The Visual Computer (2024) 40:6369–6387
https://doi.org/10.1007/s00371-023-03170-9

ORIGINAL ARTICLE

Obtaining the user-defined polygons inside a closed contour with holes

R. Molano[1] · J. C. Sancho[2] · M. M. Ávila[2] · P. G. Rodríguez[2] · A. Caro[2]

Accepted: 9 September 2023 / Published online: 7 December 2023

Abstract
In image processing, computer vision algorithms are applied to regions bounded by closed contours. These contours are often irregular, poorly defined, and contain holes or unavailable areas inside. A common problem in computational geometry includes finding the k-sided polygon (k-gon) of maximum area or maximum perimeter inscribed within a contour. This paper presents a generic method to obtain user-defined polygons within a region. Users can specify the number k of sides of the polygon to obtain. Additionally, users can also decide whether the calculated polygon should be the largest in area or perimeter. This algorithm produces a polygon or set of polygons that can be used to segment an image, allowing only relevant areas to be processed. In a real-world application, the validity and versatility of the proposed method are demonstrated. In addition, the source code developed in Java and Python is available in a GitHub repository so that researchers can use it freely.

Keywords k-Gon · Closed contour · Polygon · Holes · Computational geometry

1 Introduction

The definition of regions of interest within images plays an important role in computer vision. Many computer animation and simulation, image processing, medical imaging, shape modeling and analysis, visual analytics, or scientific visualization systems require geometric modeling and processing to delimit regions of interest. For this purpose, computational geometry algorithms are typically the best option to identify subregions within an image. These computational geometry algorithms typically have high computational complexity and usually produce straightforward polygons (e.g., triangles, rectangles, quadrilaterals) without the real possibility of including constraints specified by the practical needs demanded by the users. For example, one user might be interested in obtaining a polygon with the maximum area, while another might be interested in obtaining a polygon with the maximum perimeter. Alternatively, one user may wish to obtain polygons with three sides (triangles), while another may desire polygons with many more sides (e.g., quadrilaterals, pentagons, hexagons, etc.). In addition, many practical applications are based on obtaining polygons that are inscribed in a given region of interest. In most cases, these regions of interest are straightforward, and it is atypical to consider surfaces containing inaccessible areas (i.e., regions containing one or more holes).

All of these constraints have proven insurmountable to date in computational geometry applied to computer vision, because no method has been published that solves the computational problem of obtaining the simple polygon with any number of sides (k-gon) with maximum area or perimeter within a region of interest with holes.

This problem can be formulated as follows: given a fixed number k, compute the simple k-gon with maximum area

J. C. Sancho, M. M. Ávila, P. G. Rodríguez and A. Caro have contributed equally to this work.

✉ R. Molano
rmolano@unex.es

J. C. Sancho
jcsanchon@unex.es

M. M. Ávila
mmavila@unex.es

P. G. Rodríguez
pablogr@unex.es

A. Caro
andresc@unex.es

[1] Department of Mathematics, Universidad de Extremadura, 10003 Cáceres, Spain

[2] Department of Computer and Telematics Systems Engineering, Universidad de Extremadura, 10003 Cáceres, Spain

or perimeter contained in any closed contour C (region of interest) with h-holes. Figure 1 shows the initial problem, the closed contour C and holes, H_1 and H_2, the inaccessible areas.

1.1 Practical applications

Many practical applications are possible for the algorithm proposed in this paper, including geographic information system (GIS) [1, 2], robotics [3, 4], medical imaging [5–7] or agricultural plots [8, 9]. Thus, we downloaded images from the Ministry of Farming, Fishing and Food of Spain [10] using the application Visor SigPac v4.8 [11], as shown in Fig. 2a. The closed contour C was then created (Fig. 2b), generating a collection of points. Finally, the lattice polygon P (Fig. 2c) that is applied in Algorithm 8 (i.e., the main algorithm) is built, where there are 3 unavailable zones (H_1, H_2, H_3) and $\#(P) = 110$. Then,

$$\begin{cases} points & = P = P_0 - \bigcup_{i=1}^{h}(\iota H_i) = \{p_1, \ldots, p_{110}\} \\ polygon & = \partial P_0 = \{p_1, \ldots, p_{42}\} \\ holes & = \partial H_1 \cup \partial H_2 \cup \partial H_3 = \\ & = \{p_{92}, \ldots, p_{97}\} \cup \{p_{98}, \ldots, p_{104}\} \cup \{p_{105}, \ldots, p_{110}\} \end{cases}$$

Using Algorithm 8, we obtain the solutions shown in Fig. 3, which describes how the area of the simple k-gon increases when the number of sides increases. In addition, the solution to obtain the rectangle with maximum area is included to demonstrate the versatility of the proposed algorithm.

2 Related works

The computation of the maximum area or perimeter within a region of interest is an important optimization problem for computational geometry. This calculation is typically performed on convex or simple polygons but without adding whether it is possible to perform it on a region of interest (closed contour) with or without holes. In all the reviewed papers, the longest perimeter (or the largest area) is understood on a closed contour delimited by a finite set of points.

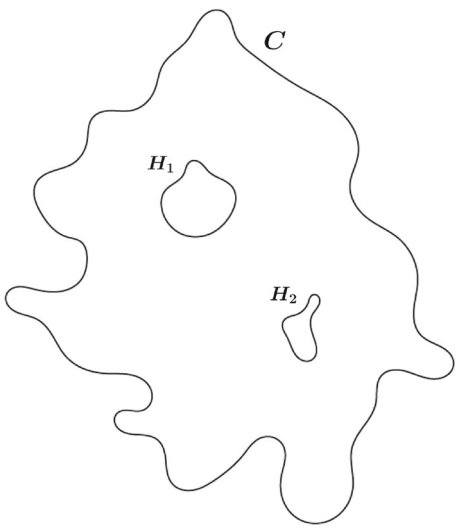


Fig. 1 Closed contour C with two holes H_1 and H_2

Considering area, several authors have proposed different solutions to find the polygon with the largest area contained in a polygon with n vertices and h holes. For triangles, Lee et al. [12] presented an algorithm to find the largest triangle that can be inscribed in a polygon with holes but under various conditions. If the polygon does not have holes, the best solutions were reported by Kallus [13] for convex polygons in $O(n)$ time, and Melissaratos and Souvaine [14] solved the problem for simple polygons in $O(n^4)$ time.

For rectangles, Alt et al. [15] computed the largest area axis-parallel rectangle in a convex polygon in $O(\log n)$ time and Boland and Urrutia [16] solved the problem in $O(n \log n)$ time for a simple polygon. Daniels et al. [17] added the condition of polygons with holes and solved it in $O(n \log^2 n)$. Removing the axis-aligned condition, Kanuer et al. [18] considered approximation algorithms and proved that the rectangle with the largest area in a convex polygon could be computed in $O(\frac{1}{\varepsilon} \log \frac{1}{\varepsilon} \log n)$ time. For simple polygons, Molano et al. [19] solved the problem in $O(n^3)$ time, and if the initial polygon has holes, Choi et al. [20] showed how to solve it in $O(n^3 \log n)$ time. For parallelograms, Jin [21] developed an algorithm to compute the largest parallelogram inside a convex polygon in $O(n \log^2 n)$ time and Molano et al. [22] solved the same problem in $O(n^3)$ time but for simple polygons. None of these methods can manage holes. Finally, Keikha et al. [23] and Rote et al. [24] found an algorithm to compute the quadrilateral of the largest area contained in a convex polygon in $O(n)$ time, without mentioning whether it is operates correctly with holes. Table 1 shows the computational costs of the previous papers.

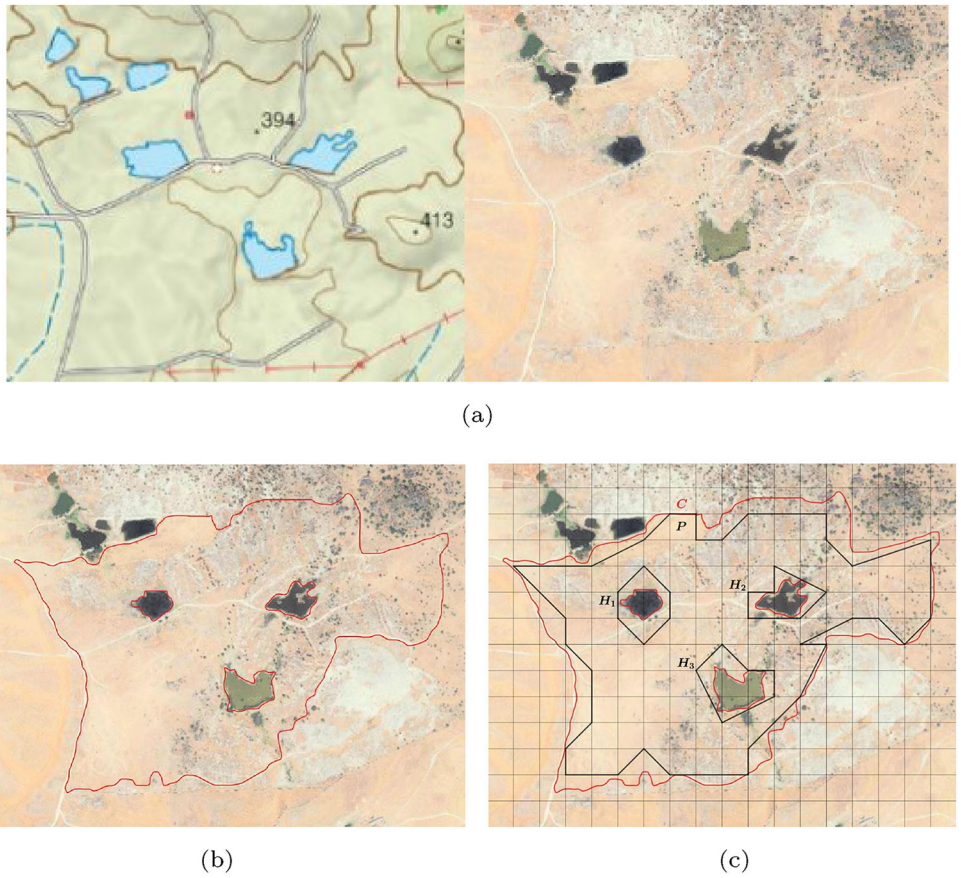
Two conclusions can be deduced from the literature. First, there is a large difference in computational cost when the initial polygon changes from convex to nonconvex. Second, there are few papers that mention polygons with holes.

