Optimization Heuristics for Computing the Voronoi Skeleton

Dmytro Kotsur[1](✉) and Vasyl Tereshchenko[2]

[1] Software Competence Center Hagenberg (SCCH), Softwarepark 21, 4232 Hagenberg, Austria
dkotsur@gmail.com

[2] Taras Shevchenko National University of Kyiv, Volodymyrska Str. 60, Kiev 01033, Ukraine
vtereshch@gmail.com

Abstract. A skeletal representation of geometrical objects is widely used in computer graphics, computer vision, image processing, and pattern recognition. Therefore, efficient algorithms for computing planar skeletons are of high relevance. In this paper, we focus on the algorithm for computing the Voronoi skeleton of a planar object represented by a set of polygons. The complexity of the considered algorithm is $O(N \log N)$, where N is the total number of polygon's vertices. In order to improve the performance of the skeletonization algorithm, we proposed theoretically justified shape optimization heuristics, which are based on polygon simplification algorithms. We evaluated the efficiency of such heuristics using polygons extracted from MPEG 7 CE-Shape-1 dataset and measured the execution time of the skeletonization algorithm, computational overheads related to the introduced heuristics and the influence of the heuristic onto the accuracy of the resulting skeleton. As a result, we established the criteria allowing us to choose the optimal heuristics for Voronoi skeleton construction algorithm depending on the critical system's requirements.

Keywords: Voronoi diagram · Voronoi graph · Skeleton · Optimization · Heuristics

1 Introduction

The skeletal representation of the planar object is essential for many problems of computer vision and pattern recognition, image processing, computer graphics and visualization [1]. Skeletons are widely used for shape matching [2, 3], optical character recognition [4] and image retrieval [2, 5]. In the area of biomedical image processing, skeletonization methods are extensively applied to compute the central line of thin objects. For example, one can extract the skeletal graph representing the retinal blood vessels topology [6, 7]. A similar technique can be applied to segment biological neural networks [8]. One can also use skeletonization methods to segment cellular filamentous structures using microscopy images [9, 10]. Thus, fast and accurate algorithms for computing the skeleton of the geometrical objects are of high relevance.

Related Work. Existing algorithms for computing the skeleton can be classified based on the type of processed data. For example, morphological thinning techniques are extensively used for computing the skeleton of a binary image [11–13]. They allow us to obtain a pixel-level representation of the thin skeleton. On the next step, such representation can be converted into a graph using the vectorization methods described [14, 15]. However, the accuracy of the skeleton is bounded by the resolution of the pixel grid. Moreover, many of these methods are not rotation-invariant [11, 12].

Other techniques are based on central line tracing. They are commonly used to segment thin-line structures on an image (e.g., axons, dendrites of neurons [16], blood vessels [17], filamentous structures [17, 18]). These methods can directly represent the skeleton as a connected graph. However, due to the iterative nature of these methods, the execution time may vary significantly.

Another class of methods allows us to compute a skeleton of an object, whose shape is represented by simple polygons. Such polygons can be either sampled directly from a vector graphics data or can be extracted from a binary image using the tracing techniques (e.g., Marching squares [19]). Methods to construct the straight skeleton using the polygon shrinking technique with $O(N \log N)$ complexity are described in papers [20, 21]. A linear complexity method for a simple polygon without holes was introduced in [22]. A more general approach for constructing the skeleton of an arbitrary object with holes employs the Voronoi diagram [23, 24], which has computational complexity $O(N \log N)$, N is a number of primitives. In comparison to the techniques above, this approach allows us to directly compute a rotation-invariant thin skeleton of an object as a graph. Moreover, one can also employ the properties of the Voronoi diagram to solve the related geometrical problems [25] (e.g., finding fast a convex hull, nearest neighbor, maximal inscribed disk). However, due to a large number of the processed simple primitives, such method can become computationally costly. Therefore, we focus on the Voronoi-based skeletonization methods and on heuristic techniques allowing us to speed up such methods by employing the shape simplification techniques.

2 Problem Statement

We assume that a planar object has G_1-continuous boundaries (except for a finite number of G_0-continuous points – see *critical points* below). The object's boundaries are represented by a set of simple planar polygons $\mathcal{S} := \{\mathcal{P}^0, \mathcal{P}^1, \ldots, \mathcal{P}^m\}$, where polygon $\mathcal{P}^k$ is defined as an ordered set of its vertices $p_1^k, p_2^k, \ldots, p_{M_k}^k$. Polygon $\mathcal{P}_0$ corresponds to the outer contour of the object. $R := \mathcal{P}_0 \setminus \bigcup_{i=1}^m \mathcal{P}_i$ defines the object's domain.

Let's denote the set of open line segments (LS') corresponding to the polygon $\mathcal{P}_k$ by $\mathcal{L}_k := \mathcal{L}(\mathcal{P}_k) = \left\{ l_i^k := \left(p_i^k, p_{i+1}^k\right) | i = 1, \ldots, M_k, p_{M_k+1}^k = p_1^k \right\}$ and the set of all vertices (line segment's endpoints) by $\mathcal{Q} = \bigcup_{k=0}^{m} \bigcup_{j=1}^{M_k} \{p_i^k\}$. A set $\mathcal{L} := \bigcup_{k=1}^{m} \mathcal{L}_k$ contains all line segments of $\mathcal{S}$.

Definition 1. The *Voronoi cell* [29] corresponding to an element $u \in \mathcal{L} \cup \mathcal{Q}$ is defined as a locus of points:

$$\mathcal{VC}(u) = \left\{ p \in \mathbb{R}^2 | dist(p,u) \leq dist(p,w), w \neq u, w \in \mathcal{L} \cup \mathcal{Q} \right\} \tag{1}$$

Definition 2. The *Voronoi diagram* [29] of a set of line segments $\mathcal{L}$ (with endpoints $\mathcal{Q}$) is defined as a set of all Voronoi cells:

$$\mathcal{VD}(\mathcal{L}, \mathcal{Q}) = \bigcup_{u \in \mathcal{L} \cup \mathcal{Q}} \{\mathcal{VC}(u)\} \tag{2}$$

Remark 1. The most of the computational algorithms (e.g., "Divide and Conquer" [26], Fortune's algorithm [27]) represent the boundaries between neighboring Voronoi cells in terms of the Voronoi graph [28, 29] $G_{\mathcal{S}} = (V_{\mathcal{S}}, E_{\mathcal{S}})$ with a set of the Voronoi vertices $V_{\mathcal{S}}$ and a set of Voronoi edges $E_{\mathcal{S}} \subseteq V_{\mathcal{S}} \times V_{\mathcal{S}}$.

