. 1 Max-tree and min-tree of the input image f. The red pixels are the CNPs of the nodes, the black pixels are pixels of the node that are stored in a descendant node, and the white pixels are the background pixels.

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 Image Foresting Transform

The *image foresting transform* (IFT) algorithm [19] 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. 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 $\texttt{root} : D_f \rightarrow D_f$ 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 $\Pi(G_{X,\mathcal{A}}) = \bigcup_{q \in X} \Pi_q(G_{X,\mathcal{A}})$ denote the set of all possible paths in $G_{X,\mathcal{A}}$. A *connectivity function* $\Psi : \Pi(G_{X,\mathcal{A}}) \rightarrow \mathbb{R}^+$ 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. The *image foresting transform* [19] takes an image graph $G_{X,\mathcal{A}}$ and a connectivity function Ψ and assigns one optimal path $\pi^*(s, p)$ from $s \in \mathcal{S}$ to every pixel $p \in D_f$. However, Ψ must satisfy certain conditions, as described in [20]; otherwise, the paths may not be optimal. During the IFT calculation, a cost map $\texttt{cost} : D_f \rightarrow \mathbb{R}^+$ is used to store the costs of the computed paths $\pi^*(s, p)$, such that $\texttt{cost}(p) = \Psi(\pi^*(s, p))$ for all $p \in X$.

For a path $\pi(r_0, r_n) = (r_0, r_1, \ldots, r_n)$ in X, a classic example of a connectivity function is given by the geodesic cost function Ψ_{Geo}:

$$\Psi_{\text{Geo}}(\pi(r_0, r_n)) = \begin{cases} \sum_{i=0}^{n-1} \|r_i - r_{i+1}\| & \text{if } r_0 \in \mathcal{S}; \\ +\infty & \text{otherwise}, \end{cases} \tag{9}$$

where $\mathcal{S}$ is a seed set and $\|\cdot\|$ denotes the standard Euclidean L_2 norm on $D_f \subset \mathbb{Z}^2$. An example of IFT computation with Ψ_{Geo} and $X = D_f$ is presented in Fig. 3a–b, including an explanation of the $\texttt{pred}$, $\texttt{root}$, and $\texttt{cost}$ maps.

2.5 Differential Image Foresting Transform

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 $\texttt{pred}$ and other maps in a time proportional to the size of the modified regions in the image (i.e., in sublinear time) [21]. 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

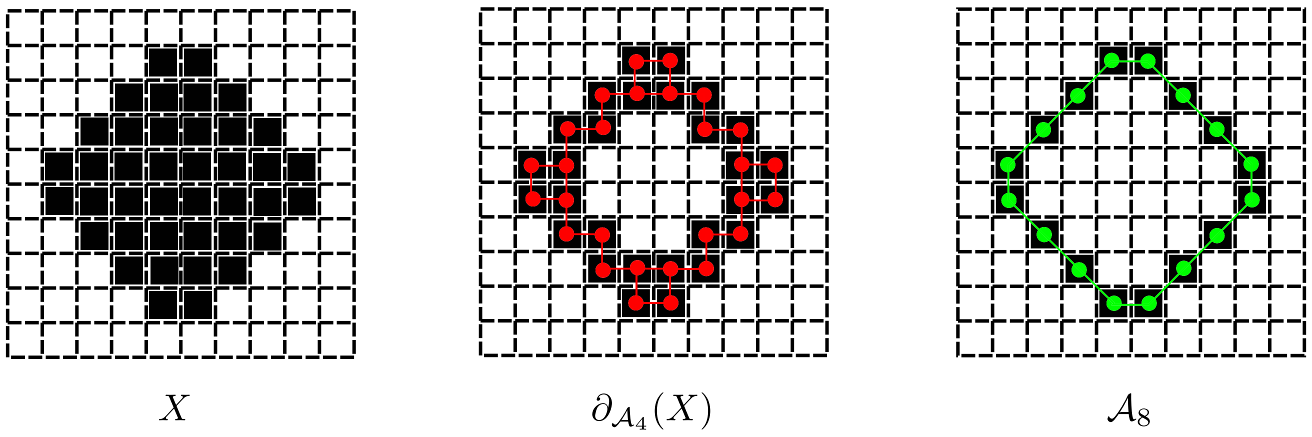


Fig. 2 Contour definition and the impact of the adopted adjacency relation. X is a binary image, $\partial_{\mathcal{A}_4}(X)$ is the contour of X extracted using $\mathcal{A}_4$, and $\partial_{\mathcal{A}_8}(X)$ is the contour of X extracted using $\mathcal{A}_8$. The graph in red (middle) shows that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

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.

An example of differential path updating by DIFT is shown in Fig. 3c–d for the connectivity function Ψ_{Geo}, where the seed e is removed from $\mathcal{S}^1 = \{e, f, q, x\}$, i.e., $\Delta^-_{\mathcal{S}^2} = \{e\}$ and $\mathcal{S}^2 = \{f, q, x\}$. In Fig. 3c, the nodes of the path tree that was rooted at e in Fig. 3b were reset to infinite cost, thus becoming available for a new dispute from the frontier nodes (nodes with purple background) of their non-removed neighboring trees, by analyzing the extension of previously computed paths of neighboring trees instead of starting from scratch. In Fig. 3d, the result of DIFT is presented, with only the nodes in the removed region being updated.

2.6 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

$$\texttt{edt}(X, p) = \min_{q \in \partial_{\mathcal{A}}(X)} \|q - p\|, \tag{10}$$

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

$$\Psi_{\text{Euc}}(\pi(p, q)) = \begin{cases} \|q - p\|^2 & \text{if } p \in \mathcal{S}; \\ +\infty & \text{otherwise}, \end{cases} \tag{11}$$

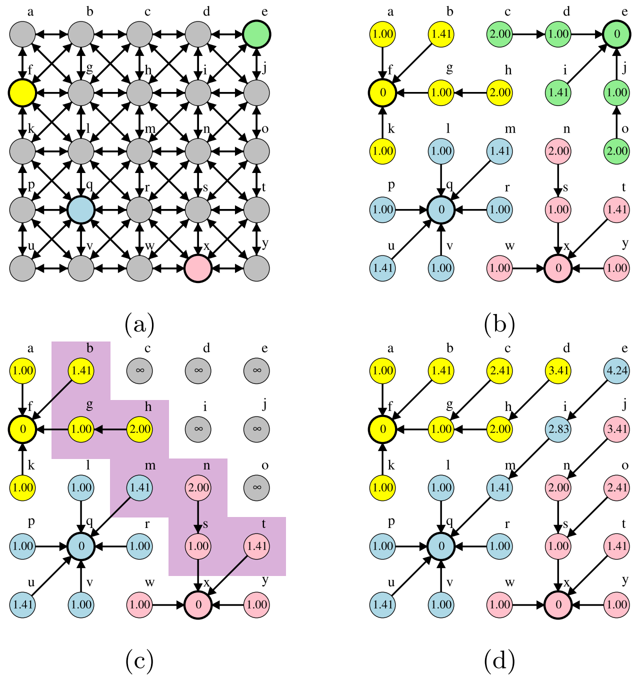


Fig. 3 **a**–**b** Example of IFT computation using Ψ_{Geo} in a graph with $\mathcal{A}_8$. **a** Input graph with marked seeds given by $\{e, f, q, x\}$. **b** The arrows indicate the predecessor of each node in the computed paths, and their costs by Ψ_{Geo} are indicated inside the nodes. For example, in the case of the path $\pi^*(f, h) = (f, g, h)$, we have that $g = \texttt{pred}(h)$, $f = \texttt{pred}(g)$, $\texttt{pred}(f) = \textbf{NIL}$, $f = \texttt{root}(h) = \texttt{root}(g)$ and $\texttt{cost}(h) = \Psi_{\text{Geo}}(\pi^*(f, h)) = 1 + 1 = 2$. **c**–**d** Example of differential path updating by DIFT, where $\Delta^-_{\mathcal{S}^2} = \{e\}$ and $\mathcal{S}^2 = \{f, q, x\}$. **c** The path tree rooted at e is removed and the frontier nodes (purple background) of neighboring trees become active for a new competition. **d** The previously computed paths ending at the frontier nodes are extended, thus generating new paths, such as path $\pi^*(q, e) = (q, m, i, e)$, which is obtained by extending $\pi^*(q, m) = (q, m)$.

where $\mathcal{S}$ is a seed set and squared values are used so that we have integer costs, which allow the use of a more efficient priority queue in IFT by bucket sorting.

For each pixel t, here we aim to compute the minimum Euclidean distance from t to $\mathcal{S}$. However, when $|\mathcal{S}| \geq 3$, this IFT algorithm using $\mathcal{A}_4$ may not output the exact DT, because the regions of the discrete Voronoi diagram may not be 4-connected (Fig. 4). On the other hand, the exact DT for arbitrary seeds can be computed by IFT if the adjacency radius is large enough [19]. In practice, the DT errors by IFT with 8-connected adjacency and Ψ_{Euc} are so small and unusual that they can be neglected for most applications (Fig. 5). Indeed, DT by IFT with $\mathcal{A}_8$ and $\mathcal{S} = \partial_{\mathcal{A}_4}(X)$ has been successfully used in several applications, including its use in multi-scale shape skeletonization [22, 23] and shape descriptors (e.g., fractal dimensions [24], contour saliences [25], and shape descriptors based on tensor scale [26]). An analysis of the upper bound of the error is presented in Appendix A.

3 Proposed Method

In this section, we present our proposed attribute (the maximum distance transform value) and a new method to computed it. It is worth noting that this new method is an optimized version of the one presented in [27]. We also observe that our method takes as input an 8−connected component tree and it considers the contours $\partial_{\mathcal{A}_4}([f \geq \lambda])$, for each $\lambda \in \mathbb{K}_f$, as defined in Equation (8). The output of our method is the maximum distance transform value for each node of the component tree.

3.1 Proposed Attribute

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

$$\mathtt{maxDist}(N) = \max_{p \in N} \mathtt{edt}(N, p). \tag{12}$$

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). Figure 6 provides an illustrative example of the maxDist attribute computed for each node in the component tree shown in Fig. 1. For better clarity, each node displays the EDT values for all pixels in the corresponding CC, making it easier to observe that maxDist is defined as the highest EDT value within each CC.

As an increasing attribute, maxDist can be applied in various morphological operations to analyze and process image structures. By capturing the maximum thickness of each connected component, maxDist allows filtering objects based on their shape rather than just their size (area). This property makes it particularly useful in applications where the preservation of structural characteristics is essential. Later, in Sect. 4.2, we demonstrate its versatility in defining attribute filters, such as attribute openings and extinction value filters, as well as its use in segmentation methods, including marker-based watershed segmentation and ultimate attribute opening.

We propose an efficient differential algorithm to compute maxDist, leveraging the hierarchical structure of the component tree to ensure fast computation.

3.2 Differential Distance Transform by DIFT

The IFT computation in a differential mode using the connectivity function Ψ_{Euc} from Eq. (11) requires an updated version of the DIFT algorithm as proposed in [28]. In this work, we adopt a customized version of this algorithm, taking into account the particularities of the DT problem, and with some extra modifications, aiming for a better integration with the subsequent computation of the new proposed attribute.

Some additional modifications are needed because we are dealing with both the expansion of the graph $G_{X,\mathcal{A}}$ and the changes in the seed set from $\mathcal{S}^{i-1}$ to $\mathcal{S}^i$, since X grows while moving toward the root of the component tree, as explained in Sect. 3.3. Therefore, we have a different X_i for each execution of DIFT with $X_{i-1} \subset X_i$. For each new pixel in the expanded graph $X_i \setminus X_{i-1}$, its neighbors in X_{i-1}, which already have computed paths rooted in seeds not marked for removal, must also be inserted into the priority queue used by DIFT to enable future path extensions. Here, we leave the handling of this issue and the management of the insertion of new seeds into the priority queue Q to the calling function (Algorithm 3 presented in the next section), for better code integration. Therefore, differently from [28], our differential algorithm (Algorithm 1) assumes that the new seeds from $\Delta^+_{\mathcal{S}^i}$ have already been inserted into Q and that the call to remove seeds in $\Delta^-_{\mathcal{S}^i}$ will also be handled externally in Algorithm 3.