In general, this type of problem can be reduced to a geometric optimization problem in the class of polygon *inclusion problems* as follows:

$Inc(\mathcal{P}, \mathcal{Q}, \mu)$: Given $P \in \mathcal{P}$, find the μ-largest $Q \in \mathcal{Q}$ that is included in P, where $\mathcal{P}$ and $\mathcal{Q}$ are families of polygons and μ is a real function on polygons such that:

$$\forall Q, Q' \in \mathcal{Q}, \quad Q' \subseteq Q \Rightarrow \mu(Q') \leq \mu(Q)$$

The "potato peeling problem" or the problem of finding the largest convex polygon contained in a simple polygon is the inclusion problem $Inc(\mathcal{P}_{all}, \mathcal{P}_{con}, \mu)$, where $\mathcal{P}_{all}$ is the family of all simple polygons, $\mathcal{P}_{con}$ denotes the class of all



(a)

(b)

(c)

Fig. 2 Image extraction for real practical application

the convex polygons and μ is the real function with respect to area or perimeter. The problem was introduced by Goodman [25] and solved by Chang and Yap [26] in $O(n^7)$ time under the area measure and in $O(n^6)$ time for the perimeter. Later, Cabello et al. [27] reported an approximation algorithm in $O(n(\log^2 n + (1/\epsilon^3)\log n + 1/\epsilon^4))$ time.

The goal of this paper is to develop an algorithm that given a fixed number k, to compute the simple k-gon of a maximum area or perimeter inscribed in any closed contour with holes. That is, to solve the inclusion problem $Inc(\mathcal{P}_{hole}, \mathcal{P}_{k\,sim}, \mu)$, where $\mathcal{P}_{hole}$ is the family of all polygons with holes and $\mathcal{P}_{k\,sim}$ denotes the class of all simple k-gons. Thus, with the proposed algorithm, the user chooses the k-gon to be shown (e.g., triangle, quadrilateral, pentagon, hexagon, etc.) and the type of solution desired maximum area or perimeter. In addition, we have developed the algorithm in pseudocode and defined it in 8 subprograms so that any researcher can use it in future investigations. Additionally, a link to a GitHub repository with all the source code developed in Java and Python is included [28].

The algorithm developed in this paper is described in the following sections. Section 3 introduces the concept of a lattice polygon with holes, and Sect. 4 describes how to calculate the adjacency matrix associated with that polygon. Section 5 explains how to compute the maximum area or perimeter simple k-gon inscribed in a lattice polygon with holes, and Sect. 6 extends the previous section and describes how to compute the maximum area or perimeter simple k-gon inscribed in a closed contour with holes. Section 7 shows where the source code can be downloaded. Finally, Sect. 8 presents the conclusions of this paper.

3 Preliminary

We define a *lattice polygon* as a polygon whose points have integer coordinates and which has the following properties:

1. It is defined on a *regular partition* $\Pi = \Pi_x \times \Pi_y$ of order $r \times s$ formed by $r+1, s+1$ equally spaced points that satisfy:

$$\Pi_x = \{a = x_0 < x_1 < \ldots < x_r = b\}$$
$$\Pi_y = \{c = y_0 < y_1 < \ldots < y_s = d\}$$

where $a, b, c, d \in \mathbb{Z}$.
2. We denote $G_L = \{(x_i, y_j) : 0 \leq i \leq r, 0 \leq j \leq s\}$, the square grid composed of points of the partition Π. We define *partition size*, $L = |x_{i+1} - x_i| = |y_{j+1} - y_j|$, the length of the side of each square formed by the square

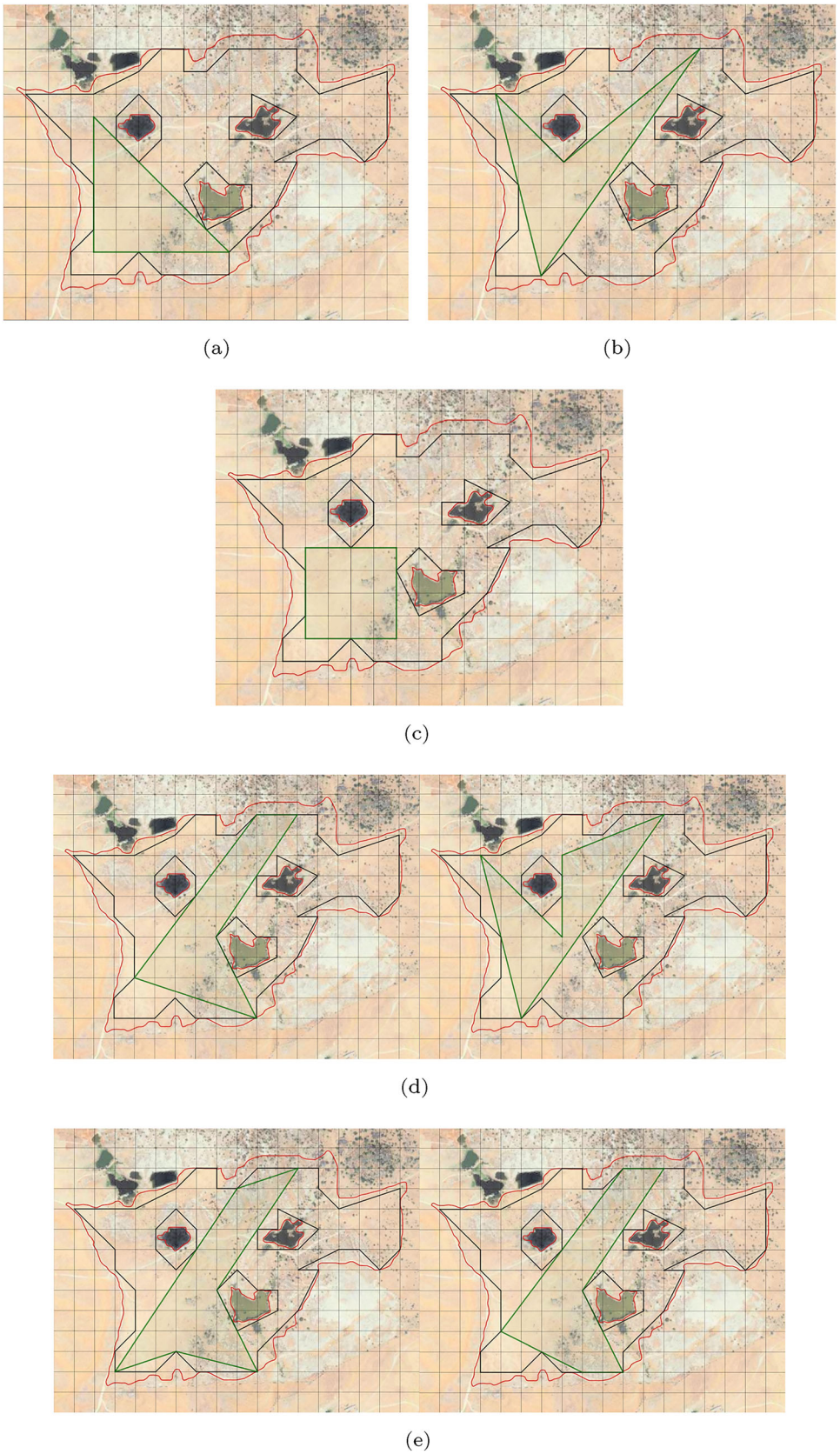


Fig. 3 Practical application solutions

Table 1 Computational cost

Reference	Polygon	k-gon	Computational cost
[12]	Holes	Triangle	$O(n^4)$
[13]	Convex	Triangle	$O(n)$
[14]	Simple	Triangle	$O(n^4)$
[15]	Convex	Axis-parallel rectangle	$O(\log n)$
[16]	Simple	Axis-parallel rectangle	$O(n \log n)$
[17]	Holes	Axis-parallel rectangle	$O(n \log^2 n)$
[18]	Convex	Rectangle	$O(\frac{1}{\varepsilon} \log \frac{1}{\varepsilon} \log n)$
[19]	Simple	Rectangle	$O(n^3)$
[20]	Holes	Rectangle	$O(n^3 \log n)$
[21]	Convex	Parallelogram	$O(n \log^2 n)$
[22]	Simple	Parallelogram	$O(n^3)$
[23, 24]	Convex	Quadrilateral	$O(n)$
Proposed algorithm	Holes	Any	$O(n^5 k)$