Definition 3. Let's assume that a polygon $\mathcal{P}$ approximates boundary of geometrical object and a vertex p of $\mathcal{P}$ corresponds to the point of the boundary, where object is G_0-continouous (but not G_1-continouous). Then vertex p is called *critical points* (vertices) of the polygon $\mathcal{P}$.

Remark 2. Vertices of polygon $\mathcal{P}$ corresponding to G_1-continouous part of object's boundary, might induce redundant edges of the Voronoi diagram – the bisectors between consecutive line segments l_i and l_{i+1} sharing common non-critical endpoint p_i. In order to obtain an approximate Voronoi diagram of an object represented by $\mathcal{S}$, such redundant edges corresponding to all non-critical points of $\mathcal{S}$ should be removed [29].

Definition 4. An *approximate Voronoi diagram* $\mathcal{VD}_a(\mathcal{S})$ [29] for a planar object represented by a set of polygons $\mathcal{S}$ is obtained as a subgraph $G_{\mathcal{S}}^a$ of the Voronoi graph $G_{\mathcal{S}}$ by removing the edges of $G_{\mathcal{S}}$ corresponding to the bisectors between two consecutive line segments l_i and l_{i+1} sharing a common non-critical vertex p_i.

Definition 5. The *Voronoi skeleton* [23] of a planar object represented by $\mathcal{S}$ is a subset of the approximate Voronoi diagram $\mathcal{VD}_a(\mathcal{S})$ located inside object's region R.

Remark 3. Thus, the Voronoi skeleton of $\mathcal{S}$ is obtained by removing (or trimming) the edges of $G_{\mathcal{S}}^a$, which do not locate in R.

Problem Statement: Given a set of polygons $\mathcal{S}$, which represent a planar object, construct the Voronoi skeleton of $\mathcal{S}$.

3 Algorithm

In this section, we describe the algorithm for computing the Voronoi skeleton. In Subsect. 3.2 we show an algorithm for transforming the Voronoi graph $G_{\mathcal{S}}$ into the final Voronoi skeleton. The complexity analysis of the algorithm is shown in Subsect. 3.3.

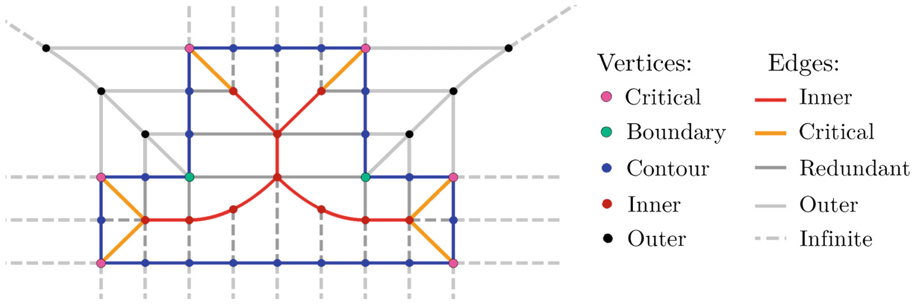


Fig. 1. Examples of the labeled Voronoi vertices and edges.

3.1 Algorithm Description

Input: $\mathcal{S} := \{\mathcal{P}^1, \mathcal{P}^2, \ldots, \mathcal{P}^m\}$ – the set of polygons, each vertex p_i^k of the polygon $\mathcal{P}^k$ has a binary attribute $\text{isCritical}[p_i^k] \in \{\text{True}, \text{False}\}$. Polygon $\mathcal{P}^k$ is oriented such that its interior of the object is to the right for any its line segment (LS) $l \in \mathcal{P}^k$.
Algorithm:

1. Compute Voronoi diagram of line segments $\mathcal{L}$ (with endpoints $\mathcal{Q}$) $\Rightarrow$ Obtain Voronoi graph $G_{\mathcal{S}} = (V_{\mathcal{S}}, E_{\mathcal{S}})$ represented as doubly-connected edge list (DCEL) [28];
2. Using the breadth-first search (BFS) algorithm to traverse the Voronoi graph $G_{\mathcal{S}}$ and label its edges and vertices (see Subsect. 3.2);
3. Remove the edges $G_{\mathcal{S}}$ with labels "R" or "O" and vertices with labels "B" or "O";
4. Remove isolated vertices of $G_{\mathcal{S}}$ if any exist;

3.2 Labeling Voronoi Graph

We traverse the edges and vertices of the Voronoi graph $G_{\mathcal{S}}$ and label them according to their role in a resulting graph of the Voronoi skeleton.