Due to the growth of X_i, for increasing values of i, we know that the exact expected value of the DT for each pixel is a non-decreasing function over the iterations, i.e., $\mathtt{edt}(X_i, t) \geq \mathtt{edt}(X_{i-1}, t)$ for all t in X_{i-1}. Therefore, we adopt a simplification in the code so that at each iteration of the DIFT we only consider the calculation of new paths ending in nodes belonging to a set open, composed of new pixels in $X_i \setminus X_{i-1}$ and pixels of removed trees with roots in $\partial_{\mathcal{A}_4}(X_{i-1}) \setminus \partial_{\mathcal{A}_4}(X_i)$. That is, for pixels in $X_i \setminus \mathtt{open}$, we keep the approximated values of the EDT already computed in the previous iteration. The usage of this simplification makes the predecessor test and the call to the

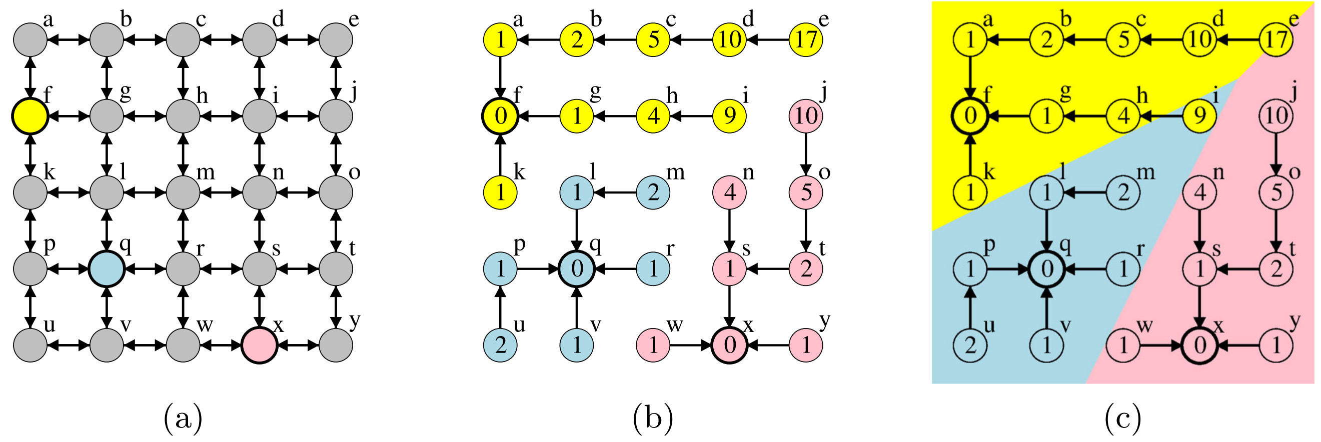


Fig. 4 Example of a non-exact DT calculation by IFT using Ψ_{Euc} in a graph with $\mathcal{A}_4$. **a** Input graph with marked seeds given by $\mathcal{S} = \{f, q, x\}$. **b** The arrows indicate the predecessor of each node in the computed paths, and their costs by Ψ_{Euc} are indicated inside the nodes. For example, in the case of the path $\pi^*(q,u) = (q,p,u)$, we have that $p = \mathtt{pred}(u)$, $q = \mathtt{pred}(p)$, $\mathtt{pred}(q) = \mathbf{NIL}$ and $\mathtt{cost}(u) = \Psi_{\mathrm{Euc}}(\pi^*(q,u)) = 2$. **c** The Voronoi diagram is depicted under the nodes, indicating that node i should belong to the Voronoi cell of seed q in an exact DT. The computed Euclidean distance to node i is $\sqrt{\mathtt{cost}(i)} = \sqrt{9} = 3$, while the exact value would be $\|q - i\| = \sqrt{8} \approx 2.828$, leading to an error of 0.172. Note that this particular illustrated problem would not happen in the case of an 8-neighborhood graph.

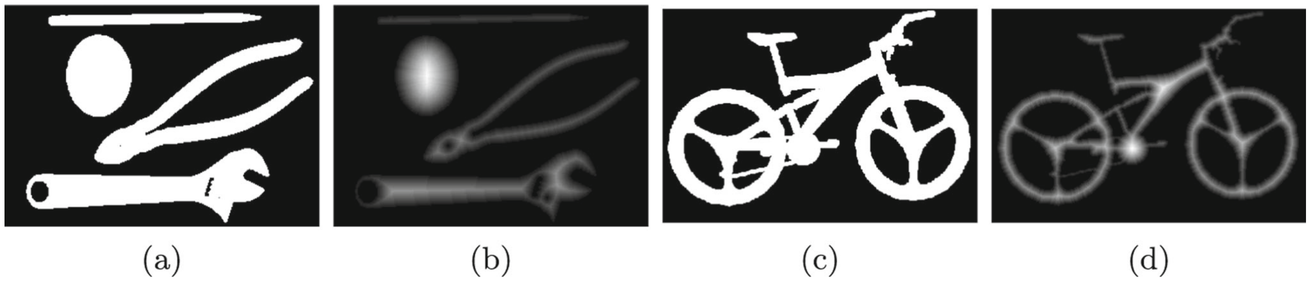


Fig. 5 Examples of DT computation by IFT with 8-connected adjacency. **a** & **c** Two given binary images X_1 and X_2, respectively. **b** & **d** The corresponding DT values computed by IFT for all pixels in X_1 and X_2, respectively. The DT error by IFT at pixel q is measured by $\epsilon(q) = \left|\mathtt{edt}(X,q) - \sqrt{\Psi_{\mathrm{Euc}}(\pi^*(p,q))}\right|$, with $\pi^*(p,q)$ the path computed by IFT in $G_{X,\mathcal{A}_8}$, such that $p \in \mathcal{S} = \partial_{\mathcal{A}_4}(X)$. In the case of X_1, the maximum error made for a pixel was 0.038490, while the mean error in X_1 was only 0.000011, with the errors affecting only 0.037161% of pixels in X_1 (i.e., only 8 pixels out of 21,528 pixels). In the case of X_2, no error was found.

auxiliary function `DIFT-RemoveSubTree` from [28] no longer necessary.

Another simplification adopted consists of not explicitly computing the map of predecessors `pred`, since we are only interested in the values of `cost`. However, for this effect to be possible, the auxiliary function `treeRemoval` (Algorithm 2) needed to be completely rewritten, compared to the version presented in [28], in order to allow the removal of trees based solely on their `cost` and `root` maps.

Another optimization adopted corresponds to the use of an adaptive neighborhood. Since the cost of the connectivity function Ψ_{Euc} depends only on the endpoints of the path, we have that during the propagation of paths, when a pixel s conquers a neighboring pixel t, later, when it is t's turn to visit its neighbors, it does not need to revisit s nor any of s's neighbors, since the cost of the paths passing through t to them will be the same. Therefore, it needs to consider only the region without overlapping of its neighborhood, i.e., $\mathcal{N}_8(t) \setminus (\mathcal{N}_8(s) \cup \{s\})$, as indicated in Fig. 7. For example, for a pixel $x \in \mathcal{N}_8(t) \cap \mathcal{N}_8(s)$, the paths $(r, \ldots, s, x)$ and $(r, \ldots, s, t, x)$ have the same cost $\|x - r\|^2$ according to the connectivity function Ψ_{Euc}. Therefore, the path passing through t does not need to be evaluated.

In fact, in the case of diagonal displacements, as shown in Fig. 7b, the two green neighbors of t in regions without overlap with $\mathcal{N}_8(s)$, can also be avoided. Without loss of generality, for example consider the green pixel at $u = s + (2,0)$, i.e., u is two pixels to the right of s. Since the Voronoi cell of the seed r has a convex shape, in case u is conquered by the extension through t via the path $(r, \ldots, s, t, u)$, it implies that $s + (1,0)$ would also be part of the region conquered by r. Therefore, u would already be reachable from r via

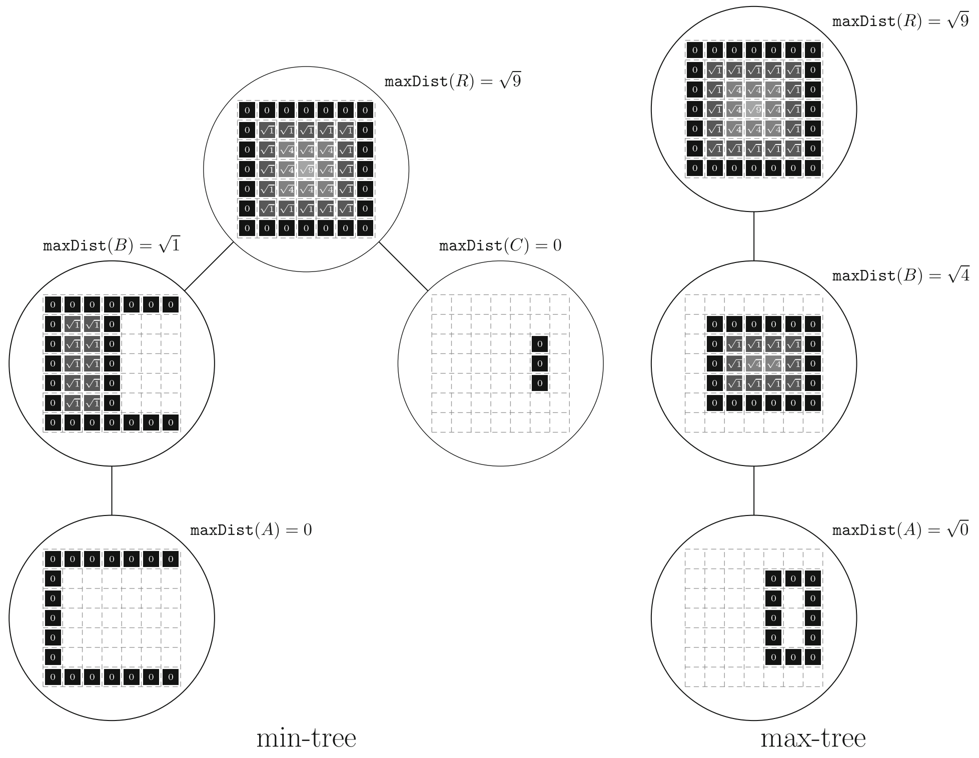

$\texttt{maxDist}(R) = \sqrt{9}$
$\texttt{maxDist}(B) = \sqrt{1}$
$\texttt{maxDist}(C) = 0$
$\texttt{maxDist}(A) = 0$
min-tree
$\texttt{maxDist}(R) = \sqrt{9}$
$\texttt{maxDist}(B) = \sqrt{4}$
$\texttt{maxDist}(A) = \sqrt{0}$
max-tree

Fig. 6 Illustrative example of the maxDist attribute computed for each node in a component tree, using the same tree structure as in Fig. 1 to ensure consistency in visualization. For clarity, each node displays the EDT values for all pixels in the corresponding CC, making it easier to observe that the maxDist attribute corresponds to the highest EDT value within each CC.

$s + (1, 0)$, so its access through t would become redundant in Algorithm 1. However, note that, in the case of accessing neighbors during the tree removal in Algorithm 2, visiting the green pixels is necessary for the correct identification of the frontier nodes belonging to non-removed neighboring trees.

Adaptive neighborhood is implemented considering a family of indexed neighborhoods numbered from 0 to 8, including the complete neighborhood of 8 neighbors at index numbered 0, as well as the four rotations of each of the two patterns shown in Fig. 7. A map $\texttt{adjmap} : D_f \to [0, 8]$ is used to store the index $k_s = \texttt{adjmap}[s]$ of the adjacency that will be used for each pixel s. Initially, the indexes start at zero, which corresponds to the complete neighborhood of 8 neighbors, but as the DIFT paths are computed, this map is updated following the scheme in Fig. 7, so that extensions of paths with more than one pixel in length will now consider only three neighbors.

Given the pair (s, k_s), the neighbors of pixel s are accessed through the set $\mathcal{N}_{adapt}(s, k_s)$ that stores ordered pairs of the form (t, k_t), where t is a neighboring pixel of s and k_t is the index of the neighborhood to be considered for t, if the path to t is selected passing through s. We consider only three neighbors (indicated with a yellow background in Fig. 7) in the case of Algorithm 1. In the case of Algorithm 2, we use an extended adaptive adjacency $\mathcal{N}^*_{adapt}(s, k_s)$ that also considers the green neighbors from Fig. 7b.