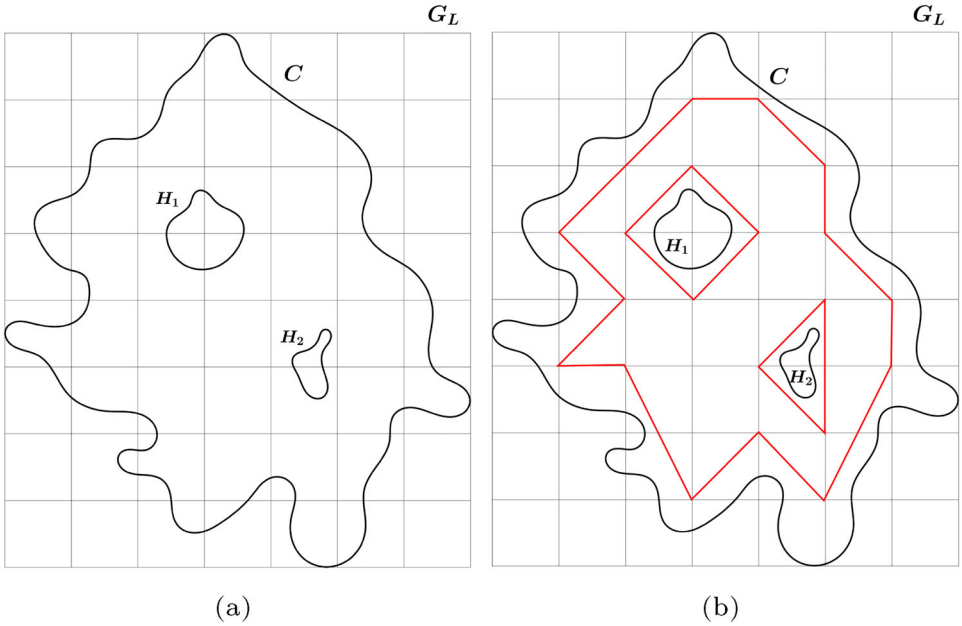


Fig. 4 Construction of the lattice polygon

grid. In addition, we state that partition $\dot{\Pi}$ is finer than partition Π, if it is verified that all points of Π belong to $\dot{\Pi}$. We denote $\Pi \preceq \dot{\Pi}$.

3. The connections between consecutive vertices are not necessarily established in the eight directions, $\pi k/4$, $k = 0, \ldots, 7$.
4. The edges of the polygon do not intersect except at their vertices.

In Fig. 4 shows how to construct a lattice polygon P from the initial problem (Fig. 1). First, we define a partition Π on the closed contour C with partition size L (Fig. 4a), extracting a collection of points (x_i, y_j) inside the closed contour. Then, in (Fig. 4b), we obtain the lattice polygon by joining those points such that we obtain the largest area polygon contained in C, and for the holes, the smallest area polygon containing H_1 and H_2. The lattice polygon P shown in Fig. 5.

If $A(P)$ denotes the area of P, $A(P_0)$ is the area of P_0 (the outer lattice polygon) and $A(H_i)$ is the area of each hole with $1 \leq i \leq h$, then:

$$A(P) = A(P_0) - \sum_{i=1}^{h} A(H_i)$$

By Pick's theorem [29],

$$A(P) = \left(I + \frac{B}{2} + h - 1 \right) \cdot L^2$$

where $I = I_0 - \sum_{i=1}^{h} (I_i + B_i)$ and $B = B_0 + \sum_{i=1}^{h} B_i$ are the number interior (ιP) and boundary (∂P) points of

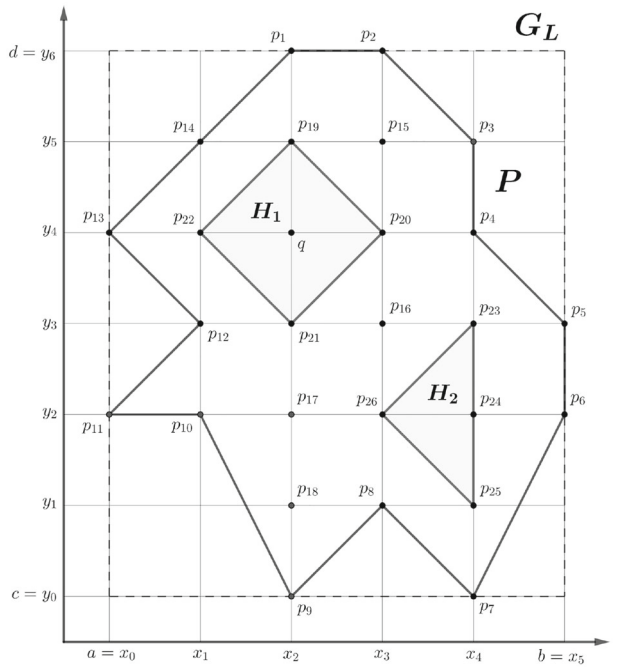


Fig. 5 Lattice polygon P with two holes on a regular partition of order 5×6 with partition size L

P, I_0 and B_0 the number interior ($\imath P_0$) and boundary (∂P_0) points of P_0 and I_i and B_i the number interior ($\imath H_i$) and boundary (∂H_i) points of each hole H_i, $1 \leq i \leq h$.

Then we decompose the lattice polygon P with h holes as follows:

$$\begin{cases} P = \partial P \cup \imath P = \{p_1, p_2, \ldots, p_{n+o}\} \\ H_i = \partial H_i \cup \imath H_i = \{q_1^i, q_2^i, \ldots, q_{m_i+r_i}^i\} \end{cases}$$

where if # represents the cardinality of the set, $\#(\partial P) = n$, $\#(\imath P) = o$, $\#(P) = N = n + o \simeq \lambda n$, $\lambda \in \mathbb{N}$, and $\#(\partial H_i) = m_i$, $\#(\imath H_i) = r_i$, $\#(H_i) = M = m_i + r_i \simeq \mu m_i$, $\mu \in \mathbb{N}$. Furthermore, let $m = \max\{m_1, m_2, \ldots, m_h\}$.

Thus, for Fig. 5:

$$\begin{cases} \partial P = \partial P_0 \cup \partial H_1 \cup \partial H_2 = \{p_1, \ldots, p_{14}, p_{19}, \ldots, p_{22}, p_{23}, \ldots, p_{26}\} \\ \imath P = \imath P_0 - (H_1 \cup H_2) = \{p_{15}, p_{16}, p_{17}, p_{18}\} \\ \partial H_1 = \{p_{19}, p_{20}, p_{21}, p_{22}\} = \{q_1^1, q_2^1, q_3^1, q_4^1\} \\ \imath H_1 = \{q\} = \{q_5^1\} \\ \partial H_2 = \{p_{23}, p_{24}, p_{25}, p_{26}\} = \{q_1^2, q_2^2, q_3^2, q_4^2\} \\ \imath H_2 = \emptyset \end{cases}$$

We compute the maximum area or perimeter simple k-gon in a lattice polygon P with h holes and coordinates belonging to the square grid G_L, and we use the following sets:

$$\begin{cases} points = P = \{p_1, p_2, \ldots, p_{26}\} = P_0 - \bigcup_{i=1}^{h} (\imath H_i) \\ polygon = \partial P_0 = \{p_1, p_2, \ldots, p_{14}\} \\ holes = \partial H_1 \cup \partial H_2 = \{p_{19}, p_{20}, p_{21}, p_{22}\} \cup \{p_{23}, p_{24}, p_{25}, p_{26}\} \\ holes[i] = \partial H_i, 1 \leq i \leq 2 \end{cases}$$

The only points that are not considered are those belonging to $\imath H_i$ with $1 \leq i \leq h$ because it is not possible to calculate a polygon with points from the interior of a hole.

4 Compute of the adjacency matrix of a lattice polygon with holes

Given N points of the lattice polygon P with h holes, the *adjacency matrix* $A = (a_{ij})$ is a symmetric square matrix of order N such that:

$$a_{ij} = \begin{cases} 1 \text{ if there is an edge between } i \text{ and } j \\ 0 \text{ otherwise} \end{cases}$$

Matrix A is required for the aim of the proposed article. For Fig. 5, the adjacency matrix is of order 26; thus, we develop a procedure to calculate matrix A using 5 subprograms that have been presented in pseudocode and defined in three steps:

1. Determine if a point is inside or on a side of *polygon* or a hole of *holes* (Algorithm 1).
2. Determine whether intersection of a segment with *polygon* or *holes* is allowed or not allowed (Algorithms 2, 3 and 4).
3. Compute of the adjacency matrix (Algorithm 5).

4.1 Point in polygon (Algorithm 1)

We compute if a given point (p) is inside or on a side of *polygon* $= \partial P_0$ or a hole of *holes* $= \bigcup_{i=1}^{h} holes[i] = \bigcup_{i=1}^{h} (\partial H_i)$ with Algorithm 1.