Listing 1. Pseudocode of the Voronoi graph labeling step with auxiliary functions

```
function InitQueue(Q)
begin
  Q := EmptyQueue();
  for edge e
    Label[e] = “None”;
    if isInfinite(e)
      EnQueue(e, Q);
      v : = non-null vertex of e;
      Label[v] := “None”;
    else
      $v_1$, $v_2$ := non-null vertices of e;
      Label[$v_1$] := Label[$v_2$] := “None”;
  return Q;
end;

procedure TraverseBFS(Q)
begin
  while not Empty(Q) do
    e := DeQueue(Q);
    v := Null;
    if (isInfinite(e)) then
      v := non-null vertex of e;
      LabelInfiniteEdge(e);
    else
      v := vertex of e with “None” label;
      LabelFiniteEdge(e);
    for edge e incident to v do
      // Add non-labeled edges to queue
      if (Label[e] = “None”) then
        EnQueue(e, Q);
end;

procedure LabelInfiniteEdge(e)
begin
  $c_1$, $c_2$ := cells of e and Twin(e);
  v := non-null vertex of e;
  if Type($c_1$) = “EP” && Type($c_2$) = “EP” then
    Label[e] := Label[v] := “O”; // Outer
  else
    p := unique EP of line segment;
  if v coincides with p then
    Label[v] := (isCritical[p]) ? “C” : “B”;
    Label[e] := “O”; // Outer
  else
    Label[v] := “I”; // Inner
    if isCritical[p] then
      Label[e] := “C”; // Critical
      Trim e to p;
    else
      Label[e] := “R”; // Redundant
end;

procedure LabelFiniteEdge(e)
begin
  $v_0$ := labeled vertex of e;
  $v_1$ := unlabeled vertex of e;
  $c_1$, $c_2$ := cells of e and Twin(e);
  if Label[$v_0$] = “I” || Label[$v_0$] = “O” then
    if Type($c_1$) = “LS” && Type($c_2$) = “LS” then
      if LS’ of $c_1$ and $c_2$ share endpoint p then
        if isCritical[p] then
          Label[$v_1$] := “C”; // Critical
          Label[e] := (Label[$v_0$]=“I”) ? “C” : “O”;
        else
          Label[$v_1$] := “B”; // Boundary
          Label[e] := (Label[$v_0$]=“I”) ? “R” : “O”;
      else
        Label[$v_1$] := Label[$v_0$];
        Label[e] := Label[$v_0$];
    else // Edge between LS and EP
      if $c_1$ and $c_2$ belong to the same LS then
        p := line segment’s endpoint;
        if p coincides with $v_1$ then
          Label[$v_1$] := (isCritical[p]) ? “C” : “B”;
          if Label[$v_0$] = “O” then
            Label[e] := “O”; // Outer
          else
            Label[e] := (isCritical[p]) ? “C” : “R”;
        else
          Add new vertex v with position p to $G_S$
          Replace e by $e_0$ := (v, $v_0$), $e_1$ := (v, $v_1$);
          Label[v] := (isCritical[p]) ? “C” : “B”;
          if Label[$v_0$] = “O” then
            Label[$v_1$] := “I”;
            if isCritical[p] then
              Label[$e_1$] := “C”;
            else
              Label[$e_1$] := “R”;
              Label[$e_0$]  := “O”;
          else
            Label[$v_1$] := Label[$e_1$] := “O”;
            Label[$e_0$] := (isCritical[p]) ? “C” : “R”;
      else // bisector is a parabolic arc
        Label[$v_1$] := Label[$v_0$];
        Label[e] := (Label[$v_0$]=“O”)?“O”:“I”;
  else // Critical or Boundary
    if $v_1$ is located to the right of $c_1$ or $c_2$ then
      Label[$v_1$] := “Inner”;
      Label[e] := (Label[$v_0$] = “C”) ? “C” : “R”;
    else
      Label[$v_1$] := “Outer”;
      Label[e] := (Label[$v_0$] = “C”) ? “C” : “O”;
end;
```

Definition 6. Voronoi vertex (cf., Fig. 1) is called (label abbreviation is in parenthesis):

- *Inner* (“I”), if the vertex is located inside the object’s polygon;
- *Outer* (“O”), if the vertex is located outside the object’s polygon;
- *Critical* (“C”), if it coincides with one of the critical vertices of the object’s polygon;
- *Boundary* (“B”), if it coincides with one of the non-critical vertices of the polygon;

Definition 7. Voronoi edge (cf., Fig. 1) e is called (label abbreviation is in parenthesis):

- *Inner* ("I"), if it locates in R and doesn't touch (intersect) any polygon from $\mathcal{S}$; $\Rightarrow$ both vertices of e are labeled as *Inner*;
- *Critical* ("C"), if it locates in R and adjacent to a critical vertex;
- *Outer* ("O"), if the edge locates outside $R \Rightarrow$ both vertices of e are labeled as *Outer*;
- *Redundant* ("R"), if it locates in R and touches polygon's non-critical vertex;

The pseudocode illustrating the labeling procedure and the related functions is shown in Listing 1. Firstly, we initialize the queue `Q` of the breadth-first search (BFS) algorithm (see function `InitQueue(Q)`) by all infinite edges of the Voronoi graph G_S. A common data structure `Label[•]` is used to store labels of Voronoi edges and vertices according to the definitions 6 and 7. Then, starting from infinite edges we label all remaining edges and vertices of G_S in function `TraverseBFS(Q)`. At each iteration of BFS algorithm we label current edge and the adjacent non-labeled vertex. In Listing 1 `(Condition)?Value1:Value2` denotes to the ternary conditional operator.

3.3 Complexity Analysis

Lemma 1. The complexity of Step 1 of the skeletonizing algorithm is $O(N \log N)$, where N is a number of points in a polygon.

Proof. At the Step 1 we construct the Voronoi diagram for polygon's line segments using Fortune's algorithm. According to [27] the complexity of this step is $O(M \log M)$, M - number of line segments. Since $N \sim M$, Step 1 has complexity $O(N \log N)$. ■

Lemma 2. The complexity of Step 2 of the skeletonizing algorithm is O(N), where N is a number of the points in an input polygon.

Proof. Step 2 is about labeling the edges and vertices of the Voronoi graph using BFS traverse algorithm. Note that the Voronoi graph is a planar connected graph. Therefore, Euler's formula $|V| - |E| + f = 2$ take place, where $|V|$, $|E|, f$ is a number of vertices, edges and faces of a graph. If $|V| = N$, then the number of edges $|E| = O(N)$. The BFS algorithm traverses all edges of the Voronoi graph. Since all operations within one BFS iteration can be performed in O(1), the complexity of BFS routine is $O(|E| + |V|) =$ O(N). Thus, the complexity of Step 2 is O(N). ■

Lemma 3. The complexity of Steps 3–4 of the skeletonizing algorithm is O(N), where N is a number of the points in an input polygon.

Proof. One edge can be removed from DCEL in O(1) by reassigning the pointers [25, 28]. According to Lemma 2, the number of edges $|E| = O(N)$. Therefore, the complexity of Step 3 is $O(N)$. A single isolated vertex can be removed from DCEL in O(1). Therefore, the complexity of Step 4 is $O(N)$. ■

Theorem 1. The complexity of the skeletonizing algorithm is O(N log N), where N is a number of the points in an input polygon.

Proof. According to analysis of the complexities of each algorithm's step provided in Lemmas 1–3, the total complexity of skeletonizing algorithm is O(N log N). ■

4 Optimization and Heuristics

We introduce an optimization heuristic allowing us to compute fast the Voronoi skeleton by reducing the number of vertices of input polygons. The main idea behind the optimization procedure is illustrated by the following lemma.