Finally, in order to facilitate the computation of the proposed attribute from Eq. (12), during the DIFT computation, we store the maximum squared distance transform value found for each DIFT tree at its root in a map Bedt, i.e., $\texttt{Bedt}(r) = \max_{t \in X_i : \texttt{root}(t) = r} \texttt{cost}(t)$ and $r \in \partial_{\mathcal{A}_4}(X_i)$.

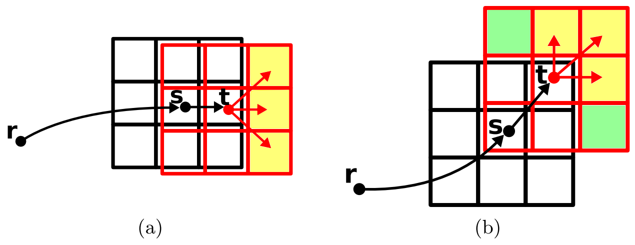


Fig. 7 Illustration of the adaptive neighborhood, which considers only non-overlapping pixels in $\mathcal{N}_8(t) \setminus (\mathcal{N}_8(s) \cup \{s\})$. **a** Case of motion along a single axis. **b** Case of diagonal motion. The green pixels should only be considered in the tree removal algorithm (Algorithm 2).

3.3 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 by removing the seeds (removed seeds) that are not contour pixels in the level set being processed.

Algorithm 1: Algorithm DIFT for EDT

```
Input: Priority queue $Q$, the set open and the maps adjmap,
       Bedt, root, and cost initialized with the result from
       the previous DIFT execution.
Output: The updated data structures $Q$, open, adjmap, Bedt,
        root, and cost passed by reference.
1  Function EDTDiff ($Q$, open, adjmap, Bedt, root, cost)
2      while $Q \neq \emptyset$ do
3          Remove $s$ from $Q$ such that
           cost$[s] = \min_{t \in Q}\{$cost$[t]\}$;
4          open $\leftarrow$ open $\setminus \{s\}$, $r \leftarrow$ root$[s]$;
5          Bedt$[r] \leftarrow \max\{$Bedt$[r]$, cost$[s]\}$;
6          $k_s \leftarrow$ adjmap$[s]$;
7          foreach $(t, k_t) \in \mathcal{N}_{adapt}(s, k_s)$ do
8              $c \leftarrow \|t - r\|$;
9              if $c <$ cost$[t]$ and $t \in$ open then
10                 if $t \in Q$ then remove $t$ from $Q$;
11                 cost$[t] \leftarrow c$, root$[t] \leftarrow r$;
12                 adjmap$[t] \leftarrow k_t$;
13                 Insert $t$ in $Q$;
```

Algorithm 2: Algorithm to remove seeds and their trees

```
Input: The set of seeds to be removed $\Delta_S^-$, priority queue $Q$, the
       sets bin and open, and the maps adjmap, root, and
       cost.
Output: The updated data structures $Q$, open, and cost passed
        by reference.
Auxiliary: Stack $T$ of nodes initially empty.
1  Function treeRemoval ($\Delta_S^-$, $Q$, bin, open, adjmap,
   root, cost)
2      $T \leftarrow \emptyset$;
3      foreach seed $r \in \Delta_S^-$ do
4          open $\leftarrow$ open $\cup \{r\}$, cost$[r] \leftarrow \infty$;
5          Insert $r$ at the top of $T$;
6      while $T \neq \emptyset$ do
7          Remove $s$ from the top of $T$;
8          $k_s \leftarrow$ adjmap$[s]$;
9          foreach $(t, k_t) \in \mathcal{N}^*_{adapt}(s, k_s)$ do
10             if cost[root$[t]$] $= \infty$ then
11                 if $t \notin$ open then
12                     open $\leftarrow$ open $\cup \{t\}$, cost$[t] \leftarrow \infty$;
13                     Insert $t$ at the top of $T$.
14             else if $t \in$ bin and $t \notin Q$ then
15                 Insert $t$ in $Q$;
```

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.

In [27], the contour is extracted by keeping the number of background neighboring pixels each pixel of the nodes currently being processed has. The number of neighboring pixels is kept by using a map which is incremented for a CNP when it finds a neighbor with a lower gray level and the map is decremented at a CNP neighbor if the neighbor has a higher gray level than the CNP. In this paper, we further optimize the contour extraction runtime. First, we define a negative value for points outside the image domain:

$$p \notin D_f \Rightarrow f(p) = -\infty. \tag{13}$$

Then we perform the grayscale erosion [29] of the input image f by $\mathtt{N}_4$, which is defined for $p \in D_f$ as:

$$[\varepsilon_{\mathtt{N}_4}(f)](p) = \min_{q \in \mathcal{N}_4(p)} f(q). \tag{14}$$

Using $\varepsilon_{\mathtt{N}_4}(f)$, we can quickly check if a pixel p of a node N is an $\mathcal{A}_4$-contour pixel by comparing $[\varepsilon_{\mathtt{N}_4}(f)](p)$ with the level associated to the node N as described by Proposition 1.

Proposition 1 *Let $N \in \mathcal{U}(f, \mathcal{A}_8)$ be a node of f's max-tree. Then, $p \in \partial_{\mathcal{A}_4}(N)$ if and only if $[\varepsilon_{\mathtt{N}_4}(f)](p) < \mathtt{level}(T_f, N)$.*

Proof $\Rightarrow$

Suppose by contradiction that $p \in \partial_{\mathcal{A}_4}(N)$ and $[\varepsilon_{\mathtt{N}_4}(f)](p) \geq \mathtt{level}(T_f, N)$. Since $[\varepsilon_{\mathtt{N}_4}(f)](p) \geq \mathtt{level}(T_f, N)$, then (i) every $q \in \mathcal{N}_4(p)$ has gray level greater than or equal to $\mathtt{level}(T_f, N)$. Also, (ii) $p \in \partial_{\mathcal{A}_4}(N) \Rightarrow p \in N$. Thus, (i) and (ii) implies that $q \in N$. But, it is a contraction because $\nexists q \in \mathcal{N}_4(p)$ such that $q \notin N$, which contradicts $p \in \partial_{\mathtt{N}_4}(N)$. $\Leftarrow$

$$\begin{aligned}[\varepsilon_f(\mathtt{N}_4)](p) < \mathtt{level}(T_f, N) &\Rightarrow \exists q \in \mathcal{N}_4(p) : f(q) \\ &\quad < \mathtt{level}(T_f, N) \\ &\Rightarrow q \notin N \\ &\Rightarrow p \in \partial_{\mathtt{N}_4}(N).\end{aligned}$$

□

In this way, we save the time of revisiting the neighbors of the children's contours for counting the background neighbors, which is performed in the original approach.

We summarize our full approach with the erosion optimization in Algorithm 3. In the algorithm, we denote by $\mathbb{Z}^+$ the set of positive integers, $\mathbb{R}^+$ the set of positive real numbers, and $D_f^* = D_f \cup \{\mathbf{NIL}\} : \mathbf{NIL} \notin D_f$, where **NIL** denotes an invalid pixel.

Algorithm 3 starts by initializing the maxDist map and the DIFT variables in lines 2–6. Line 7 initializes the node to contours map with empty contours and the erosion of f by $\mathtt{N}_4$ is computed in line 8. In line 9, we create a map $\mathtt{level}ToNodes$ that maps to λ all nodes associated with gray 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 10–32 and skip processing levels that are not in the

Algorithm 3: Differential algorithm

```
   Input: A grayscale image $f : D_f \to \mathbb{K}_f$ and its component tree
          $T_f = (V(T_f), E(T_f))$
   Output: A map maxDist : $V(T_f) \to \mathbb{R}^+$
 1 Function computeMaximumDistanceDifferential($f$, $T_f$)
 2     Let maxDist : $V(T_f) \to \mathbb{R}^+$ with maxDist$[N] \leftarrow 0, \forall N \in V(T_f)$;
 3     Let bin $\leftarrow \emptyset$, open $\leftarrow \emptyset$ and cost : $D_f \to \mathbb{R}^+$ be the cost image;
 4     Let root : $D_f \to D_f$ with root$[p] \leftarrow p, \forall p \in D_f$ and $Q$ be a cost queue;
 5     Let Bedt : $D_f \to \mathbb{Z}$ with Bedt$[p] \leftarrow 0, \forall p \in D_f$;
 6     Let adjmap : $D_f \to \mathbb{Z}$ with adjmap$[p] \leftarrow 0, \forall p \in D_f$;
 7     Let contours : $V(T_f) \to \mathcal{P}(D_f)$ with contours$[N] \leftarrow \emptyset, \forall N \in V(T_f)$;
 8     Let $f_{\text{eroded}} = \varepsilon_{\mathrm{N}_4}(f)$;
 9     Let levelToNodes : $\mathbb{K}_f \to V(T_f)$ with
       levelToNodes$[\lambda] = \{N \in V(T_f) :$ level$(T_f, N) \leftarrow \lambda\}, \forall \lambda \in \mathbb{K}_f$;
10     foreach $\lambda \in \max(\mathbb{K}_f)$ down to $\min(\mathbb{K}_f)$ do
11         if levelToNodes$[\lambda] = \emptyset$ then continue;
12         foreach $N \in$ levelToNodes$[\lambda]$ do
13             Let $\Delta_S^- \leftarrow \emptyset$ and Ncontour $\leftarrow \emptyset$ be two sets;
14             foreach $C \in$ children$(T_f, N)$ do
15                 foreach $p \in$ contours$[C]$ do
16                     if $f_{eroded}(p) \geq \lambda$ then $\Delta_S^- \leftarrow \Delta_S^- \cup \{p\}$;
17                     else Ncontour $\leftarrow$ Ncontour $\cup \{p\}$;
18             if $\Delta_S^- \neq \emptyset$ then
               treeRemoval($\Delta_S^-$, $Q$, bin, open, adjmap, root, cost);
19             foreach $p \in \hat{N}$ do
20                 bin $\leftarrow$ bin $\cup \{p\}$;
21                 if $f_{eroded}(p) < \lambda$ then
22                     Ncontour $\leftarrow$ Ncontour $\cup \{p\}$;
23                     root$[p] \leftarrow p$, cost$[p] \leftarrow 0$;
24                     insert($Q$, cost, $p$);
25                 else
26                     open $\leftarrow$ open $\cup \{p\}$; cost$[p] \leftarrow +\infty$;
27                     foreach $q \in \mathcal{N}_4(p) : q \in$ bin do
28                         if $q \notin Q$ and cost$[q] \neq +\infty$ then
                               insert($Q$, cost, $q$);
29             contours$[N] \leftarrow$ Ncontour;
30         EDTDiff($Q$, open, adjmap, Bedt, root, cost);
31         foreach $N \in$ levelToNodes$[\lambda]$ do
32             maxDist$[N] \leftarrow \sqrt{\max_{p \in \text{contours}[N]} \text{Bedt}[p]}$;
33     return maxDist;
```

image using the **if** statement in line 11. Then, we loop over all nodes N associated with λ in lines 12–29. For each node N, we scan the contour of the children and apply Proposition 1 to evaluate if they are in the contour of N. The children's contour pixels that are not in the contour of N are stored in the Δ_S^- set (line 16). Line 17 updates the `Ncontour` to include the children's contour pixels, which are also contour pixels in N. In line 18, we remove the collected seeds by calling `treeRemoval` (see Algorithm 2).

Next, we scan the CNPs of N (lines 19–28). For each CNP p, we (i) include it in the current level set (line 20) such that when finishing the loop of lines 12–29, we have `bin` $= [f \geq \lambda]$, (ii) we apply Proposition 1 to check if p is a contour pixel (line 21); if it is, we include it in the contour of N and update DIFT variables to include it as a new seed (lines 23 and 24), otherwise we set the variables of the DIFT to make it a non-seed pixel which needs to be processed (lines 26–28). The loop on the CNPs finishes in line 29; at this point, the contours of the node N are stored in `Ncontour` which is registered in the map `contours`. In line 30, 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` (see Algorithm 1) which computes the distance transform using a DIFT and returns an image `Bedt` containing the maximum distance transform value 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 31–32.

The main idea of Algorithm 3 is to incrementally keep the sets of maintained, inserted, and removed seeds (MIR sets for short). It keeps these sets implicitly in lines 16 (removed seeds), 17 (maintained seeds), and 22–24 (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 8 depicts the MIR sets for each level set of a grayscale image.