We traverses all sides of "poly" in sequence (Lines 5–9), calculating the angle formed by its points and straight lines connecting them to point p. Adding all the angles, two solutions are possible, 0 and $\pm 2\pi$, which determine the position of the point in the polygon: 0 if the point is outside, and 2π if inside (Lines 10–11). In addition, in the path, we can determine if the point is on one of the sides using the functions *DIRECTION (p,q,r: points)*, which returns the orientation of the three points [30], and *ALIGNEDp (p,q,r: points)*, which tells us if the points are aligned. If the values output by these functions are 0 and true, respectively, the algorithm is finished, and point p is on one side of "poly".

The computational cost is determined by Line 5 in $O(n)$ for *polygon* and $O(m)$ for $holes[i]$, where $m = \max\{m_1, m_2, \ldots, m_h\}$. The functions *DIRECTION(p,q,r: points)* and *ALIGNEDp(p,q,r: points)* are computed in $O(1)$ time and the function *ANGLE(u,v: vectors)* is computed in $O(1)$ time, which calculates the angle formed by the vectors $\vec{u}$ and $\vec{v}$.

Algorithm 1: *PointIn (points, p, poly)*

Input: p: point, poly: *polygon* or *holes[i]*, $1 \leq i \leq h$
Output: *true* if point p is inside or on side of poly and *false* if they do not

```
1  point-in ← false; pol ← ∅
2  for i ← 1 to Length(poly) do
3  |  Insert(pol, poly[i])
4  Insert(pol, poly[1])
5  for i ← 1 to Length(pol)-1 do
6  |  point-in ← DIRECTION (points[pol[i]], points[pol[i+1]], p)
   |  = 0 and ALIGNEDp(points[pol[i]], points[pol[i+1]], p) = true
7  |  if point-in = true then
8  |  |  Break
9  |  sum ← sum + ANGLE (points[poly[i]]-p,
   |  points[poly[i+1]]-p)
10 if Abs(sum) = 2π then
11 |  point–in ← true
12 return point–in
```

4.2 Segment-polygon intersection

We compute when the intersection of a segment with *polygon* or *holes* is allowed or not allowed with Algorithm 2 in $O(1)$ time; Algorithm 3 in $O(n)$ time for *polygon* and $O(m^2)$ time for *holes[i]*; and Algorithm 4 in $O(n^2)$ time.

4.2.1 Intersection point of segments (Algorithm 2)

We now compute the intersection point of the nonconsecutive segments l_1 and l_2 by considering three cases according to their relative position (Fig. 6).

Let $l_1 = \{l_1[1], l_1[2]\}$ with $l_1[1] = (l_1[1][1], l_1[1][2]) = (x_1, y_1)$ and $l_1[2] = (l_1[2][1], l_1[2][2]) = (x_2, y_2)$. Then, if $\vec{u_1} = (x_2 - x_1, y_2 - y_1)$ is the director vector of the line r passing through l_1, r can be written as follows:

$$\frac{x - x_1}{x_2 - x_1} = \frac{y - y_1}{y_2 - y_1} \Rightarrow (y_2 - y_1) \cdot (x - x_1) = (x_2 - x_1) \cdot (y - y_1)$$
$$\Rightarrow (y_2 - y_1)x + (x_1 - x_2)y = x_1 \cdot y_2 - x_2 \cdot y_1$$

Therefore, $r \equiv Ax + By = C$, where:

$$\begin{cases} A = y_2 - y_1 = l_1[2][2] - l_1[1][2] \\ B = x_1 - x_2 = l_1[1][1] - l_1[2][1] \\ C = x_1 \cdot y_2 - x_2 \cdot y_1 = l_1[1][1] \cdot l_1[2][2] - l_1[2][1] \cdot l_1[1][2] \end{cases}$$

Doing the same for side $l_2 = \{l_2[1], l_2[2]\}$ with $l_2[1] = (l_2[1][1], l_2[1][2]) = (x_3, y_3)$, $l_2[2] = (l_2[2][1], l_2[2][2]) = (x_4, y_4)$ and $\vec{u_2} = (x_4 - x_3, y_4 - y_3)$ as director vector, then the line s is of the form $s \equiv Dx + Ey = F$, where:

$$\begin{cases} D = y_4 - y_3 = l_2[2][2] - l_2[1][2] \\ E = x_3 - x_4 = l_2[1][1] - l_2[2][1] \\ F = x_3 \cdot y_4 - x_4 \cdot y_3 = l_2[1][1] \cdot l_2[2][2] - l_2[2][1] \cdot l_2[1][2] \end{cases}$$

To calculate the point of intersection of segments l_1 and l_2 the following system of linear equations must be solved:

$$\left.\begin{matrix} Ax + By = C \\ Dx + Ey = F \end{matrix}\right\}$$

If the straight lines intersect at a point (p), the solution of the system is:

$$p = (x, y) = \left(\frac{BF - CE}{BD - AE}, \frac{AF - CD}{AE - BD}\right)$$

Considering the cases of the Fig. 6:

- **Case 1:** $l_1[1][1] = l_1[2][1] \Rightarrow x_1 = x_2 \Rightarrow B = 0 \Rightarrow p = \left(\frac{C}{A}, \frac{AF - CD}{AE}\right)$
- **Case 2:** $l_1[1][2] = l_1[2][2] \Rightarrow y_1 = y_2 \Rightarrow A = 0 \Rightarrow p = \left(\frac{BF - CE}{BD}, \frac{C}{B}\right)$
- **Case 3:** $A, B \neq 0 \Rightarrow p = \left(\frac{BF - CE}{BD - AE}, \frac{AF - CD}{AE - BD}\right)$

Algorithm 2 begins with the assumption that the two segments intersect (Line 4). In such a case, the segments

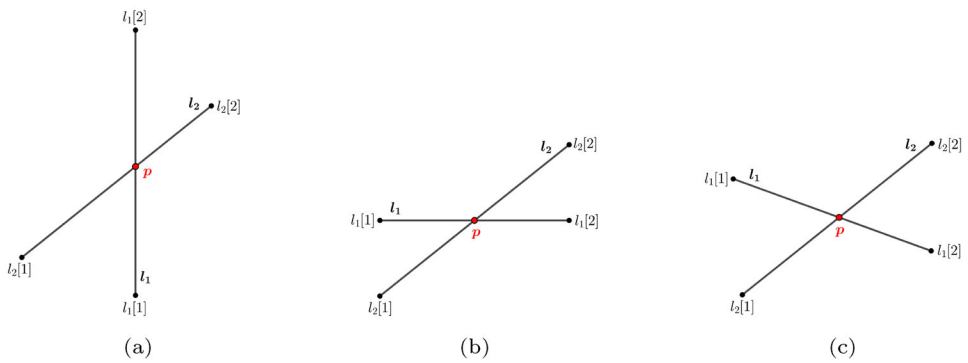


Fig. 6 Algorithm 2: relative position of segments l_1 and l_2

can be secant or coincident. If they are secant (Line 5), the intersection point p is calculated considering case 1 (Line 10), case 2 (Line 12) or case 3 (Line 14). If they are coincident (Line 16), infinite intersection points are obtained and, for example, $p = l_2[1]$ is considered to be the intersection point. The computational cost is determined by the function *INTERSECTION (seg1, seg2: segments)* in $O(1)$ time. This algorithm returns *true* if the segments intersect and *false* if they do not. Except for minor modifications, this function can be found in Cormen et al. [30].

Algorithm 2: *INTERSECTIONp* (l_1, l_2)

Input: l_1, l_2: segments
Output: point of intersection of segments l_1 and l_2

```
1  u1 ← {l1[2][1] − l1[1][1], l1[2][2] − l1[1][2]]}
       // u1 = (x2 − x1, y2 − y1)
2  u2 ← {l2[2][1] − l2[1][1], l2[2][2] − l2[1][2]]}
       // u2 = (x4 − x3, y4 − y3)
3  paral ← u1[1] · u2[2] = u1[2] · u2[1]
   // u1//u2 ⇔ (x2 − x1) · (y4 − y3) = (y2 − y1) · (x4 − x3)
4  if INTERSECTION (l1, l2) = true then
5      if paral = false then
6          A ← l1[2][2] − l1[1][2]; B ← l1[1][1] − l1[2][1]
7          C ← l1[1][1] · l1[2][2] − l1[2][1] · l1[1][2]
8          D ← l2[2][2] − l2[1][2]; E ← l2[1][1] − l2[2][1]
9          F ← l2[1][1] · l2[2][2] − l2[2][1] · l2[1][2]
10         if l1[1][1] = l1[2][1] then
11             p ← {C/A, (A · F − C · D)/(A · E)}
12         else if l1[1][2] = l1[2][2] then
13             p ← {(B · F − C · E)/(B · D), C/B}
14         else
15             p ← {(B · F − C · E)/(B · D − A · E), (A · F − C ·
                   D)/(A · E − B · D)}
16     else
17         p ← l2[1]
18 return p
```

4.2.2 Segment-side intersection (Algorithm 3)

This algorithm allows us to determine when the intersection of a segment l and a side s is possible within a *polygon* or a hole of *holes*. This algorithm returns *true* if the intersection is not allowed and *false* otherwise. It begins by assuming that the intersection exists (Line 2) in $O(1)$ time and computes the intersection point by Algorithm 2 (Line 3) in $O(1)$ time. Because there are three relative positions for the intersection of the segments (Fig. 6) (Lines 4, 6, 8), the algorithm calculates the points m_1 and m_2, given an epsilon (Line 1, $\epsilon \approx 0$) radio environment. The calculation of m_1 and m_2 is made from the relative position of segment l and side s considering Fig. 6 and is as follows:

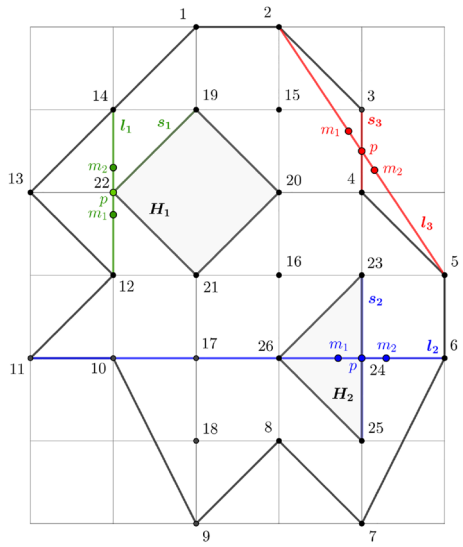


Fig. 7 Algorithm 3: segment-side intersection

- **Case 1:** $l_1[1][1]{=}l_1[2][1] \Rightarrow \begin{cases} m_1{=}(l_1[1][1],\ p[2]-\epsilon) \\ m_2{=}(l_1[1][1],\ p[2]+\epsilon) \end{cases}$
- **Case 2:** $l_1[1][2]{=}l_1[2][2] \Rightarrow \begin{cases} m_1{=}(p[1]-\epsilon,\ l_1[1][2]) \\ m_2{=}(p[1]+\epsilon,\ l_1[1][2]) \end{cases}$
- **Case 3:** Let $r \equiv Ax + By = C$ be the straight line passing through the segment l. If $p = (x, y)$ and an epsilon radius environment is made, $x \pm \epsilon$, we obtain $A(x \pm \epsilon) + By = C$. Then, $y = \dfrac{C - A(x \pm \epsilon)}{B}$ and the coordinates of m_1 and m_2 are as follows:

$$\begin{cases} m_1{=}\left(p[1]-\epsilon,\ \dfrac{C-A(x-\epsilon)}{B}\right) \\ m_2{=}\left(p[1]+\epsilon,\ \dfrac{C-A(x+\epsilon)}{B}\right) \end{cases}$$

Figure 7 shows three intersections, one for each of the cases presented in Fig. 6, and a new function *ALIGNEDpol (points, p, holes[i])* is used. The algorithm returns *true* if point p belongs to one of the sides of $holes[i]$ and can be computed in $O(m)$ time. In addition, $num = \{0, 1\}$ does not allow us to decide if the intersection is for *polygon* ($num = 0$) or for *holes[i]* ($num = 1$). The first intersection, $l_1 \cap s_1$, is possible for H_1 because the points m_1 and m_2 are outside, *PointIn (points,* m_1, H_1*) = false* and *PointIn (points,* m_2, H_1*) = false* (Line 17); the second, $l_2 \cap s_2$, is not possible because m_1 belongs to the interior of H_2 and is not a point on side s_2, *PointIn (points,* m_1, H_2*) = true* and *ALIGNEDpol (points,* m_1, H_2*) = false* (Line 17); and the third, $l_3 \cap s_3$, is also not possible because m_2 is outside the *polygon*, *PointIn (points,* m_2, *polygon) = false* (Line 14).

The computational cost of Algorithm 3 is determined by *polygon* (Line 13, $num = 0$) in $O(n)$ time by the *PointIn*

function and *holes[i]* (Line 16, $num = 1$) in $O(m^2)$ time by the functions *PointIn* in $O(m)$ time and *ALIGNEDpol* in $O(m)$ time.

Algorithm 3: *INTERSECTIONpoly (l, s, poly, num)*

Input: l: segment, s: side of poly (*polygon* or *holes[i]*), num = {0, 1}
Output: *true* if the intersection segment l and side s is not allowed and *false* if they do

```
1  int ← false; ε ≈ 0
2  if INTERSECTION (l, s) = true then
3      p = INTERSECTIONp (l, s) // Alg.2
4      if l[1][1] = l[2][1] then
5          m1 ← {l[1][1], p[2] − ε}; m2 ← {l[1][1], p[2] + ε}
6      else if l[1][2] = l[2][2] then
7          m1 ← {p[1] − ε, l[1][2]}; m2 ← {p[1] + ε, l[1][2]}
8      else
9          A ← l[2][2] − l[1][2]; B ← l[1][1] − l[2][1]
10         C ← l[1][1] · l[2][2] − l[2][1] · l[1][2]
11         m1 ← {p[1] − ε, (C − A · (p[1] − ε))/B}
12         m2 ← {p[1] + ε, (C − A · (p[1] + ε))/B}
13     if num = 0 then
14         if PointIn (points, m1, poly) = false or PointIn (points,
           m2, poly) = false then
15             int ← true
16     else
17         if (PointIn (points, m1, poly) = true and ALIGNEDpol
           (points, m1, poly) = false) or (PointIn (points, m2, poly) =
           true and ALIGNEDpol (points, m2, poly) = false) then
18             int ← true
19 return int
```

4.2.3 Segment-polygon intersection (Algorithm 4)

We compute when the intersection of a segment l with *polygon* or *holes* is possible. This algorithm returns *true* if the intersection is not allowed and *false* otherwise, and begins by first calculating the *polygon* sides using the function *SIDESpol (points, poly: polygon or holes[i])* in $O(n)$ time (Line 2) and then applies Algorithm 3 (Line 4) to determine if the intersection of segment l with any polygon side is not allowed. If true, the algorithm terminates and returns *true* (Line 5). If all intersections are possible (Line 7) the algorithm compare segment l with all sides of the holes: $holes[1], holes[2], \ldots, holes[h]$ (Lines 9, 13) and perform the same process as done for *polygon*. If for some $holes[i]$, $1 \leq i \leq h$, the intersection is not allowed, the algorithm terminates, returns *true* (Line 11) and is not executed for the following holes: $i + 1, \ldots, h$. In addition, Algorithm 4 allows us to decide if the initial polygon is with holes or without holes (Line 8); thus, it is possible to calculate the solution in the same algorithm for each of the two versions.

The computational cost of Algorithm 4 can be solved for *polygon* in $O(n^2)$ time by loop (Line 3) computed in $O(n)$ time and Algorithm 3 (Line 4) in $O(n)$ time. If the polygon has holes (Line 8), the computational cost is $O(hm^4)$ because we must compute a loop (Line 9) in $O(h)$ time; function *SIDESpol (points, poly: polygon or holes[i])* in $O(m)$ time; loop (Line 13) in $O(m)$ time; and Algorithm 3 (Line 14) in $O(m^2)$ time. Therefore, the final computational cost of Algorithm 4 is $Max(n^2, hm^4) = n^2$ if we assume $\#(P) \simeq n >> m = \max\{m_1, m_2, \ldots, m_h\}$. This cost is valid for an initial polygon with holes or without holes.

Algorithm 4: *INTERSECTIONpol (l)*

Input: l : segment
Output: *true* if the intersection of segment l with *polygon* or *holes* is not allowed and *false* if they do

```
1  int ← false
2  sides ← SIDESpol (points, polygon)
3  for i ← 1 to Length(sides) do
4      int ← INTERSECTIONpoly (l, sides[i], polygon, 0)
           // Alg.3
5      if int = true then
6          Break
7  if int = false then
8      if holes ≠ ∅ then
9          for i ← 1 to Length(holes) do
10             hol ← SIDESpol (points, holes[i])
11             if int = true then
12                 Break
13             for j ← 1 to Length(holes[i]) do
14                 int ← INTERSECTIONpoly (l, hol[j], holes[i],
                       1) // Alg.3
15                 if int = true then
16                     Break
17 return int
```

4.3 Computation of the adjacency matrix (Algorithm 5)

We compute the adjacency matrix for a lattice polygon with holes or without holes by Algorithm 5 in $O(n^5)$ time using the sets:

$$\begin{cases} points = P = P_0 - \bigcup_{i=1}^{h} (\iota H_i) \\ polygon = \partial P_0 \\ holes = \bigcup_{i=1}^{h} holes[i] = \bigcup_{i=1}^{h} (\partial H_i) \end{cases}$$

Algorithm 5 first initializes the adjacency matrix with all its terms equal to 0 (Line 1). Then, all *polygon* sides are calculated (Line 2), and using two loops (Lines 3, 4), all the points of the set *points* are traversed with $j = i + 1$ to

form the l sides. If side l belongs to a *polygon* side (Line 6), the value 1 is assigned to the matrix for points p_i and p_j; otherwise (Line 8), Algorithm 4 (Line 9) is used to decide if the intersection of l with *polygon* or *holes* is allowed or not. If the intersection is allowed, the midpoint of the side is taken (Line 11), and if that point is inside or on a *polygon* side (Line 12), the adjacency matrix is assigned the value 1. Of all the intersections that have been considered valid (Line 9) (i.e., those in which the intersection of a segment l and any side s is not allowed), we must remove or assign the matrix value 0 when we find holes such as those shown in Fig. 5. Thus, if for H_1 we consider the segment l formed by the points $\{19, 21\}$, $matrix[19, 21] = 0$ because the midpoint is not on one of the sides of H_1. Similarly, $matrix[20, 22] = 0$ and $matrix[24, 26] = 0$. Finally, the assignment $matrix[j, i] = matrix[i, j]$ (Line 21) is performed because the adjacency matrix is a symmetric matrix and for the moment, only the values above the primary diagonal have been calculated, thus forming an upper triangular matrix.