Lemma 4. Let $\mathcal{P} = \{p_1, p_2, \ldots, p_N\}$ be a polygon and l_i denotes the line segment between points p_i and p_{i+1} of a polygon $\mathcal{P}$, $i = 1, \ldots, N$, $(p_{N+1} = p_0)$. The polygon $\mathcal{P}'$ is obtained by subdividing line segments $l_i, i = 1, \ldots N$ of a polygon $\mathcal{P}$ such that line segment l_i is replaced by a polyline $p_{i,1}, p_{i,2} \ldots, p_{i,R_i}$ of points on l_i, $i = 1, 2, \ldots, N$ $(p_{i,1} = p_i,\ p_{i,R_i} = p_{i+1})$. Then the Voronoi skeletons $\mathcal{VS}(\mathcal{P})$ and $\mathcal{VS}(\mathcal{P}')$ constructed using the skeletonizing algorithm above are equal (in terms of the Hausdorff distance between the corresponding Voronoi graphs).

Proof. The Voronoi diagram of line segments of $\mathcal{P}$ and $\mathcal{P}'$ consist of the bisectors of the following types: a bisector between two line segment's interiors, a bisector between a line segment's interior and an endpoint, bisector between two endpoints. Let's consider these cases separately:

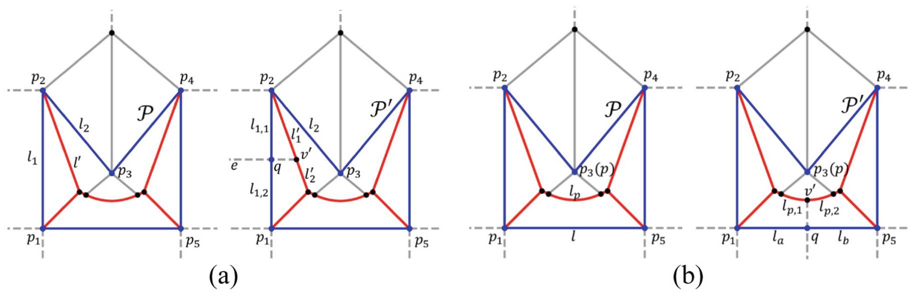


Fig. 2. Illustration of the proof of Lemma 4

Case 1 (see Fig. 2a). The bisector between two line segment's interiors l_1 and l_2 is a line segment l' [27, 28]. Let's suppose that in $\mathcal{P}'$ line segment l_2 remains the same and l_1 is subdivided into two parts $l_{1,1}$ and $l_{1,2}$ connected by a shared endpoint q. Then, the Voronoi cell corresponding to l_1 in $\mathcal{VD}(\mathcal{P})$ will be split into two Voronoi cells (corresponding $l_{1,1}$ and $l_{1,2}$) of $\mathcal{VD}(\mathcal{P}')$ by the Voronoi edge e such that e is a bisector between $l_{1,1}$ and $l_{1,2}$ which passes through q and is perpendicular to l_1 (and therefore, $l_{1,1}$ and $l_{1,2}$). Thus, the Voronoi edge e will divide bisector line segment l' in $\mathcal{VD}(\mathcal{P})$ into two parts l'_1 and l'_2 in $\mathcal{VD}(\mathcal{P}')$ such that l'_1 is a Voronoi edge of the Voronoi cell of

$l_{1,1}$ and l'_2 is a Voronoi edge of the Voronoi cell of $l_{1,2}$. Note that l'_1, l'_2 and edge e are connected together by a newly introduced Voronoi vertex v'. The remaining part of the Voronoi diagrams for $\mathcal{P}'$ and $\mathcal{P}$ stays the same. The BFS labeling procedure (see Step 2 of the algorithm above) for Voronoi edges and vertices of $\mathcal{VD}(\mathcal{P}')$ will split the introduced in $\mathcal{VD}(\mathcal{P}')$ Voronoi edge e into two parts e_1 and e_2: one part will be labeled as "Outer" and the other part will be labeled as "Redundant". Therefore, both parts will be removed at Step 3 of the skeletonizing algorithm and the resulting Voronoi skeleton $\mathcal{VS}(\mathcal{P}')$ will contain the line segment edges l'_1, l'_2 connected by v'.

Case 2. In case of a line segment's interior l and an endpoint p, two possible scenarios take place. First scenario is when p is an endpoint of l. In this case Voronoi diagram contains an edge e' coming through p and perpendicular l. The edge e' can be either removed or not by BFS procedure depending on the type of p. Subdividing l into two parts l_a and l_b which share an endpoint q will introduce a new edge e parallel to e', which will be classifies as "Redundant" and removed from the final skeleton. The second scenario (see Fig. 2b) is when p is not an endpoint of l. Then the bisector between p and l is a parabolic arc l_p, which is subdivided into two parts $l_{p,1}$, $l_{p,2}$ if we split l into l_a and l_b. The analysis in this case is the similar to the Case 1 except that now l'_1 and l'_2 are parabolic arcs $l_{p,1}$ and $l_{p,2}$, respectively.

Case 3. The bisector between two different endpoints of $\mathcal{VD}(\mathcal{P}')$ or $\mathcal{VD}(\mathcal{P})$ is an infinite edge (ray), which is classified at Step 2 of the algorithm above as "Outer" and, therefore, removed from both $\mathcal{VS}(\mathcal{P})$ and $\mathcal{VS}(\mathcal{P}')$ at Step 3.

The case of single subdivision $(L = 1)$ of polygon's line segment for different possible bisectors of the Voronoi diagram is covered above. The general case for several subdivisions L can be proved by induction on L.