3.4 Complexity Analysis

This section discusses the time complexity of Algorithm 3. To simplify the notation, we denote $V(T_f)$ by V when no confusion is likely to arise. To compute the execution time complexity of Algorithm 3, we divide it into steps:

- *(i) Initialization*: The initialization step consists of initializing all associated variables and data structures as performed by lines 2–9. Since $|V| < |D_f|$, the complexity of initializing the image size maps in lines 4, 5, 6, and 8 is

  $$\mathcal{O}(|D_f|).$$

- *(ii) Computing Δ_S^- and initializing* `Ncontour`: The loop in lines 12–17 computes the set of removed seeds Δ_S^- and initializes `Ncontour` with the children contour pixels that remain contour pixels in N. The loop in lines 14–17 scans the contours of the children. If we denote the length of the longest contour between all nodes and the number of children of N by $\delta_{\max}$ and c_N, respectively, then we have $\sum_{N \in V(T_f)} c_N = |V| - 1$, and thus the following

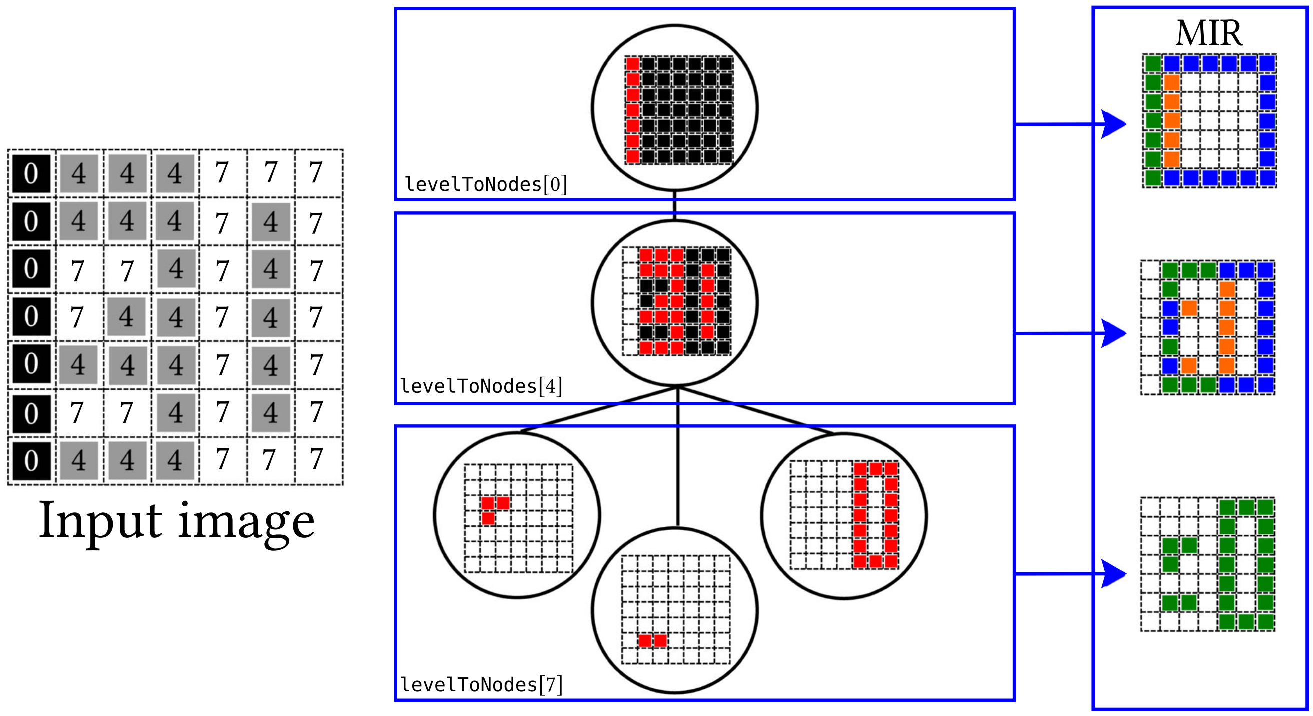


Fig. 8 MIR sets for each level set of the input image. The pixels in blue represent the seeds which are maintained from the previous computed λ, the green pixels represent the inserted seeds at the current level λ, and the orange pixels represent the seeds removed from the previous processed λ.

time complexity

$$\mathcal{O}(|V|\delta_{\max}).$$

- *(iii) DIFT seed removal:* Line 18 performs a call to `treeRemoval` (Algorithm 2) which runs in $\mathcal{O}(\Delta^{-}_{\mathcal{S},\max})$. If we denote the biggest $\Delta^{-}_{\mathcal{S}}$ by $\Delta^{-}_{\mathcal{S},\max}$, then line 18 is upper bounded by

$$\mathcal{O}(|V| \times |\Delta^{-}_{\mathcal{S},\max}|).$$

- *(iv) Computing new seeds*: The time complexity of the loop in lines 19–28 is proportional to the number of times line 20 is performed since it is the only line performed at every iteration. Since the loop is performed once for each node and it iterates over the CNPs of the node, the time complexity of the loop is

$$\mathcal{O}(|D_f|).$$

- *Performing DIFT for EDT computing*: Algorithm 3 calls `EDTDiff` (Algorithm 1) once for each level in $\mathbb{K}_f$. `EDTDiff` runs in sublinear time [21] since it processes the full IFT trees rooted at the removed and inserted seeds. Let U be the biggest set of updated nodes during the differential recomputation of the forest by DIFT, then line 30 runs in

$$\mathcal{O}(|\mathbb{K}_f| \times |U|).$$

- *Finding the maximum distance value*: The maximum distance value of each node is computed by the loop in lines 31 and 32. It scans the contour pixels of each node of the component tree, resulting in the comprehensive complexity:

$$\mathcal{O}(|V|\delta_{\max}).$$

Thus, the full worst-case time complexity of Algorithm 3 is

$$\mathcal{O}\left(|D_f| + (|V| \times \delta_{\max}) + (|V| \times |\Delta^{-}_{\mathcal{S},\max}|) + (|\mathbb{K}_f| \times |U|)\right).$$

It is worth noting that the time complexity is the same as that of the differential algorithm proposed in [27]. The proposed optimizations only reduce the hidden constants. Section 4 experimentally shows that our proposed optimizations make the differential approach run quicker.

Similarly, the non-differential approach requires computing the full IFT for each value in $\mathbb{K}_f$, including the contour extraction. It also requires calling node reconstruction for each node to find the maximum distance value of the node.

Thus, the non-differential approach runs in

$$\mathcal{O}\left((|\mathbb{K}_f| \times |D_f|) + (|V| \times |D_f|)\right).$$

Since we usually have $|U|$, $|\delta_{\max}|$, $|\Delta^{-}_{\mathcal{S},\max}| << |D_f|$, our differential approach is usually quicker, as experimentally shown in Sect. 4.

3.5 Extension Summary

In this section, we summarize our proposed optimizations for our previous differential approach [12]:

- We change our approach to find the contours. Previously, we counted the number of background neighbors. In the new approach, we erode the input image and find the contour pixels by comparing their gray level in the input and eroded images.
- We have introduced the set `open`, which keeps the set of pixels open to be conquered by the seeds being processed. Using this set, we could remove the predecessor test of the previous approach.
- We have removed the predecessor map and its computation by rewriting the function `treeRemoval` (Algorithm 2).
- We have introduced an adaptive adjacency relation that avoids visiting redundant regions.

Table 1 compares the previous method with our proposed optimizations.

4 Experimental Results

In this section, we present a quantitative evaluation of the experimental results, comparing running times, as well as a qualitative analysis of how the proposed attribute relates to other existing attributes.

4.1 Execution Time Analysis

We have run experiments to measure the execution time of our proposed method and to compare it against other approaches for computing the maximum distance attribute. We implemented four maximum distance approaches in C++14 and used the *morphotree* library for building and managing morphological trees [30]. We have implemented two non-differential node-reconstruction approaches and two differential approaches as follows:

- *Exact EDT*: We have implemented the EDT computation method described in [31]. This method can compute the exact EDT from a binary image in linear time ($\mathcal{O}(|D_f|)$ by scanning the image domain four times. To compute the maximum distance in component trees, we generate the map `levelToNodes` as done in Algorithm 3 and build each level set of the input image in the same way as the `bin` is created in Algorithm 3. Then, we run the exact EDT approach in the level set and compute the maximum distance for each node associated to the level by reconstructing the node and finding the maximum value of the EDT for the foreground pixels of the node.
- *Non-differential approach*: Similarly to the *Exact EDT* approach, we compute the maximum distance level set by using `levelToNodes` and `bin`. However, instead of using the exact EDT algorithm, we compute the EDT using IFTs (see Sect. 2.4 and 2.6).
- *Differential approach*: We compute the maximum distance attribute of the component tree using the incremental contour and DIFTs as described in [12].
- *Optimized differential approach*: We compute the maximum distance using Algorithms 1, 2, and 3.

We implemented the experiments in single-thread C++14 programs. The only multi-threaded operation was the erosion operation performed in line 7 of Algorithm 3 in the optimized differential approach. For our IFT- and DIFT-based approaches, we adopted the priority queue Q as presented in [32]. The experiments were run on a laptop computer with Ubuntu 19.10, 16GiB of RAM, and an Intel®Core™i7-8750H CPU (2.20Ghz ×12) processor. We have adopted the validation dataset of the Occluded RoadText Challenge from the Robust Reading Competition portal [33]. It is composed of 202 full-HD photographs (1920 × 1080 pixels) taken during driving over different environments. To produce our experimental dataset, we converted the photographs to grayscale. We also produce alternative datasets by scaling down the photographs to 960 × 540, 480 × 270, 240 × 135, and 120 × 68 using the `rescale` method from the module `transform` of scikit-image [34].

We ran each maximum distance attribute computation approach 3 times for each image of all datasets (all image scales) and the mean execution time for each image was registered. From the collected data, we analyze the execution time by image resolution and the number of nodes of the max-tree built from the images of the dataset.

Our optimized differential approach has run the quickest in our analysis by image resolution. It ran 5.00, 4.30, and 2.13 times quicker (on average) than the exact EDT, non-differential, and non-optimized differential approach, respectively. Table 2 presents the computed speed-up per image resolution.

We have also plotted the mean execution time "in log scale" per image resolution to visualize how the execution time grows for each maximum distance computation approach and how they compare in Fig. 9.

Table 1 Summary of optimization introduced on the previous differential approach for computing maximum distance value attribute.

Previous approach	Optimization
Identify contours by counting background neighbors.	Identify contours by comparing the gray level of input and eroded images.
Use predecessor map following the generic differential algorithm from [28].	Use a map `open` to keep the pixel open for conquest.
Use predecessor map to remove the DIFT trees,	Remove DIFT trees using only `cost` and `root` maps,
Fixed adjacency relation,	Adaptive adjacency relation, which avoids redundant regions,

Table 2 Execution time speed-up of our proposed method obtained over the other maximum distance attribute computing approaches organized by image resolution.

	Speed-up		
Resolution	Exact EDT	non-differential	differential
120×68	4.38	3.49	1.65
240×135	5.73	4.97	2.39
480×240	5.52	4.81	2.30
1920×1080	4.04	3.61	1.95
mean	5.00	4.30	2.13

Table 3 Execution time speed-up of our proposed method obtained over the other maximum distance attribute computing approaches by the number of nodes of the component trees.

Number of nodes	Speed-up with respect to:		
intervals	Exact EDT	non-differential	differential
[41665, 103000)	4.21	3.66	1.87
[103000, 131086)	3.99	3.57	1.92
[131086, 163296)	3.94	3.52	1.98
[163296, 298507)	4.02	3.69	2.03
mean	4.03	3.61	1.95

For analyzing the behavior of our implementations of the different approaches in relation to the number of nodes of the component trees, we create buckets for the number of nodes such that each bucket fits approximately 50 samples (images). In this analysis, we just considered the collected data from the experiments of the full-HD images and computed the mean execution time and speed-up of our proposed approach over the other approaches for each bucket. First, we present the speed-up computed for each bucket in Table 3.

We also present the plot of the mean execution time of each method by the number of node intervals in Fig. 10.

The details of the experiments, including source code, scripts, and results, are available on GitHub [35]. In summary, our experiments show that our proposed method with all optimizations described in this paper is the quickest method to compute the maximum distance attribute in the tested dataset. It is between 4.03 and 5, 3.6 and 4.3, and 1.95 and 2.13 times quicker than the exact EDT, non-differential, and non-optimized differential approaches, respectively, to compute the maximum distance attribute on average, depending on the number of nodes of the component trees or image resolutions.