The computational cost of Algorithm 5 is determined by Lines 3–13 in $O(n^5)$ time by loop (Line 3) computed in $O(n)$ time, loop (Line 4) in $O(n)$ time, Algorithm 4 (Line 9) in $O(n^2)$ time and the *PointIn* function (Line 12) in $O(n)$ time. If, in addition, the polygon has holes (Line 14), the computational cost is $O(hm^3)$ by loop (Line 15) computed in $O(h)$ time, loop (Line 16) in $O(m)$ time, loop (Line 17) in $O(m)$ time and the function *ALIGNEDpol* in $O(m)$ time. Therefore, Algorithm 5 can be solved in $O(n^5)$ time for an initial polygon with holes or without holes, because $Max(n^5, hm^3) = n^5$.

Algorithm 5: *MATRIX (points)*

Input: points
Output: adjacency matrix

```
1  Initialize matrix, matrix[i,j] ← 0
2  sides ← SIDESpol (points, polygon)
3  for i ← 1 to Length(points)-1 do
4      for j ← i + 1 to Length(points) do
5          l ← {points[i], points[j]}
6          if Length(Union(sides, {l})) = Length(sides) then
7              matrix[i,j] ← 1
8          else
9              if INTERSECTIONpol (l) = False // Alg.4
10             then
11                 m ← MEDIO (points[i], points[j])
12                 if PointIn (points, m, polygon) = true then
13                     matrix[i,j] ← 1
14 if holes ≠ ∅ then
15     for i ← 1 to Length(holes) do
16         for j ← 1 to Length(holes[i])-1 do
17             for k ← j + 1 to Length(holes[i]) do
18                 m ← MEDIO (points[holes[i][j]],
                   points[holes[i][k]])
19                 if ALIGNEDpol (points, m, holes[i]) = False then
20                     matrix[holes[i][j], holes[i][k]] ← 0
21 Symmetric matrix, matrix[j,i] ← matrix[i,j]
22 return matrix
```

5 Maximum area or perimeter simple *k*-gon in a lattice polygon with holes

Once the adjacency matrix has been calculated, we obtain the maximum area or perimeter simple k-gon in a lattice polygon P with h holes and $\#(P) = N \simeq n$. We thus follow the next process in three steps:

1. Determine if a hole is contained within a k-sided polygon (Algorithm 6).
2. Compute the sides of polygons that are a certain distance from two points (Algorithm 7).
3. Compute the simple k-gon of a maximum-area or perimeter inscribed in a lattice polygon with holes (Algorithm 8).

5.1 Hole contained in a *k*-sided polygon (Algorithm 6)

Algorithm 6 allows us to determine when a hole H is contained in a k-sided polygon, and returns *false* if it is not contained or does not intersect the polygon and *true* otherwise.

The computational cost of Algorithm 6 can be solved in $O(k^2m)$ time by loop (Line 5) computed in $O(m)$ time, Algorithm 1 (Line 7) in $O(k)$ time and the function ALIGNEDpol (Line 9) in $O(k)$ time. Figure 8 shows two holes: $H_1 = (19,$

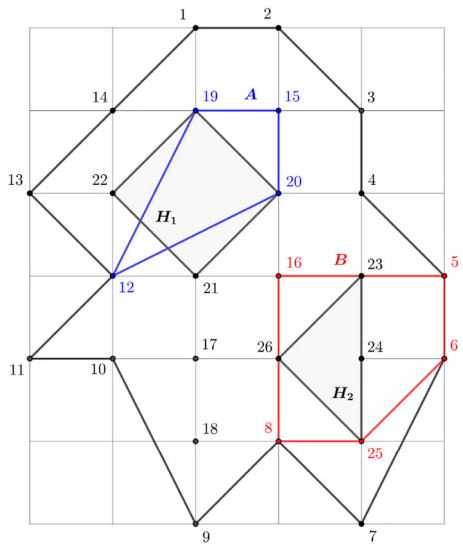


Fig. 8 Algorithm 6: hole contained in a k-sided polygon

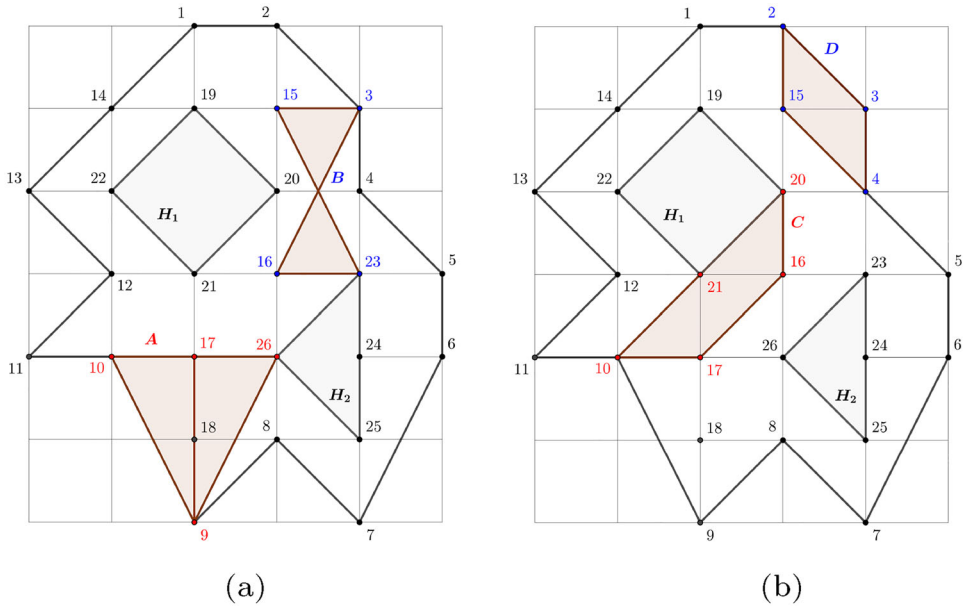


Fig. 9 *SEGMENTS (points, path)* and *ALIGNED (points, path)* functions

20, 21, 22) and H_2 = (23, 24, 25, 26), and two polygons: A = (15, 20, 12, 19) and B = (5, 6, 25, 8, 16). Algorithm 6 gives the following results: *HOLES (points, A, H_1) = true* and *HOLES (points, B, H_2) = true*.

Algorithm 6: *HOLES (points, polygon, H)*

Input: points, polygon, H: hole
Output: *false* if the hole H is not contained or not intersect the k-sided polygon and *true* otherwise

```
1  int ← false; hole ← ∅
2  for i ← 1 to Length(H) do
3  |  Insert(hole, H[i])
4  Insert(hole, H[1])
5  for i ← 1 to Length(hole)-1 do
6  |  m ← MEDIO (points[hole[i]], points[hole[i+1]])
7  |  if PointIn (points, m, polygon) = true // Alg.1
8  |  then
9  |  |  if ALIGNEDpol (points, m, polygon) = false then
10 |  |  |  int ← true
11 |  |  |  break
12 return int
```

5.2 Sides (Algorithm 7)

We compute the sides of the polygons that are a certain distance from *point1* to *point2* using Algorithm 7. First, the algorithm begins by considering that the two points are connected (Line 2). Then, three subalgorithms are used: Lines 3–11, Lines 12–15 and Lines 16–24, with computational cost of $O(n^3k)$, $O(n^2k^3)$ and $O(n^2k^2hm)$, respectively. Therefore, Algorithm 7 can be solved in $O(n^3k)$ time because $Max(n^3k, n^2k^3, n^2k^2hm) = n^3k$ if $k >> h$ and $k >> m$ is assumed. An exhaustive analysis of each of the subalgorithms is as follows:

Subalgorithm 1 (Lines 3–11): Computes all edges from *point1* to *point2*. First, this subalgorithm computes all the edges that are at distance 1 from *point1* and successively stores them until the chosen distance is reached. The computational cost is determined by the loops: Line 3 is computed in $O(k)$ time because $k = distance + 1$; Line 5 is computed in $O(n^2)$, because the maximum number of edges coincides with the combinatorial number $\binom{N}{2} = \frac{N*(N-1)}{2} \simeq N^2 \simeq n^2$; Line 8 is computed in $O(n)$ time, because the adjacency matrix is a symmetric square matrix of order $N \simeq n$. Therefore, the computational cost is $O(n^3k)$.