Let's assume that for $L = n$ subdivisions of $\mathcal{P}$ holds that $\mathcal{VS}(\mathcal{P})$ and $\mathcal{VS}(\mathcal{P}')$ are equal. The polygon $\mathcal{P}''$ is obtained from $\mathcal{P}'$ by subdividing an arbitrary line segment of $\mathcal{P}'$ into two line segments. Therefore, we can apply one of the proved cases for a single subdivision above and obtain that Voronoi skeletons $\mathcal{VS}(\mathcal{P})$ and $\mathcal{VS}(\mathcal{P}'')$ are equal. Thus, by induction $\mathcal{VS}(\mathcal{P})$ and $\mathcal{VS}(\mathcal{P}')$ are equal for any $L > 0$. ■

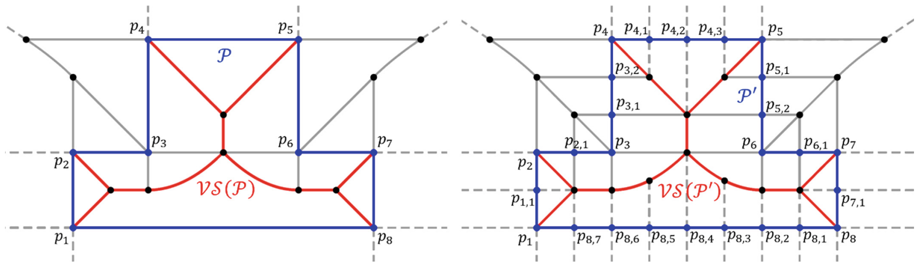


Fig. 3. The Voronoi skeletons (red) for polygon P (blue) and its subdivided version P' (blue) and respective Voronoi diagrams (gray). (Color figure online)

Remark. It follows from Lemma 4 that the Voronoi skeleton $\mathcal{VS}(\mathcal{P}')$ for a subdivided polygon $\mathcal{P}'$ is the same (w.r.t. Hausdorff distance) as the Voronoi skeleton $\mathcal{VS}(\mathcal{P})$ for the original polygon $\mathcal{P}$ (see Fig. 3). However, in comparison to $\mathcal{VS}(\mathcal{P})$, $\mathcal{VS}(\mathcal{P}')$ is represented with a larger number of Voronoi edges and vertices. Therefore, the concept of the Voronoi skeleton with a minimal number of vertices/edges take place. Applying Lemma 4 in the reverse direction allows us to reduce the number of vertices and edges of the Voronoi skeleton. This in turn reduces the execution time of skeletonization algorithm and compresses the resulting graph representation of a skeleton.

Therefore, our aim is to design a heuristic based on simplification operation (reverse to subdivision) and obtain polygon $\mathcal{P}$ from $\mathcal{P}'$. According to Lemma 4 simplification procedure (algorithm) should meet the following requirement:

Simplification Requirement (SR): The polygon simplification heuristic removes the points corresponding to colinear connected line segments of the polygon representing such line segments by a single line segment.

Table 1. The overview of polygon (polyline) simplification algorithms.

Name of algorithm (Abbr.)	Average complexity	Worst-case complexity	SR
Ramer-Douglas-Peucker (DP) [30]	$O(N \log N)$	$O(N^2)$	Yes
Visvalingam-Whyatt (VW) [31]	$O(N \log N)$	$O(N \log N)$	Yes
Reumann-Witkam (RW) [32]	$O(N)$	$O(N)$	Yes
Opheim (OP) [33]	$O(N)$	$O(N)$	Yes
Lang (LA) [34]	$O(NK)$	$O(NK^2)$	Yes
Zhao-Saalfeld (ZS) [35]	$\mathrm{O}(N)$	$O(N)$	Yes
Rapso (RA) [36]	$O(N)$	$O(N)$	No
Li-Openshaw (LO) [37]	$O(N)$	$O(N)$	No
Nth point (NP) [38]	$O(N)$	$O(N)$	No
Circle (CI) [38]	$O(N)$	$O(N)$	No
Perpendicular distance (PD) [38]	$O(NK)$	$O(N)$	Yes

Thus, we introduce Step 0 of the skeletonizing algorithm: simplify each polygon of a set $\mathcal{S}$ by reducing the points associated with colinear connected line segments (SR). This operation can be performed using one of the existing polygon simplification algorithms satisfying the simplification requirement (SR).

Table 2. Suitable polygon simplification algorithms, their parameter and heuristics.

Algorithm	Parameter(s)	Heuristics for 2nd parameter
DP	$\varepsilon > 0$ – tolerance parameter;	No
VW	$A > 0$ – minimum triangle area;	No
RW	$\varepsilon > 0$ – distance tolerance;	No
OP	$\varepsilon_{min}, \varepsilon_{max} > 0$ – tolerances;	$\varepsilon_{max} = +\infty$ (large number)
LA	$\varepsilon > 0$ – distance tolerance; $R \in \mathbb{N}$ – size of search region;	$R = \theta \cdot N$, N – number of points; $\theta \in \{0.05, 0.1, 0.2, 0.25, 0.5, 1.0\}$.
ZS	$\varepsilon > 0$ – sector bound error;	No
PD	$\varepsilon > 0$ – distance tolerance; $K \in \mathbb{N}$ – number of repetitions;	$R = \theta \cdot N$, N – number of points; $\theta \in \{0.05, 0.1, 0.2, 0.25, 0.5, 1.0\}$.

Analysis of Simplification Algorithms. We have analyzed the most commonly used algorithms for polygon (polyline) simplification and summarized the results in Table 1.

However, certain simplification strategies do not agree with the simplification requirement (SR). For example, a naive Nth point [38] method merely removes every Nth point from a polygon ignoring its geometry. Circle simplification [38] method groups together points forming spatial clusters based on the distance threshold. Then, a single representative point replaces each such cluster. Li-Openshaw [37] and Rapso [36] algorithms simplify polyline based on spatial pixel (or hexagon-based) grid. These algorithms instead solve the problem of polyline digitization (useful for solving the problem of optimal map rescaling). Therefore, we consider only the algorithms fulfilling SR (see Table 2). Most of the analyzed algorithms have complexity $O(N)$ except DP [30] and VW [31] algorithms with $O(N \log N)$ complexity. In order to select the algorithm, which shows the best performance improvement, has the smallest computational overhead and influences the resulting skeleton the least, we investigated these algorithms empirically as described in the evaluation section.

5 Evaluation

We evaluate the performance of the skeletonization algorithm in terms of the execution time and measure the influence of the introduced heuristics onto the accuracy, execution time of the overall algorithm. We also estimate the computational overheads related to the line simplification algorithms.

Dataset. In order to evaluate the performance of the skeletonization algorithm and individual optimization heuristics, we used polygons obtained from MPEG 7 CE-Shape-1 dataset. These polygons were extracted from binary images using the Marching Squares algorithm [19]. In total the dataset consists of 1282 polygons (see Fig. 4).