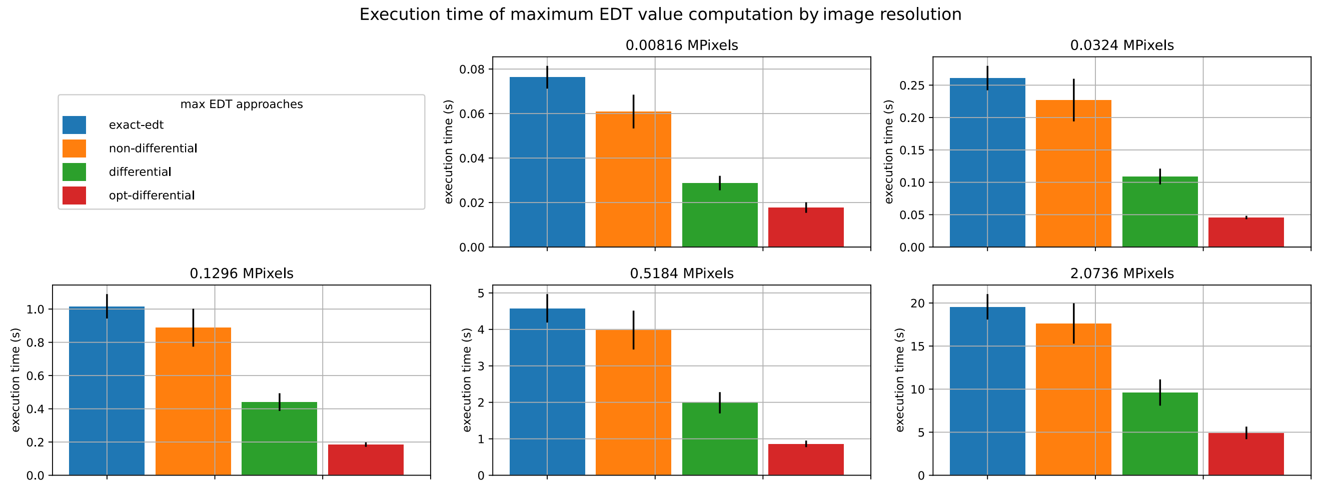


Fig. 9 Average execution time of each maximum distance computation approach per image resolution. Each plot depicts the average execution time for a unique image resolution. Note that the vertical axes are in different scales for different image resolutions.

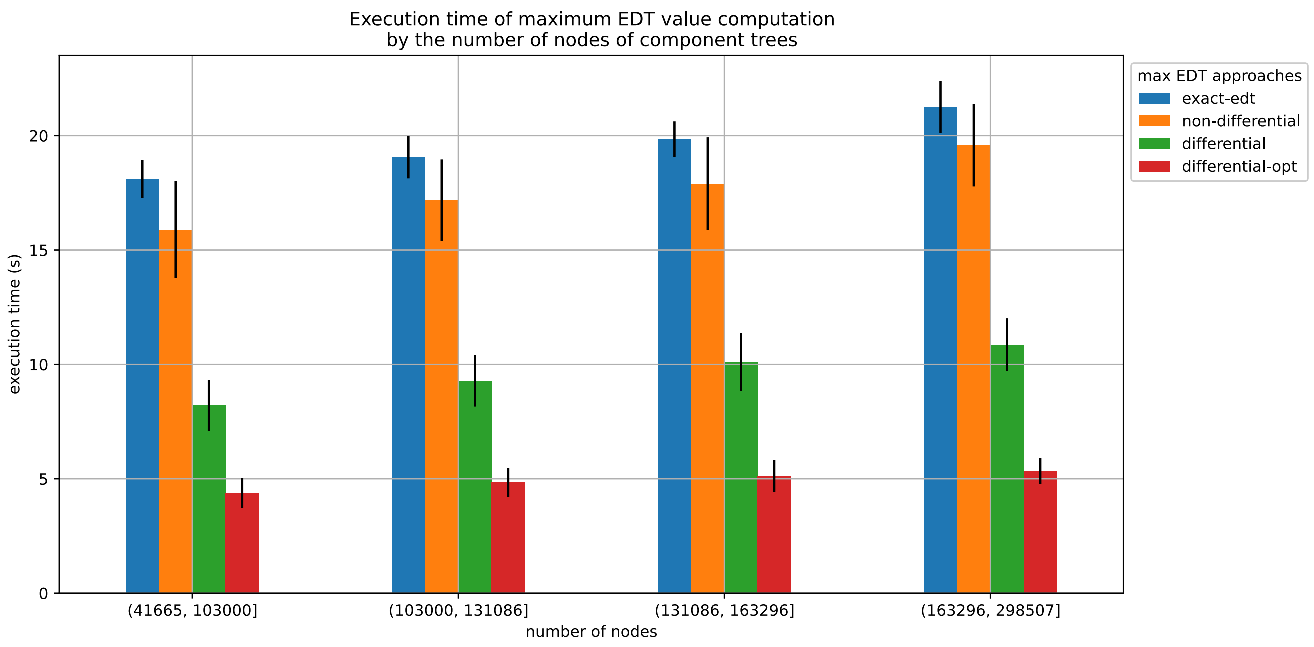


Fig. 10 Average execution time of each maximum distance computation approach per the number of nodes of the component trees.

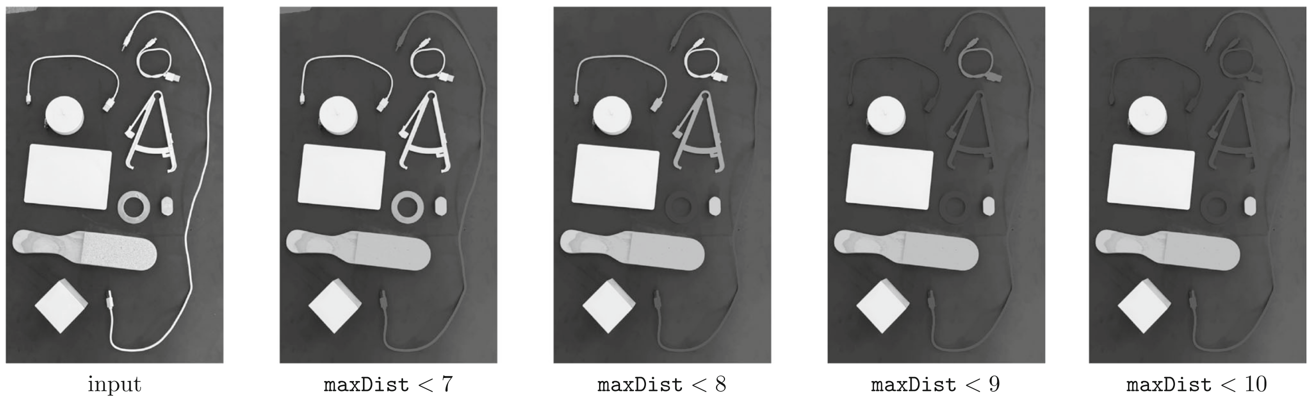


Fig. 11 Sequence of `maxDist` attribute openings demonstrating the removal of objects based on their thickness for threshold values ranging from 7 to 10. The sequence clearly shows how objects are progressively removed as the threshold increases, illustrating the effectiveness of `maxDist` as a simple yet powerful filter. From left to right: input image, and results for `maxDist` < 7, `maxDist` < 8, `maxDist` < 9, and `maxDist` < 10.

4.2 Possible Applications

In this subsection, we explore how the `maxDist` attribute can be applied to define connected operators and segmentation of regions of interest (ROI) in various image processing contexts. Each application described here illustrates the versatility and potential of the `maxDist` attribute to solve specific problems.

4.2.1 Illustrative Examples of Attribute Filters

In mathematical morphology, one of the most interesting applications of attributes in morphological trees is the computation of attribute filters [1, 10]. These filters represent a large class of connected operators that have powerful contour-preservation properties. They act by removing CCs from level sets, based on attribute values. Increasing attributes, such as `maxDist`, are particularly valued in mathematical morphology, as they enable the computation of various classes of order-preserving filters, including attribute openings and closings, as well as extinction value filters. We demonstrate through several examples how the increasing attribute `maxDist` can be used to simplify image structures according to the maximum thickness of regions, highlighting its potential in various filtering approaches.

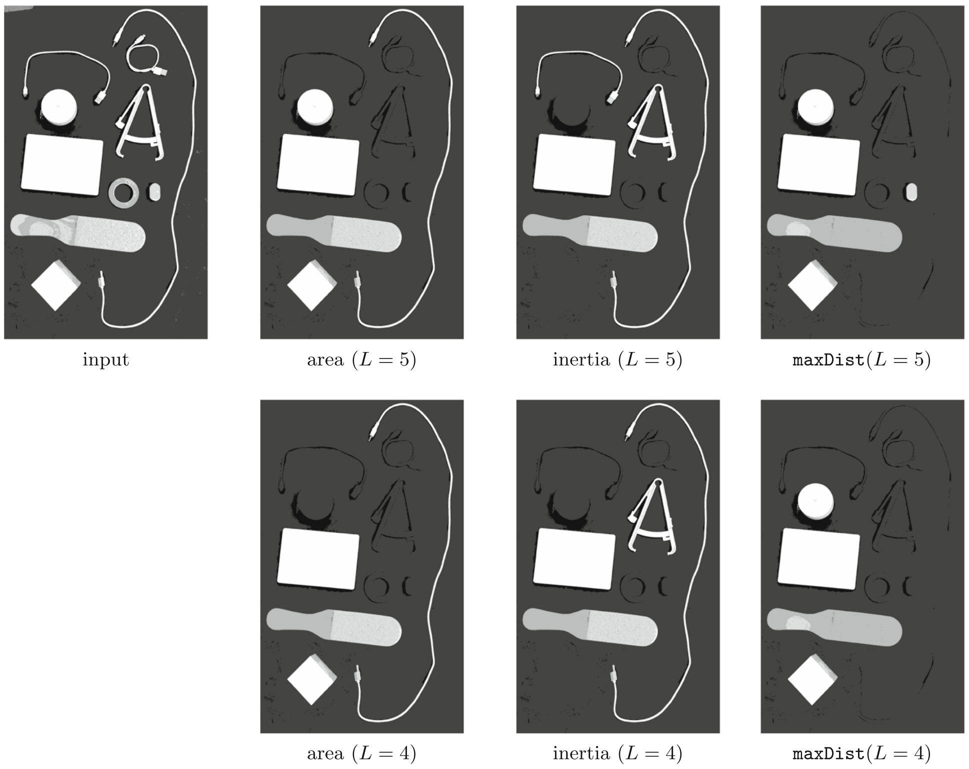


Fig. 12 Demonstration of extinction value filters applied using different attributes to keep L leaves for each attribute. The sequence shows the input image followed by the effects of applying the extinction value filter for area, inertia, and `maxDist` attributes at thresholds $L = 5$ and $L = 4$.

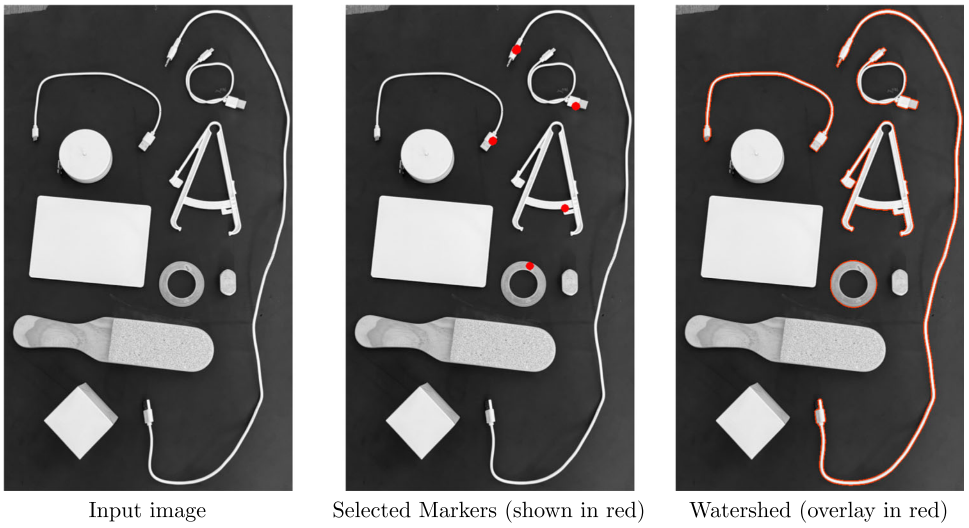


Fig. 13 Watershed segmentation example: input image (left), selected markers (center), and final segmentation result with watershed lines overlay (right).