Subalgorithm 2 (Lines 12–15): Of all the solutions that appear in "temp" (Line 11), we keep those where the final point coincides with *point2* (Line 14). In addition, functions *SEGMENTS (points, path)* and *ALIGNED (points, path)* are used to discard those possible solutions that cannot form a polygon. The first function determines when two segments (edges) intersect or not, in $O(k^2)$ time, and the second, in $O(k)$ time, returns 0 if three consecutive points of the *path* are aligned and not 0 otherwise. In Fig. 9a we see that no possible solutions, A = (10, 26, 9, 17) and B = (3, 16, 23, 15) can form a quadrilateral, and in Fig. 9b, we observe that C = (20, 16, 17, 10, 21) cannot be a pentagon because the points 10, 21 and 20 are aligned, because the polygon is a quadrilateral. However, D = (2, 3, 4, 15) is a quadrilateral. The computational cost is determined by the loop in Line 13 in $O(n^2)$ and the above functions, therefore obtaining a computational cost of $O(n^2k^3)$ time.

Subalgorithm 3 (Lines 16–24): From all of the polygons obtained in Line 15, we remove those containing one or more holes. The computational cost is determined by the loops: Line 17 computed in $O(n^2)$ time; Line 19 computed in $O(h)$, because P is a lattice polygon with h holes; Line 20 computed in $O(k^2m)$ by Algorithm 6. Thus, the computational cost is $O(n^2k^2hm)$.

Algorithm 7: *SIDES (point1, point2, distance, matrix)*

Input: point1, point2 ∈ *points*, distance ∈ ℕ, matrix: adjacency matrix
Output: sides between point1 and point2 for a certain distance

```
1  temp ← {point1}
2  if matrix(point1, point2) = 1 then
3      for i ← 1 to distance do
4          edges ← ∅
5          for j ← 1 to Length(temp) do
6              last ← temp[j][i]
7              if last ≠ point2 then
8                  for k ← 1 to N do
9                      if matrix(last,k) = 1 and
                       Length(Union(temp[j],k)) ≠ Length(temp[j])
                       then
10                         Insert(edges, Insert(temp[j],k))
11         temp ← edges
12     edges ← ∅
13     for i ← 1 to Length(temp) do
14         if temp[i][distance + 1] = point2 and SEGMENTS
           (points, temp[i]) = false and ALIGNED (points, temp[i])
           ≠ 0 then
15             Insert(edges, temp[i])
16     sides ← ∅
17     for i ← 1 to Length(edges) do
18         h ← 0
19         for j ← 1 to Length(holes) do
20             if HOLES (points, edges[i], holes[j]) = false
                   // Alg.6
21             then
22                 h ← h+1
23         if h = Length(holes) then
24             Insert(sides, edges[i])
25 return sides
```

5.3 Solutions (Algorithm 8)

We compute the maximum-area or perimeter simple k-gon contained in a lattice polygon P with h holes and with coordinates belonging to the square grid G_L by Algorithm 8. First, the algorithm computes all simple k-gons (Lines 1–6) by Algorithm 7 in $O(n^5k)$ time by the loops in Lines 2–3 computed in $O(n^2)$ time and the function *SIDES (point1, point2, distance, matrix)* in $O(n^3k)$ time. Then, the functions *UPDATE (polygons, function)* and *DUPLICATES (polygons)* are computed in $O(n^2k)$ and $O(n^4k\log k)$ time, respectively [28]. The first function calculates the largest area or perimeter polygon depending on the value of the "function" parameter (0 for the largest area and 1 for the perimeter), and the second function eliminates repeated solutions. For example, if we take one of the solutions of the simple 5-gon, (4, 18, 10, 12, 21) in Fig. 5, there are $2k$ solutions with the same area and perimeter for each k-gon. Thus, the function *DUPLICATES(polygons)* chooses a representative of the above solutions. Therefore, Algorithm 8 can be solved in $O(n^5k)$ time, because $Max(n^5k, n^2k, n^4k\log k) = n^5k$.

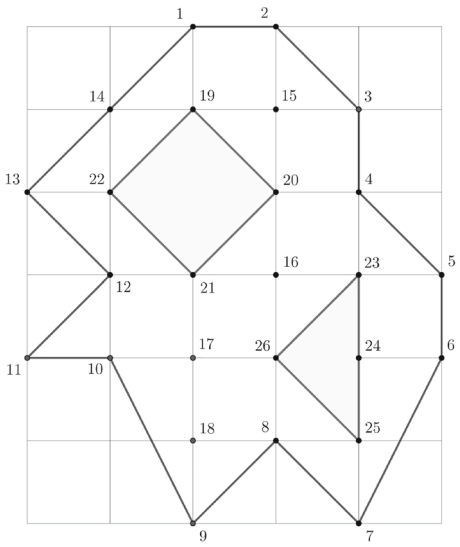


Fig. 10 Example 1: maximum-area and perimeter simple k-gon

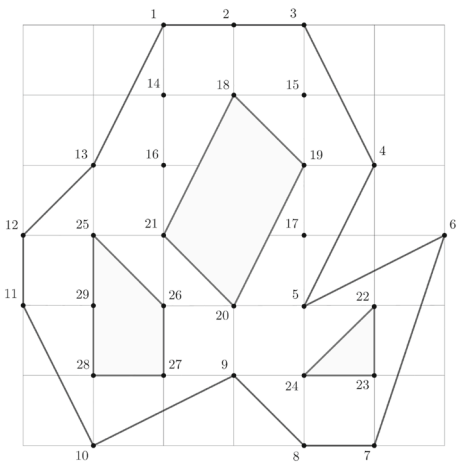


Fig. 11 Example 2: maximum-area and perimeter simple k-gon

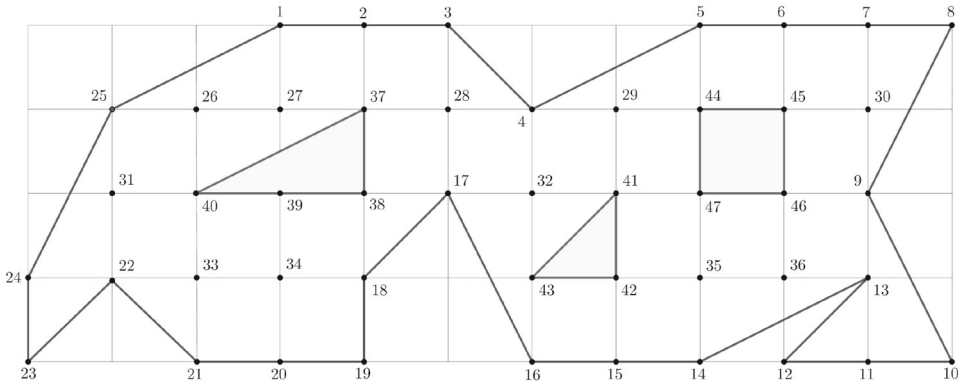


Fig. 12 Example 3: maximum-area simple k-gon

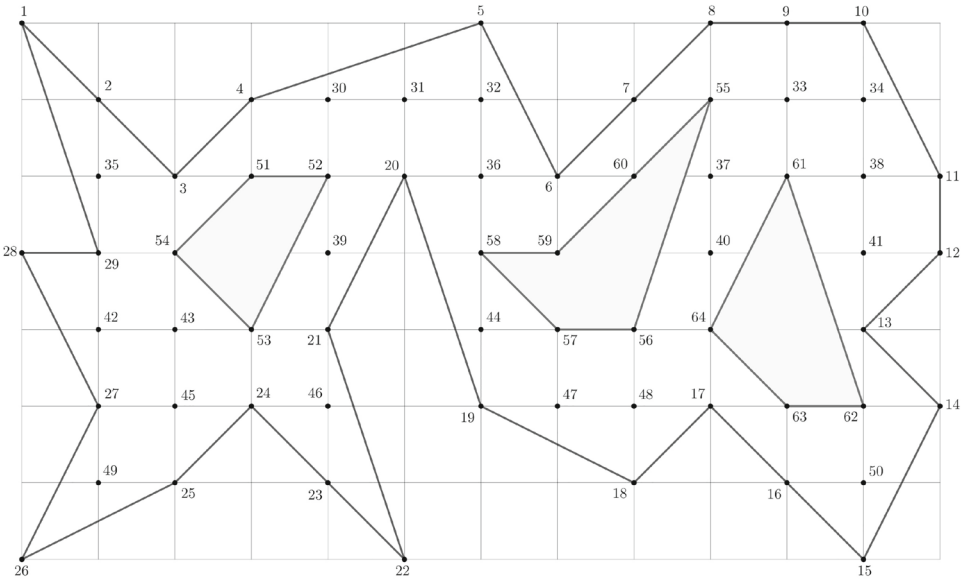


Fig. 13 Example 4: maximum-area simple k-gon

Algorithm 8: *POLYGONS (N, distance, matrix, function)*

Input: N: number of points of P, distance $\in \mathbb{N}$, matrix: adjacency matrix, function = {0, 1}
Output: simple k-gons contained in P with h holes

```
polygons ← ∅; sides ← ∅
for i ← 1 to N-1 do
    for j ← i + 1 to N do
        if SIDES (i, j, distance, matrix) ≠ ∅ // Alg.7
        then
            Insert(sides, SIDES (i, j, distance, matrix))
polygons ← DUPLICATES (UPDATE (sides, function))
return polygons
```

5.4 Our experimental results

Using Algorithm 8, proven it is possible to compute the maximum-area or maximum-perimeter simple k-gon contained in a lattice polygon with holes. We now consider some solutions in Fig. 5 (Fig. 10) with computation time in seconds.

In addition, we consider three new examples (Figs. 11, 12, 13) to demonstrate how Algorithm 8 can be applied to random polygons and Tables 2, 3, 4 and 5, where the solutions of the simple k-gons of maximum-area or perimeter are presented.