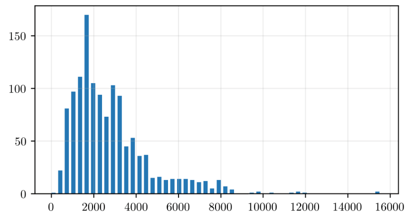


Fig. 4. Distribution of polygon's sizes

Measures. We have measured the following quantities:

1. *Execution time* (ms) of each simplification algorithm, skeletonizing algorithm with (without) the mentioned heuristics and overall execution time. The experiments were carried on Intel Core i7, 2.2 GHz, 16 Gb RAM.
2. *Hausdorff distances* d_H (errors) [39] between the simplified and original polygons and also between the ground truth skeleton and one obtained using the skeletonization with heuristics;
3. *Simplification rate* (%) of the polygon is computed as follows:

$$SR(P, P') = \frac{|P| - |P'|}{|P|} \cdot 100\% \tag{3}$$

where P and P' are original and simplified polygons, respectively. $|P|$ is the number of vertices of P (large values of $SR(P, P')$ correspond to small $|P'|$ w.r.t. $|P|$).

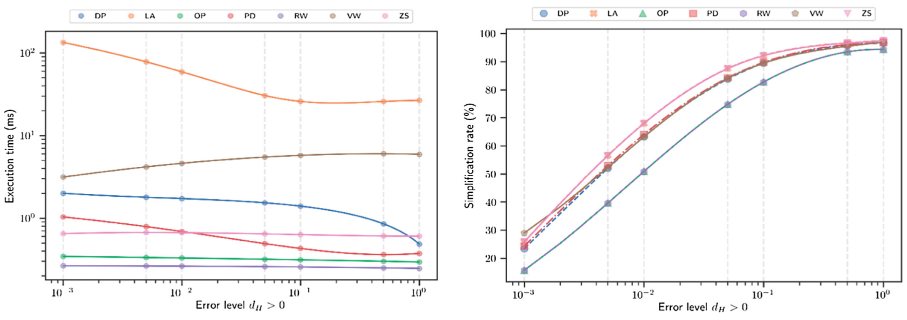
Parameters. The parameters of the simplification algorithms (see Table 2) were chosen using the line search method such that the maximum simplification rate is achieved for a given threshold value of Hausdorff distance d_H between the simplified polygon P' and an original polygon P. This allows us to compare different simplification algorithm with respect to the maximum tolerable error. The established parameters of the simplification algorithms for the respective values of d_H are shown in Table 3.

For the algorithms with two parameters we applied additional heuristics to choose the value of the second parameter (see Table 2). These heuristics were devised to achieve the maximum simplification rate for a given Hausdorff error threshold d_H. It was established that for LA and PD algorithms the optimal value of θ is 0.25 (for $\theta > 0.25$ the simplification rate does not increase, but the execution time of these simplification algorithms rises).

Table 3. Parameters of the simplification algorithms.

Hausdorff distance d_H	Algorithm parameters						
	DP	VW	RW	OP	LA (0.25)	ZS	PD (0.25)
0.001	0.001	0.0007	0.001	0.001	0.001	0.001	0.001
0.005	0.005	0.0025	0.005	0.005	0.005	0.005	0.005
0.01	0.01	0.005	0.009	0.009	0.01	0.01	0.01
0.05	0.05	0.025	0.04	0.04	0.05	0.05	0.05
0.1	0.1	0.05	0.08	0.08	0.1	0.1	0.1
0.5	0.5	0.25	0.4	0.4	0.5	0.5	0.5
1.0	1	0.5	0.8	0.8	1	1	1

Evaluation Results. We have measured the execution time of each suitable simplification algorithm for fixed values of Hausdorff error thresholds d_H (see Fig. 5a). These measurements show the computational overheads related to the optimization step of the skeleton algorithm. In order to compare the quality of the simplification algorithms, we measured the respective simplification rates for given values of d_H.



(a) Average execution time (ms)

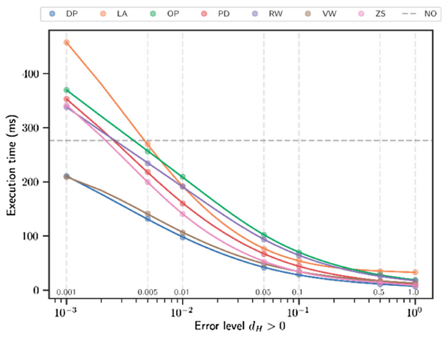
(b) Simplification rates (%), RW≈OP

Fig. 5. Execution time (ms), simplification rates (%) of optimization heuristics

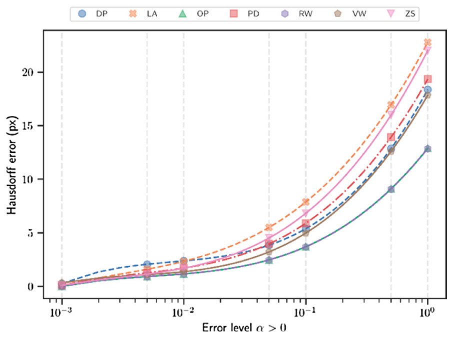
Figure 5b shows that the algorithms of LA and ZS have the most substantial extent of polygon simplification (compression) for a given d_H having nearly identical dependency curves. PD, VW, and PD algorithms achieve slightly smaller simplification rates showing almost undistinguishable behavior for most of the cases. However, VW algorithm overperforms other algorithms for small values of $d_H < 0.002$. OP and RW algorithms have the lowest simplification rates with nearly identical dependency curves.

In Fig. 5 one notices that despite being the fastest, algorithms of OP and RW have the smallest simplification rate and, therefore, might not guarantee the fastest execution of the skeletonization algorithm. Therefore, we measured the total execution time of the skeletonization algorithm depending on the value of d_H taking into account the overhead time of the simplification heuristics (see Fig. 6a).

Based on Fig. 6a we can choose the fastest optimization heuristics. However, different values of d_H threshold might affect the accuracy of the final skeleton. Therefore, we investigated the influence of d_H on the result of the skeletonization algorithm. We calculated the skeletonization error as Hausdorff distance between the ground truth skeleton and the result of optimized skeletonization algorithm (see Fig. 6b).