- *Attribute openings*: Attribute opening is one of the simplest and most effective filters that can be implemented using a component tree. The result of this operator is achieved by reconstructing a tree that has been pruned, where nodes with values of an increasing attribute below a specified threshold are removed. Figure 11 illustrates a sequence of `maxDist` openings with threshold values ranging from 7 to 10. As the threshold increases, objects with thickness below the given value are progressively removed, showcasing the ability of `maxDist` to act as an intuitive and efficient filtering attribute.
- *Extinction Value Filters*: Unlike attribute openings, which remove components based on a fixed threshold, extinction value filters adopt a hierarchical approach. These filters are particularly useful for ranking and selectively removing less significant structures from an image. The intuition is as follows: in a grayscale image interpreted as a topographic map, where the pixel intensities represent elevation, regional maxima correspond to peaks in the landscape. Extinction values quantify the persistence of these peaks, representing the minimum attribute threshold required to eliminate each maximum. A high

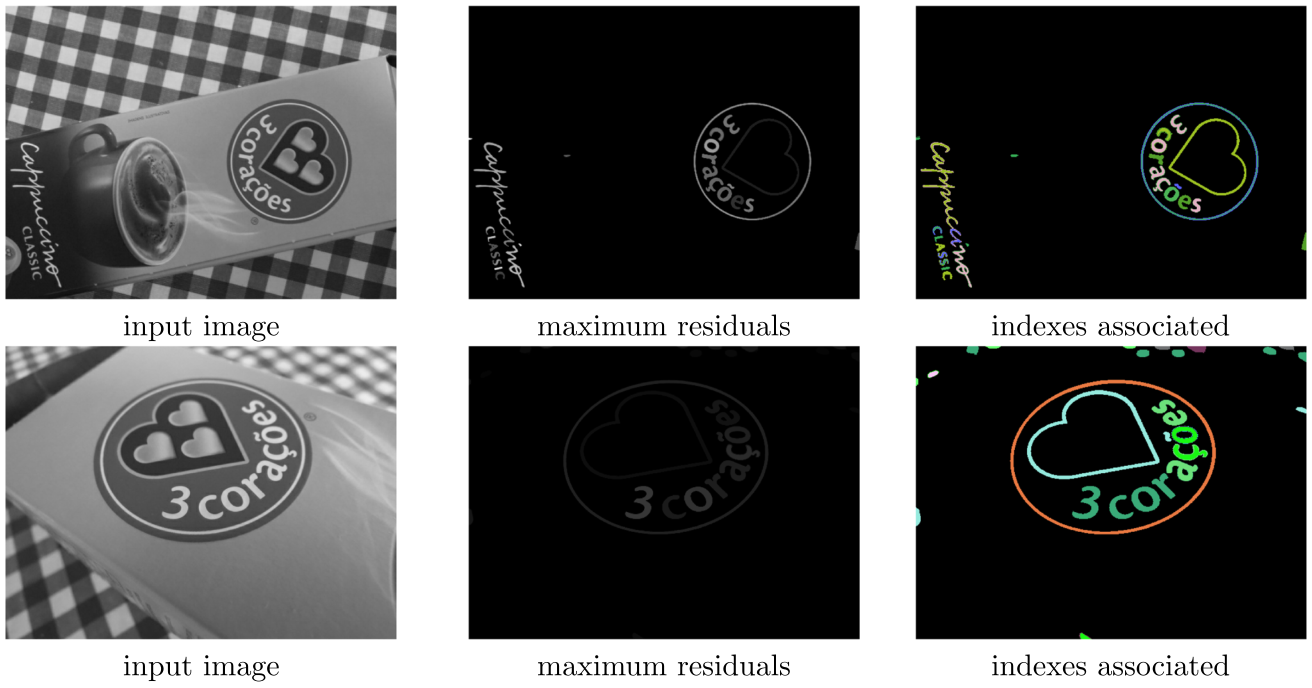


Fig. 14 Application of ultimate `maxDist` opening for detecting objects with thin structures, such as text. The first column shows the input images, the second column highlights the maximum residuals extracted through the ultimate `maxDist` opening, and the third column represents the objects colored randomly based on the indices associated with the maximum residuals.

extinction value means that a regional maximum persists across many threshold levels, indicating a significant structure. Conversely, a low extinction value suggests that the maximum is easily removed and is likely to be a small or insignificant feature. By associating extinction values with the leaves of the component tree, we construct connected filters that organize and prune these leaves based on their importance. Instead of setting a fixed attribute threshold, extinction value filters remove the L least persistent regional maxima, preserving only the L most significant structures. Figure 12 presents a comparison of extinction value filters applied to different attributes, including area, moment of inertia, and `maxDist`. The results illustrate how each attribute influences the selection of preserved maxima. Notably, `maxDist` provides a more precise representation of thickness-based structures, effectively removing thin objects before affecting thicker ones. To highlight the effectiveness of extinction value filtering, Fig. 12 illustrates the removal of thin objects before thicker ones when using `maxDist`. This behavior is particularly useful in applications requiring selective filtering of components based on structural persistence rather than a fixed attribute threshold.

4.2.2 Illustrative Examples of Segmentation

We present two examples of how the `maxDist` attribute can be used to guide the segmentation process. Specifically, we demonstrate its application in marker-based watershed segmentation and ultimate attribute opening. Although we focus on these two methods, the proposed attribute can be employed in other segmentation techniques with the same purpose.

- *Watershed*: This example demonstrates how the `maxDist` attribute can be applied to identify relevant markers that guide the marker-based watershed algorithm in precise segmentation, particularly for objects with specific thickness. The watershed algorithm takes as input the image gradient and the markers obtained from the leaves (regional maxima) of the component tree, whose extinction value with the `maxDist` attribute is in the interval between 7 and 10, and whose areas exceed a given threshold to ensure that markers from noisy regions are not selected. Figure 13 illustrates the segmentation process, showing the input image, the selected markers, and the watershed lines overlay. As observed, objects with low thickness values within the specified range (7 to 10) are correctly segmented.

- *Ultimate* attribute *opening*: Now, we present an example of applying the *ultimate attribute opening* operator using the proposed `maxDist` attribute for the segmentation of objects with thin structures. The *ultimate attribute opening* operator analyzes the evolution of residual values between consecutive operators in an attribute opening scale-space, retaining the maximum residue for each pixel [36, 37]. The residual values provided by these operators can reveal important contrast details in images, allowing the detection of high-contrast objects when a significant residue is produced as they are filtered by one of the attribute openings. Additionally, other associated data, such as the indexes of the operators that produced the maximum residual value, can also be obtained, which are crucial for segmentation applications. In this example, we use the `maxDist` operator to extract objects with thin structures, such as texts. The residuals were extracted only from maximally stable extremal regions (MSERs), as employed in [36]. Figure 14 shows the input image, the extracted maximum residuals, and a randomly colored image based on the indices associated with the maximum residuals. As can be seen, objects with thin structures were effectively captured by the `maxDist` opening.

5 Conclusion

The use of the distance transform as a method to describe the thickness of connected components in binary images has been further enhanced in this extended study. We have significantly improved the computational efficiency of measuring the maximum distance transform value in component trees by refining both the incremental contour computation and the differential image foresting transform (DIFT). These optimizations have enabled our algorithm to achieve processing speeds that are now twice as fast as those obtained with our previous differential method.

In addition to these performance enhancements, we have expanded the scope of applications for the maximum distance attribute. This extended study includes new demonstrations of the attribute's effectiveness through various image processing techniques such as attribute opening, extinction value filters, watershed segmentation, and ultimate attribute openings. These applications highlight the attribute's capability to isolate and emphasize thin structures effectively, thus broadening its utility in practical image processing scenarios.

Further research could explore the application of our method for thickness characterization of 3D medical structures in CT and MR images. Another research avenue could investigate adapting our proposed algorithm to efficiently compute other binary image transforms, such as the medial axis transform for component trees [38], or examine additional attributes derived from thickness. Additionally, parallelization strategies could be explored to further improve computational efficiency. Existing partitioned algorithms for the IFT [39] and linear-time EDT algorithms [31, 40, 41] offer promising approaches for parallel execution. Another research direction involves investigating hardware-based implementations, such as FPGA architectures [42], to accelerate processing in large-scale applications. These optimizations could significantly enhance the scalability and applicability of the proposed method in real-world scenarios. We also plan to investigate the behavior of our proposed algorithm for different (higher) bit-depth images.

Glossary

4-connected adjacency relation	N_4	2.1
8-connected adjacency relation	N_8	2.1
4-connected adjacent arcs	$\mathcal{A}_4$	2.1
8-connected adjacent arcs	$\mathcal{A}_8$	2.1
4-connected neighborhood of p	$\mathcal{N}_4(p)$	2.1
8-connected neighborhood of p	$\mathcal{N}_8(p)$	2.1
$\mathcal{A}$-contour of X	$\partial_{\mathcal{A}}(X)$	2.3
$\mathcal{A}_4$-contour of X	$\partial_{\mathcal{A}_4}(X)$	2.3
$\mathcal{A}_8$-contour of X	$\partial_{\mathcal{A}_8}(X)$	2.3
Adaptive adjacency relation	$\mathcal{N}_{adapt}(s, k_s)$	3.2
Adjacent arcs	$\mathcal{A}$	2.1
Arbitrary neighborhood of p	$\mathcal{N}(p)$	2.1
Arbitrary node in a tree	N	2.2
Binary image built for our proposed algorithm	`bin`	3.3
Border to maximum Euclidean distance transform value map	`Bedt`	3.2
Children of N in T	$\mathtt{children}(N, T)$	2.2
Compact component tree	$\hat{T}_f$	2.2
Compact component tree edges	$\hat{E}(T_f)$	2.2
Compact component tree nodes	$\hat{V}(T_f)$	2.2
Compact node pixel	CNP	2.2
Compact representation of node N	$\hat{N}$	2.2
Component tree of f	T_f	2.2