(a) Total execution time (ms)



(b) Skeletonization error (Hausdorff), RW≈OP

Fig. 6. Total average execution time (ms) and skeletonization errors;

6 Discussion

Figure 6a shows that DP- and VW-based heuristics reduce the computational time to the greatest extent. Only these two heuristics overperform the optimization-free approach (NO) for small values of $d_H \leq 0.001$. The optimization based on OP and RW algorithms shows the smallest skeletonization error among the other approaches (see Fig. 6b). However, for $d_H < 0.002$ these algorithms have outsized computational overheads eliminating the whole effect of the optimization. Therefore, it is reasonable to use OP and RW algorithm only for $d_H > 0.002$. Note that the variance of skeletonization error for different heuristics decreases as $d_H \to 0$ (see Fig. 6b).

We computed 2-sample t-test to validate the hypothesis that DP- and VW-based optimizations produce different average skeletonization errors. The test showed that the errors produced by DP and VW optimizations are undistinguishable (p-value $\approx$ 0.24).

Another hypothesis testing was performed to distinguish the execution time between DP and VW heuristics. It showed that for the most of the cases (except $d_H = 0.001$) DP algorithm overperforms VW (p-value <0.001).

Speed-Accuracy Trade-Off. Figure 6 shows that none of the tested algorithms minimizes the accuracy and execution time of the skeletonizing method at the same time. Therefore, the choice of the heuristics is a trade-off between accuracy and the execution time. Based on the performed computational experiments the following conclusions are drawn:

1. If accuracy of the resulting skeleton is critical, then for $d_H > 0.002$ the optimization can be performed using OP or RW algorithms. However, for $d_H < 0.002$ the only reasonable optimization is using the DP or VW algorithms;
2. If execution time of the algorithm is more critical than the accuracy, then optimization can be performed using DP or VW algorithms, which according to the provided experiments give 1.7 times less accurate result then RW and OP heuristics;

Pruning Effect of Polygon Simplification. It was experimentally discovered, that the introduced optimization heuristics influences the skeleton in a similar way as pruning methods [40]. Figure 7 shows that for large values of d_H (see bottom row) simplification heuristics tends to regularize shape of the object in a way that the branches of the skeleton corresponding to small shape perturbation disappear (cf., Fig. 7, top row). Therefore, such optimization allows us not only to speed-up the execution of the skeletonization, but also to achieve a pruning effect and remove the noisy branches of the skeleton.

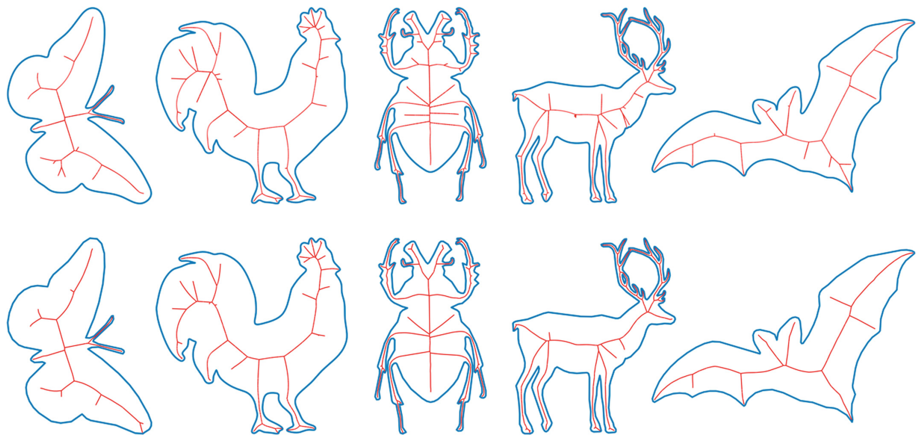


Fig. 7. Examples of the optimized Voronoi skeletons for shapes from MPEG 7 CE-Shape-1 dataset. Optimization heuristics is DP. For $d_H = 0.001$ (top row of images) skeletons contain redundant branches in comparison to $d_H = 1.0$ (the bottom row of images).

7 Conclusion

We proposed optimization heuristics for computing the Voronoi skeleton of the polygonal data. This topic is of relevance due to its direct relation to efficient processing of the vectorized geometrical representations (e.g., for image processing, computer vision, computer graphics). We illustrated in detail the main steps of the Voronoi-based skeletonization algorithm and determined that its complexity is $O(N \log N)$, where N is the number of vertices in a polygon. We also established an optimization criterion (requirement) and proposed theoretically justified optimization heuristics based on the polygon simplification algorithms. In order to evaluate the efficiency of the proposed heuristic, a series of computational experiments were conducted using the polygons from MPEG 7 CE-Shape-1 dataset. Seven state-of-the-art simplification algorithms were evaluated to determine the most suitable optimization heuristic fulfilling the established criterion. We measured the execution time of the skeletonization algorithm with and without the heuristic optimizations and determined the computational overheads related to such heuristics. We also determined the accuracy of the skeleton produced by the optimized algorithm based on the proposed heuristics. As a result, we established the criteria, which allow us to choose the optimal heuristics depending on the system's requirement. For example, DP- and VW-based heuristics allow us to speed up the skeleton computation at least by 30%. It was discovered experimentally, that the optimization heuristics have a pruning effect onto the resulting skeleton.

Acknowledgements. The research reported in this paper has been partly supported by the Austrian Ministry for Transport, Innovation and Technology, the Federal Ministry of Science, Research and Economy, and the Province of Upper Austria in the frame of the COMET center SCCH.

References

1. Saha, P.K., Borgefors, G., Sanniti di Baja, G.: A survey on skeletonization algorithms and their applications. Pattern Recognit. Lett. **76**, 3–12 (2016)
2. Sundar, H., Silver, D., Gagvani, N., Dickinson, S.: Skeleton based shape matching and retrieval. In: 2003 Shape Modeling International, Seoul, pp. 130–139. IEEE (2003)
3. Xie, J., Heng, P.-A., Shah, M.: Shape matching and modeling using skeletal context. Pattern Recognit. **41**, 1756–1767 (2008)
4. Chaudhuri, A., Mandaviya, K., Badelia, P., Ghosh, S.K.: Optical character recognition systems for different languages with soft computing. SFSC, vol. 352. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-50252-6
5. Torres, R.d.S., Falcão, A.X.: Contour salience descriptors for effective image retrieval and analysis. Image Vis. Comput. **25**, 3–13 (2007)
6. Rezaee, K., Haddadnia, J., Tashk, A.: Optimized clinical segmentation of retinal blood vessels by using combination of adaptive filtering, fuzzy entropy and skeletonization. Appl. Soft Comput. **52**, 937–951 (2017)

7. Lasso, W., Morales, Y., Torres, C.: Image segmentation blood vessel of retinal using conventional filters, Gabor transform and skeletonization. In: 2014 XIX Symposium on Image, Signal Processing and Artificial Vision, Columbia. IEEE (2014)
8. Al-Kofahi, Y., Dowell-Mesfin, N., Pace, C., Shain, W., Turner, J.N., Roysam, B.: Improved detection of branching points in algorithms for automated neuron tracing from 3D confocal images. Cytom. Part A. **73A**, 36–43 (2008)
9. Faulkner, C., et al.: An automated quantitative image analysis tool for the identification of microtubule patterns in plants. Traffic **18**, 683–693 (2017)
10. Beil, M., Braxmeier, H., Fleischer, F., Schmidt, V., Walther, P.: Quantitative analysis of keratin filament networks in scanning electron microscopy images of cancer cells. J. Microsc. **220**, 84–95 (2005)
11. Changxian, S., Yulong, M.: Morphological thinning based on image's edges. In: ICCT 1998, 1998 International Conference on Communication Technology. Publishing House of Construction Materials, Beijing (1998)
12. Zhang, T.Y., Suen, C.Y.: A fast parallel algorithm for thinning digital patterns. Commun. ACM **27**, 236–239 (1984)
13. Yan, T.-Q., Zhou, C.-X.: A continuous skeletonization method based on distance transform. In: Huang, D.-S., Gupta, P., Zhang, X., Premaratne, P. (eds.) ICIC 2012. CCIS, vol. 304, pp. 251–258. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31837-5_37
14. Chen, J., Du, M., Qin, X., Miao, Y.: An improved topology extraction approach for vectorization of sketchy line drawings. Vis. Comput. **34**, 1633–1644 (2018)
15. Hilaire, X., Tombre, K.: Robust and accurate vectorization of line drawings. IEEE Trans. Pattern Anal. Mach. Intell. **28**(6), 890–904 (2006)
16. Acciai, L., Soda, P., Iannello, G.: Automated neuron tracing methods: an updated account. Neuroinformatics **14**, 353–367 (2016)
17. De, J., et al.: A graph-theoretical approach for tracing filamentary structures in neuronal and retinal images. IEEE Trans. Med. Imaging **35**, 257–272 (2016)
18. Stein, A.M., Vader, D.A., Jawerth, L.M., Weitz, D.A., Sander, L.M.: An algorithm for extracting the network geometry of three-dimensional collagen gels. J. Microsc. **232**, 463–475 (2008)
19. Maple, C.: Geometric design and space planning using the marching squares and marching cube algorithms. In: 2003 International Conference on Geometric Modeling and Graphics, 2003, London. IEEE Computer Society (2003)
20. Aichholzer, O., Aurenhammer, F., Alberts, D., Gärtner, B.: A novel type of skeleton for polygons. In: Maurer, H., Calude, C., Salomaa, A. (eds.) J.UCS The Journal of Universal Computer Science, pp. 752–761. Springer, Heidelberg (1996). https://doi.org/10.1007/978-3-642-80350-5_65
21. Eppstein, D., Erickson, J.: Raising Roofs, crashing cycles, and playing pool: applications of a data structure for finding pairwise interactions. Discret. Comput. Geom. **22**, 569–592 (1999)
22. Chin, F., Snoeyink, J., Wang, C.A.: Finding the medial axis of a simple polygon in linear time. Discret. Comput. Geom. **21**, 405–420 (1999)
23. Ogniewicz, R., Ilg, M.: Voronoi skeletons: theory and applications. In: Proceedings 1992 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, Champaign. IEEE Computer Society Press (1992)
24. Siddiqi, K., Pizer, S.M. (eds.): Medial Representations. Springer, Dordrecht (2008). https://doi.org/10.1007/978-1-4020-8658-8
25. Preparata, F.P., Shamos, M.I.: Computational Geometry. Springer, New York (1985). https://doi.org/10.1007/978-1-4612-1098-6

26. Shamos, M.I., Hoey, D.: Closest-point problems. In: 16th Annual Symposium on Foundations of Computer Science, Berkley. IEEE (1975)
27. Fortune, S.: A sweepline algorithm for Voronoi diagrams. Algorithmica **2**, 153–174 (1987)
28. de Berg, M., Cheong, O., van Kreveld, M., Overmars, M.: Computational Geometry. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-77974-2
29. Okabe, A., Boots, B., Sugihara, K., Chiu, S.N., Kendall, D.G. (eds.): Spatial Tessellations. Wiley, Hoboken (2000)
30. Douglas, D.H., Peucker, T.K.: Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. Cartographica **10**, 112–122 (1973)
31. Visvalingam, M., Whyatt, J.D.: Line generalisation by repeated elimination of points. Cartogr. J. **30**, 46–51 (1993)
32. Reumann, K., Witkam, A. P. M.: Optimizing curve segmentation in computer graphics. In: International Computing Symposium, Elsevier, North Holland, pp. 467–472 (1973)
33. Opheim, H.: Fast data reduction of a digitized curve. Geo-Processing **2**, 33–40 (1982)
34. Lang, T.: Rules for robot draughtsman. Geogr. Mag. **42**, 50–51 (1969)
35. Zhao, Z., Saalfeld, A.: Linear-time sleeve-fitting polyline simplification algorithms. In: Proceedings of the Annual Convention and Exposition. Technical Papers, Seattle, USA, pp. 214–223 (1997)
36. Raposo, P.: Scale-specific automated line simplification by vertex clustering on a hexagonal tessellation. Cartogr. Geogr. Inf. Sci. **40**, 427–443 (2013)
37. Nie, H., Huang, Z.: A new method of line feature generalization based on shape characteristic analysis. Metrol. Meas. Syst. **18**, 597–606 (2011)
38. Song, J., Miao, R.: A novel evaluation approach for line simplification algorithms towards vector map visualization. Int. J. Geo-Inf. **5**, 223 (2016)
39. Taha, A.A., Hanbury, A.: An efficient algorithm for calculating the exact hausdorff distance. IEEE Trans. Pattern Anal. Mach. Intell. **37**, 2153–2163 (2015)
40. Beristain, A., Graña, M., Gonzalez, A.I.: A pruning algorithm for stable Voronoi skeletons. J. Math. Imaging Vis. **42**, 225–237 (2011)