Connectivity function for computing Euclidean distance transform	$\Psi_{\text{Euc}}(\pi(p,q))$	2.6
Connected component	CC	1
Connected component of $G_{X,\mathcal{A}}$ that contains p	$CC(G_{X,\mathcal{A}}, p)$	2.1
Connected components of the lower level sets	$\mathcal{L}(f, \mathcal{A})$	2.1
Connected components of the upper level sets	$\mathcal{U}(f, \mathcal{A})$	2.1
Contours of node N	$\texttt{Ncontour}$	3.3
Differential Image Foresting Transform	DIFT	1
Distance transform	DT	2.6
Distance transform value at pixel p	$\texttt{edt}(X, p)$	2.6
Eroded f	$f_{\texttt{eroded}}$	3.3
Extended adaptive adjacency	$\mathcal{N}^*_{adapt}(s, k_s)$	3.2
Full High Definition	Full-HD	1
General attribute	$\texttt{attr}$	2.2
General binary image	X	2.1
General grayscale image	f	2.1
Geodesic cost function	Ψ_{Geo}	2.4
Graph representation of a binary image	$G_{X,\mathcal{A}}$	2.1
Graph representation of a grayscale image	$\mathcal{G}_{f,\mathcal{A}}$	2.1
Gray level associated to node N	$\texttt{level}(N, T_f)$	2.2
Grayscale erosion of f by structuring element N_4	$\varepsilon_f(\text{N}_4)$	3.3
i-th binary image of a sequence of nested binary images	X_i	3.2
IFT and DIFT root map	$\texttt{root}$	2.4
IFT and DIFT connectivity function	Ψ	2.4
IFT and DIFT cost function	$\texttt{cost}$	2.4
Image Foresting Transform	IFT	1
Image codomain	$\mathbb{K}_f$	2.1
Image domain	D_f	2.1
Infimum	inf	2.2
Invalid pixel or value	**NIL**	2.4
Image domain with an invalid pixel	D^*_f	2.4
Level to nodes map	$\texttt{level}ToNodes$	3.3
Lower level set of f wrt. λ	$[f \leq \lambda]$	2.1
Maintained, inserted, and removed seed sets	MIR	3.3
Map that maps a p to an adaptive index	$\texttt{adjmap}$	3.2
Maximally Stable Extremal Regions	MSER	4.2
Maximum distance transform value attribute	$\texttt{maxDist}$	3.1
Node to contours map	$\texttt{contours}$	3.3
Number of children of N	c_N	3.4
Number of leaves	L	4.2
Parent of N in T	$\texttt{parent}(N, T)$	2.2
Path from s to p computed by an IFT	$\pi^*(s, p)$	2.4
Path from p to q	$\pi(p, q)$	2.1
Predecessor path	$\texttt{pred}$	2.4
Priority queue	Q	3.2
Region of Interest	ROI	4.2
Result of an IFT considering $\mathcal{S}^i$	$IFT_{(\mathcal{S}^1)}$	2.4
Root node of the tree T	$\texttt{rootTree}(T)$	2.2
Rooted directed tree	T	2.2
Seeds that were inserted into $\mathcal{S}^{i-1}$ to create $\mathcal{S}^i$	$\Delta^+_{\mathcal{S}^i}$	3.2
Seeds that are removed from $\mathcal{S}^i$ to create $\mathcal{S}^i$	$\Delta^-_{\mathcal{S}^i}$	3.2
Set of all adjacent pixels (arbitrary adjacency relation)	$\mathcal{A}(X)$	2.1
Set of all possible paths in $G_{X,\mathcal{A}}$	$\Pi(G_{X,\mathcal{A}})$	2.4
Set of all possible paths in $G_{X,\mathcal{A}}$ ending at q	$\Pi_q(G_{X,\mathcal{A}})$	2.4
Set of connected components of $G_{X,\mathcal{A}}$	$CC(G_{X,\mathcal{A}})$	2.1
Set of (directed) edges of T	$E(T)$	2.2
Set of nodes between N and P	$\texttt{between}(N, P, T_f)$	2.2
Set of positive integers	$\mathbb{Z}^+$	3.3
Set of positive real numbers	$\mathbb{R}^+$	2.4
Set of seeds	$\mathcal{S}$	2.4
Set of seeds for the i-th IFT in a sequence of IFTs	$\mathcal{S}^i$	2.4

Set of vertices/nodes of T	$V(T)$, V	2.2
Small component of p	$\mathcal{SC}(T_f, p)$	2.2
Standard Euclidean L_2 norm	$\|.\|$	2.6
Supremum	sup	2.2
The biggest $\Delta^-_{\mathcal{S}}$	$\Delta^-_{\mathcal{S},\max}$	3.4
The biggest set of updated nodes in DIFT	U	3.4
The length of the longest contour	$\delta_{\max}$	3.4
Upper level set of f wrt. λ	$[f \geq \lambda]$	2.1
Variable that keeps the set of pixels in $X_i \setminus X_{i-1}$	`open`	3.2

Appendix A Worst Case Analysis of Non-exact DT by IFT

As stated in Sect. 2.6, when $|\mathcal{S}| \geq 3$, the IFT algorithm with connectivity function Ψ_{Euc} may not output the exact DT, if the adjacency relation is not big enough. An example of a non-exact DT calculation by IFT using Ψ_{Euc} in a graph with $\mathcal{A}_8$ is presented in Fig. 15. However, in practice, the DT errors by IFT with 8-connected adjacency and Ψ_{Euc} are so small and unusual that they can be neglected for most applications. In this appendix section, we present an analysis of the upper bound of the error of non-exact DT by IFT using Ψ_{Euc} with $\mathcal{A}_8$, which follows arguments similar to those presented in [43].

As indicated in [19], the computed path tree of each seed $r \in \mathcal{S}$ will always contain the subset of its discrete Voronoi region which is 8-connected to r. The approximation error occurs when the Voronoi polygon has a thin corner that passes between two neighboring lateral pixels, because the regions of the discrete Voronoi diagram may not be 8-connected (Fig. 15).

The worst case is the one depicted in Fig. 16, where the pixels in D and E are neighbors of P through $\mathcal{A}_8$, but D and E do not belong to the Voronoi cell of seed in C. The pixel at P will be incorrectly assigned to the seed at B with distance $\|P - B\| = r + 1$, where $\|\cdot\|$ denotes the standard Euclidean L_2 norm on $D_f \subset \mathbb{Z}^2$. Note that by triangle inequality of triangle CEP we have that $d < r + 1$, therefore C should be the correct seed to conquer P. The error ε is:

$$\varepsilon = r + 1 - d. \tag{A1}$$

The maximum error occurs when d is as small as possible (without causing the Voronoi cell of seed in C to expand over the two neighbor pixels D and E) which means that C is at distance $r + \delta$ from E with $\delta \approx 0$. Applying the Pythagorean theorem and some geometric relationships we can conclude that:

$$\begin{aligned} r^2 &= (d \cdot \cos\theta - 1)^2 + (d \cdot \sin\theta)^2, \qquad \text{(A2)}\\ r^2 &= d^2 \cdot \cos^2\theta - 2 \cdot d \cdot \cos\theta + 1 + d^2 \cdot \sin^2\theta, \\ r^2 &= d^2 \cdot (\cos^2\theta + \sin^2\theta) - 2 \cdot d \cdot \cos\theta + 1, \\ cos\theta &= \frac{1 + d^2 - r^2}{2 \cdot d}. \qquad \text{(A3)} \end{aligned}$$

To find angle θ, we consider the following relationships between points:

$$\begin{aligned} D &= P - (1, 1), \qquad \text{(A4)}\\ C &= P - (d \cdot \sin\theta, 1 + d \cdot \cos\theta - 1) \\ &= P - (d \cdot \sin\theta, d \cdot \cos\theta). \qquad \text{(A5)} \end{aligned}$$

Combining the previous equations, we have:

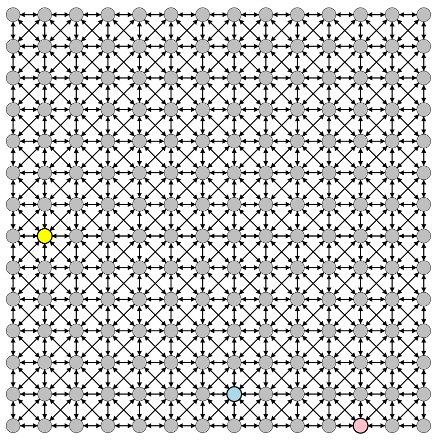
$$\begin{aligned} C - D &= P - (d \cdot \sin\theta, d \cdot \cos\theta) - P + (1, 1), \\ C - D &= (1 - d \cdot \sin\theta, 1 - d \cdot \cos\theta). \qquad \text{(A6)} \end{aligned}$$

Given that $\|C - D\| = r + 1 - \sqrt{2}$ and combining with Eq.A6, we have:

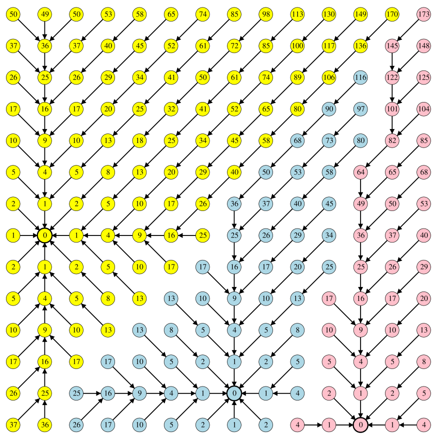
$$\begin{aligned} \|C - D\| = \sqrt{(1 - d \cdot \sin\theta)^2 + (1 - d \cdot \cos\theta)^2} &= r + 1 - \sqrt{2}, \\ (1 - d \cdot \sin\theta)^2 + (1 - d \cdot \cos\theta)^2 &= (r + 1 - \sqrt{2})^2, \\ d^2 \cdot (\sin^2\theta + \cos^2\theta) + 2 - 2 \cdot d \cdot (\sin\theta + \cos\theta) &= (r + 1 - \sqrt{2})^2, \\ \sin\theta + \cos\theta &= \frac{d^2 + 2 - (r + 1 - \sqrt{2})^2}{2 \cdot d}. \qquad \text{(A7)} \end{aligned}$$

By combining Eqs. A3 and A7, we have:

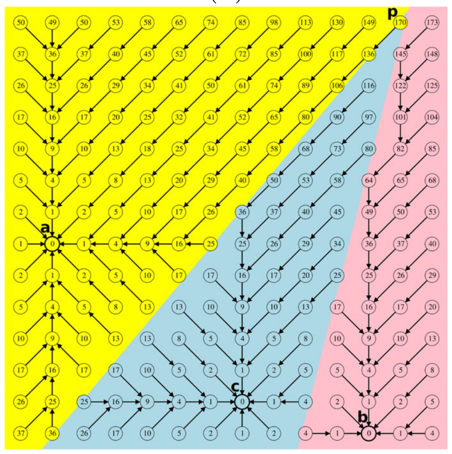
$$\begin{aligned} \sin\theta &= \frac{d^2 + 2 - (r + 1 - \sqrt{2})^2}{2 \cdot d} + \frac{r^2 - 1 - d^2}{2 \cdot d}, \\ \sin\theta &= \frac{1 - (r + 1 - \sqrt{2})^2 + r^2}{2 \cdot d}, \\ \sin\theta &= \frac{r^2 + 1 - \left[r^2 + 2 \cdot r \cdot (1 - \sqrt{2}) + (1 - \sqrt{2})^2\right]}{2 \cdot d}, \\ \sin\theta &= \frac{1 - 2 \cdot r + 2 \cdot r \cdot \sqrt{2} - (1 - 2 \cdot \sqrt{2} + 2)}{2 \cdot d}, \\ \sin\theta &= \frac{(-2 \cdot r)(1 - \sqrt{2}) - 2 + 2 \cdot \sqrt{2}}{2 \cdot d}, \\ \sin\theta &= \frac{(\sqrt{2} - 1) \cdot (r + 1)}{d}, \qquad \text{(A8)} \end{aligned}$$



(a)



(b)



(c)

Fig. 15 Example of a non-exact DT calculation by IFT using Ψ_{Euc} in a graph with $\mathcal{A}_8$. **a** Input graph with three marked seeds given by the colored nodes. **b** The arrows indicate the predecessor of each node in the computed paths, and their costs by Ψ_{Euc} are indicated inside the nodes. **c** The Voronoi diagram is depicted under the nodes, indicating that node p should belong to the Voronoi cell of seed c in an exact DT. The computed Euclidean distance to node p is $\sqrt{\texttt{cost}(p)} = \sqrt{170} \approx 13.0384$, while the exact value would be $\|c - p\| = \sqrt{5^2 + 12^2} = \sqrt{169} = 13$, leading to an error of 0.0384.

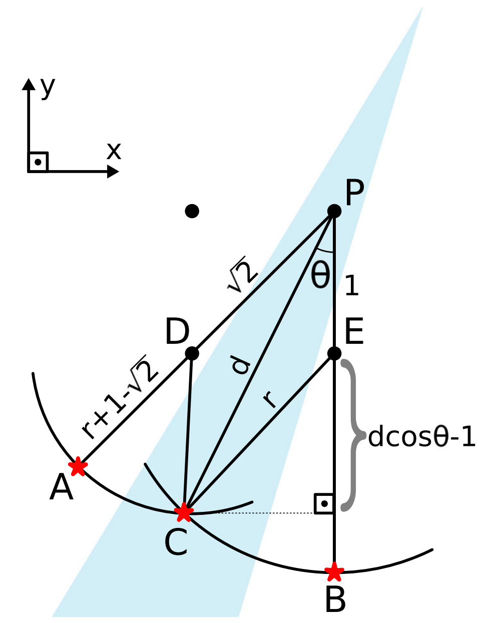


Fig. 16 Worst-case error for non-exact DT calculation by IFT using Ψ_{Euc} in a graph with $\mathcal{A}_8$.

which for large r becomes:

$$\sin\theta \approx (\sqrt{2} - 1) \implies \theta \approx 24.46980052^\circ \qquad \text{(A9)}$$

Isolating d in Eq. A2, we have:

$$d = \sqrt{r^2 - 1 + \cos^2\theta} + \cos\theta. \qquad \text{(A10)}$$

which for large r becomes:

$$d = r + \cos\theta \qquad \text{(A11)}$$

Therefore the error ε for large r becomes:

$$\varepsilon = r + 1 - (r + \cos\theta) = 1 - \cos\theta \approx 0.090 \qquad \text{(A12)}$$

Therefore, for large r, the error is bounded to be less than 0.090 pixel units, which is negligible for most practical purposes.

The resulting errors for small r values ($1 \leq r \leq 20$), were analyzed in detail in [43], and all of them fell below the calculated upper bound of 0.090 pixel units. Also in our experiments, no approximation error exceeding this value was found.

Acknowledgements 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.

Author Contributions Everybody worked on designing the main approach. D.J.S implemented the source code with P.A.V.M., ran the execution time experiments, and partially wrote Sections 1, 2, 3 and 4. P.A.V.M. was crucial in designing the main approach, particularly on the part regarding IFT and DIFT, he also wrote and reviewed the manuscript especially Sections 2.4, 2.5, and 3.2. W.A.L.A. partially wrote the manuscript especially Section 4.2 and implemented the applications described in Section 2. R.F.H., J.K. and J.B.T.M. manage the work, review the manuscript and supervise the work. R.F.H. also scheduled meetings to discuss the work and steered the progress of the project.

Data availability We have made available all source code and data collected from our execution time measurement experiment.They are available in a GitHub Respository and it is included as a reference in the manuscript.

Declarations

Competing interests The authors declare no competing interests.

References

1. Salembier, P., Wilkinson, M.H.F.: Connected operators. IEEE Signal Process. Mag. **26**(6), 136–157 (2009)
2. Neumann, L., Matas, J.: Real-time scene text localization and recognition. In: Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference On, pp. 3538–3545. IEEE (2012)
3. Alves, W.A., Gobber, C.F., Araújo, S.A., Hashimoto, R.F.: Segmentation of retinal blood vessels based on ultimate elongation opening. In: International Conference Image Analysis and Recognition, pp. 727–733. Springer (2016)
4. 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)
5. Silva, D.J., Alves, W.A.L., Hashimoto, R.F.: Incremental bit-quads count in component trees: theory, algorithms, and optimization. Pattern Recogn. Lett. **129**, 33–40 (2020)
6. Climent, J., Oliveira, L.S.: A new algorithm for number of holes attribute filtering of grey-level images. Pattern Recogn. Lett. **53**, 24–30 (2015)
7. Gray, S.B.: Local properties of binary images in two dimensions. IEEE Trans. Comput. **C–20**(5), 551–561 (1971)
8. Najman, L., Couprie, M.: Building the component tree in quasi-linear time. IEEE Trans. Image Process. **15**(11), 3531–3539 (2006)
9. Silva, A.G., Lotufo, R.A.: New extinction values from efficient construction and analysis of extended attribute component tree. In: 2008 XXI Brazilian Symposium on Computer Graphics and Image Processing, pp. 204–211 (2008)
10. Salembier, P., Oliveras, A., Garrido, L.: Antiextensive connected operators for image and sequence processing. IEEE Trans. Image Process. **7**(4), 555–570 (1998)
11. Telea, A.: Image Visualization. Data Visualization: Principles and Practice, 2nd edn., pp. 363–379. CRC Press, Boca Raton (2014)
12. 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 computation in component trees. In: Brunetti, S., Frosini, A., Rinaldi, S. (eds.) Discrete Geometry and Mathematical Morphology, pp. 67–79. Springer, Cham (2024)
13. Ramalingam, G., Reps, T.: On the computational complexity of dynamic graph problems. Theor. Comput. Sci. **157**(2), 257–289 (1996)
14. Demetrescu, C., Italiano, G.F.: A new approach to dynamic all pairs shortest paths. J. ACM **51**(6), 968–992 (2004)
15. Bernstein, A., Roditty, L.: Improved dynamic algorithms for maintaining approximate shortest paths under deletions. In: Proceedings of the ACM Symposium on Theory of Computing (STOC), pp. 745–754 (2011)
16. Thorup, M., Karger, D.R.: Dynamic graph algorithms with applications. In: Seventh Scandinavian Workshop on Algorithm Theory (2000)
17. Najman, L., Couprie, M.: Building the component tree in quasi-linear time. IEEE Trans. Image Process. **15**(11), 3531–3539 (2006)
18. Gonzalez, R.C., Woods, R.E.: Digital Image Processing, 4th edn. Pearson, New York (2018)
19. Falcão, A.X., Stolfi, J., Alencar Lotufo, R.: The image foresting transform: theory, algorithms, and applications. IEEE Trans. Pattern Anal. Mach. Intell. **26**(1), 19–29 (2004)
20. Ciesielski, K.C., Falcão, A.X., Miranda, P.A.V.: Path-value functions for which Dijkstra's algorithm returns optimal mapping. J. Math. Imaging Vis. **60**(7), 1025–1036 (2018)
21. Falcão, A.X., Bergo, F.P.: Interactive volume segmentation with differential image foresting transforms. IEEE Trans. Med. Imaging **23**(9), 1100–1108 (2004)
22. Falcão, A.X., Costa, L.F., Cunha, B.S.: Multiscale skeletons by image foresting transform and its application to neuromorphometry. Pattern Recogn. **35**(7), 1571–1582 (2002)
23. 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., Sanniti di Baja, G. (eds.) Skeletonization, pp. 43–70. Academic Press, London (2017)
24. da Torres, R.S., Falcão, A.X., da Costa, L.F.: A graph-based approach for multiscale shape analysis. Pattern Recogn. **37**(6), 1163–1174 (2004)
25. Torres, R.S., Falcão, A.X.: Contour salience descriptors for effective image retrieval and analysis. Image Vis. Comput. **25**(1), 3–13 (2007)
26. Andalo, F.A., Miranda, P.A.V., Silva 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 (2007)
27. 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 computation in component trees. In: Brunetti, S., Frosini, A., Rinaldi, S. (eds.) Discrete Geometry Math. Morphol., pp. 67–79. Springer, Cham (2024)
28. Condori, M.A.T., Cappabianco, F.A.M., Falcão, A.X., Miranda, P.A.V.: An extension of the differential image foresting transform and its application to superpixel generation. J. Vis. Commun. Image Represent. **71**, 102748 (2020)
29. Soille, P.: Morphological Image Analysis: Principles and Applications. Springer, Berlin, Heidelberg (2004)
30. Silva, D.J.: Morphotree. https://github.com/dennisjosesilva/morphotree. accessed in 1st December of 2022 (2022)

31. Meijster, A., Roerdink, J.B.T.M., Hesselink, W.H.: A general algorithm for computing distance transforms in linear time. In: Goutsias, J., Vincent, L., Bloomberg, D.S. (eds.) Mathematical Morphology and its Applications to Image and Signal Processing, pp. 331–340. Springer, Boston (2000)
32. Falcao, A.X., Udupa, J.K., Miyazawa, F.K.: An ultra-fast user-steered image segmentation paradigm: live wire on the fly. IEEE Trans. Med. Imaging **19**(1), 55–62 (2000)
33. Tom, G., Mathew, M., Mondal, A., Weinman, J., Karatzas, D., Jawahar, C.V.: Robust Reading Competition—Occluded RoadText. https://rrc.cvc.uab.es/?ch=29. Accessed in 23rd July of 2024 (2024)
34. Scikit-image: API Reference. https://scikit-image.org/docs/stable/api/skimage.transform.html#skimage.transform.rescale. accessed in 23rd July of 2024 (2024)
35. Silva, D.J., Miranda, P.A.V., Alves, W.A.L., Hashimoto, R.F., Kosinka, J., Roerdink, J.B.T.M.: Efficient Maximum Euclidean Distance Transform Computation in Component Trees Using the Differential Image Foresting Transform-Experiment analysis webpage and source code. https://github.com/dennisjosesilva/maxdist-diff-extended. accessed in 11 May of 2025
36. Alves, W.A.L., Hashimoto, R.F., Marcotegui, B.: Ultimate levelings. Comput. Vis. Image Underst. **165**(C), 60–74 (2017)
37. Alves, W.A.L., Gobber, C.F., Silva, D.J., Morimitsu, A., Hashimoto, R.F., Marcotegui, B.: Image segmentation based on ultimate levelings: from attribute filters to machine learning strategies. Pattern Recogn. Lett. **133**, 264–271 (2020)
38. 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)
39. Bergo, F.P., Falcão, A.X.: A partitioned algorithm for the image foresting transform. In: Mathematical Morphology and Its Applications to Signal and Image Processing (ISMM), pp. 425–436 (2007)
40. Saito, T., Toriwaki, J.-I.: New algorithms for Euclidean distance transformation of an n-dimensional digitized picture with applications. Pattern Recogn. **27**(11), 1551–1565 (1994)
41. Hirata, T.: A unified linear-time algorithm for computing distance maps. Inf. Process. Lett. **58**(3), 129–133 (1996)
42. Cappabianco, F.A., Araujo, G., Falcao, A.X.: The image forest transform architecture. In: International Conference on Field-Programmable Technology, pp. 137–144. IEEE (2007)
43. Danielsson, P.-E.: Euclidean distance mapping. Comput. Graph. Image Process. **14**(3), 227–248 (1980). https://doi.org/10.1016/0146-664X(80)90054-4

Dennis J Silva received his PhD double-degree in computer science from the University of São Paulo (Brazil) and the University of Groningen (the Netherlands) in 2025. Currently, he is a postdoc researcher at the Université de Reims Champagne Ardenne (France). His research interests include image processing using mathematical morphology and computer graphics.

Paulo A. V. Miranda is currently professor at the Institute of Mathematics and Statistics (IME) of the University of São Paulo (USP), SP, Brazil. He received a B.Sc. in Computer Engineering (2003) and a M.Sc. in Computer Science (2006) from the University of Campinas (UNICAMP), SP, Brazil. During 2008-2009, he was with the Medical Image Processing Group, Department of Radiology, University of Pennsylvania, Philadelphia, USA, where he worked on image segmentation for his doctorate. He got his doctorate in Computer Science from the University of Campinas (UNICAMP) in 2009. After that, he worked as a post-doctoral researcher in a project in conjunction with the professors of the Department of Neurology, UNICAMP. He is a research productivity fellow at CNPq and a board member of IAPRTC18-Discrete Geometry and Mathematical Morphology. He has experience in computer science, with emphasis on graph-based approaches to image analysis and computer vision, hierarchical analysis, and mathematical morphology.

Wonder Alexandre Luz Alves received his Ph.D. in Computer Science from the University of São Paulo, Brazil, in 2015. He is currently an Associate Professor in the Graduate Program in Informatics and Knowledge Management at Nove de Julho University, São Paulo. His research interests include machine learning, computer vision, and mathematical morphology, with a particular focus on image analysis and pattern recognition.

Ronaldo F. Hashimoto is currently an Associate Professor at the Institute of Mathematics and Statistics (IME) of the University of São Paulo (USP), Brazil, where he also serves as Director. He received a B.Sc. in Physics (1988), an M.Sc. (1994), and a Ph.D. (2000) in Applied Mathematics from the University of São Paulo. He carried out postdoctoral research at Texas A&M University between 2000 and 2002. His research spans Computer Science, Image Processing, Machine Learning, and Bioinformatics, with specific interests in Hierarchical Image Representations, Mathematical Morphology, Biological Networks, and Boolean Network Models. He has contributed to interdisciplinary studies in computational biology, cancer systems biology, and infectious diseases such as malaria and COVID-19. He has supervised over 30 graduate students and postdocs, and is a member of the editorial board of Scientific Reports.

Jiri Kosinka is an Associate Professor at the Bernoulli Institute of the Faculty of Science and Engineering, University of Groningen, where he leads the Scientific Visualization and Computer Graphics research group. His interests inclu de topics in the area of visual computing, with particular emphasis on geometric modelling, computer graphics, and image vectorisation. He currently serves as Associate Editor for two Elsevier journals, Computer Aided Design and Graphical Models. He has been involved in organising several conferences in his field, such as the SIAM Conference on Computational Geometric Design as programme co-chair in 2021 and conference co-chair in 2023. He has co-authored over 100 scientific publications and served on more than 40 international programme committees.

Jos Roerdink received his PhD degree in theoretical physics from the University of Utrecht in 1983. He is emeritus professor of Scientific Visualization and Computer Graphics at the University of Gro ningen, the Netherlands. His re search interests include mathematical morphology and biomedical visualization.