- Figure 10. $N = \#(points) = 26$, k = distance +1, function = {0,1}.
- Figure 11. $N = \#(points) = 29$, k = distance +1, function = {0,1}.
- Figure 12. $N = \#(points) = 47$, k = distance +1, function = {0,1}.
- Figure 13. $N = \#(points) = 64$, k = distance +1, function = {0,1}.

Finally, Figure 14 geometrically shows some solutions for the maximum area of a simple 9-gon for Example 3 and Example 4.

Table 2 Solutions example 1: maximum-area and perimeter simple k-gon

Dist.	k-gon	POLYGONS (26, distance, matrix, 0), area	Time
2	Triangle	(3,10,9)	0,01
3	Quadrilat.	(3,10,9,4)	0,54
	Rectangle	(10,20,23,18)	0,67
4	Pentagon	(1,20,10,9,3), (1,20,10,9,4), (2,20,10,9,4), (3,23,26,9,10)	1,23
5	Hexagon	(1,20,10,9,4,2), (1,20,10,9,4,3), (2,19,20,10,9,4)	4,76
6	Heptagon	(1,19,20,10,9,4,2), (1,20,10,9,4,3,2), (1,13,19,20,10,9,3)	34,67
		(1,19,20,10,9,4,3), (1,20,10,9,26,23,3), (1,13,19,20,10,9,4)	
		(1,20,10,18,23,7,5), (2,19,20,10,9,4,3), (2,13,19,20,10,9,4)	
		(2,14,19,20,10,9,4), (2,13,7,25,22,19,6), (2,20,10,18,23,7,6)	
7	Octagon	(1,13,7,25,22,19,6,2), (1,13,19,20,10,9,4,2)	210,45
		(1,20,10,18,23,7,6,2), (1,13,19,20,10,9,4,3)	
		(1,20,10,9,26,23,7,5), (2,19,20,10,18,23,7,6)	
		(2,20,10,9,26,23,7,6)	
8	Nonagon	(1,20,10,9,26,23,7,6,2), (2,19,20,10,9,26,23,7,6)	1.820,34

Dist.	k-gon	POLYGONS (26, distance, matrix, 1), perimeter	Time
2	Triangle	(3,10,9)	0,01
3	Quadrilat.	(3,18,22,9)	0,45
4	Pentagon	(3,18,4,9,10)	1,12
5	Hexagon	(2,13,7,22,19,6)	5,01
6	Heptagon	(2,26,23,25,13,7,6)	32,67
7	Octagon	(2,13,7,22,19,23,15,6)	207,19
8	Nonagon	(2,22,17,3,18,4,9,12,13)	1.914,56

6 Approximation for the maximum area or perimeter simple k-gon in a closed contour with holes

Let C be a closed contour with h holes, $H_1, H_2, \ldots, H_h$ and C_0 the outer contour to C. Moreover, let R the rectangle of the minimum area that encloses the closed contour C_0 [31] and Π be a regular partition of R with *partition size* L. We define *lower area* $\underline{A}(C_0, \Pi)$ as the largest area lattice polygon P_0 contained in C_0 and *upper area* $\overline{A}(H_j, \Pi)$ as the smallest area lattice polygon Q_j containing each H_j, $1 \leq j \leq h$ and both built by points of G_L. By Pick's theorem [29],

$$\underline{A}(C_0, \Pi) = \left(\#(\imath P_0) + \frac{\#(\partial P_0)}{2} - 1\right) \cdot L^2$$
$$\overline{A}(H_j, \Pi) = \left(\#(\imath H_j) + \frac{\#(\partial H_j)}{2} - 1\right) \cdot L^2$$

Theorem 6.1 *Let C be a closed contour with h holes, $H_1, H_2, \ldots, H_h$ and C_0 the outer contour to C. Then, there exists a sequence of regular partitions $\{\Pi_n\}_{n\in\mathbb{N}}$ with $\Pi_i \preceq \Pi_{i+1}$ for all i such that $\lim_{n\to\infty}(\underline{A}(C_0, \Pi_n) - \sum_{j=1}^{h} \overline{A}(H_j, \Pi_n)) = A(C)$, where $A(C)$ is the area of the closed contour C.*

Proof We consider a regular partition $\dot{\Pi}$ as finer than Π. Then, $\underline{A}(C_0, \Pi) \leq \underline{A}(C_0, \dot{\Pi})$ and $\overline{A}(H_j, \Pi) \geq \overline{A}(H_j, \dot{\Pi})$ with $1 \leq j \leq h$. Therefore, $\underline{A}(C_0, \Pi) - \sum_{j=1}^{h} \overline{A}(H_j, \Pi) \leq \underline{A}(C_0, \dot{\Pi}) - \sum_{j=1}^{h} \overline{A}(H_j, \dot{\Pi})$. Then, exists a sequence of regular partitions $\{\Pi_n\}_{n\in\mathbb{N}}$ with $\Pi_i \preceq \Pi_{i+1}$ for all i such that $\lim_{n\to\infty}(\underline{A}(C_0, \Pi_n) - \sum_{j=1}^{h} \overline{A}(H_j, \Pi_n)) = A(C)$. □

In general, Fig. 15 shows how to obtain the area of the closed contour C with h holes from the outer contour C_0 and the holes H_j, by the inscribed limit area within the lattice polygon P with h holes, constructing finer partitions with $L_{i+1} = L_i/2$ if $\Pi_i \preceq \Pi_{i+1}$ for all i. Moreover, if by Algorithm 8 we prove that it is possible to compute the maximum-area or perimeter simple k-gon contained in a lattice polygon P with h holes, with Theorem 6.1 we achieve the goal of the paper, the simple k-gon contained in P is also the maximum in area or perimeter within C. In particular, we show three partitions for the same closed contour C with two holes, with partition sizes: $L_1 = 4$, $L_2 = 2$, $L_3 = 1$, number of points: 26, 112, 450 and adjacency matrix: A, $\dot{A}$, $\ddot{A}$, respectively, and compute the maximum area simple 4-gon: POLYGONS (26,3,A,0) (Fig. 10), POLYGONS (112,3,$\dot{A}$,0) and POLYGONS (450,3,$\ddot{A}$,0). As the results show, the area of the simple 4-gon is larger if the partition is thinner each time.

Table 3 Solutions example 2: maximum-area and perimeter simple k-gon

Distance	k-gon	POLYGONS (29, distance, matrix, 0), area
2	Triangle	(1,11,3)
3	Quadrilateral	(1,11,18,2), (1,13,25,3), (1,11,18,4), (1,15,18,11)
		(2,12,11,3), (2,18,21,12), (7,13,25,8), (8,25,20,19)
	Rectangle	(1,14,15,3), (5,26,27,24)
4	Pentagon	(1,11,10,25,3), (1,11,18,4,3)
5	Hexagon	(1,25,8,7,21,2), (1,11,18,19,4,3), (1,25,8,7,21,18)
		(2,21,7,8,25,12), (2,21,7,25,10,12), (2,21,8,25,10,12)
6	Heptagon	(1,11,18,19,9,4,3), (2,21,7,8,25,10,12)
7	Octagon	(1,11,10,25,8,7,21,2), (1,25,8,7,21,18,4,3)
		(1,18,21,7,8,25,10,11), (2,21,20,19,8,25,10,12)
8	Nonagon	(1,11,10,25,8,19,20,21,2), (1,11,10,25,7,21,18,4,3)
		(1,11,10,25,8,7,21,18,3), (1,11,10,25,8,21,18,4,3)
		(1,11,25,8,7,21,18,4,3), (1,25,8,19,20,21,18,4,3)
		(1,18,21,20,19,8,25,10,11)

Distance	k-gon	POLYGONS (29, distance, matrix, 1), perimeter
2	Triangle	(1,11,3)
3	Quadrilateral	(2,25,7,21)
4	Pentagon	(2,21,7,25,11)
5	Hexagon	(2,21,7,25,10,12)
6	Heptagon	(3,25,28,27,6,10,12)
7	Octagon	(1,11,14,25,7,21,18,4)
8	Nonagon	(3,25,28,27,4,5,6,10,12)

Table 4 Solutions example 3: maximum-area simple k-gon

Distance	k-gon	POLYGONS (47, distance, matrix, 0), area
2	Triangle	(5,33,8)
3	Quadrilateral	(1,43,6,4)
	Rectangle	(3,41,43,37), (19,38,40,21), (37,44,47,38)
4	Pentagon	(2,25,24,37,13), (2,38,41,15,13)
5	Hexagon	(1,24,37,43,6,4)
6	Heptagon	(1,25,24,37,43,6,4), (2,25,24,37,43,6,4)
		(2,25,24,37,41,15,13), (3,25,24,37,43,6,4)
7	Octagon	(2,25,24,37,43,41,15,13)
8	Nonagon	(1,25,24,37,43,41,15,13,2), (2,25,21,17,40,37,41,15,13)
		(2,25,23,40,37,43,41,15,13), (2,25,24,37,43,41,15,14,13)

If we were interested in computing the rectangle with the largest area by only slightly modifying the algorithms, we could obtain the images shown in Fig. 16. We have taken the same partition sizes, number of points and adjacency matrices as in Fig. 15 and renamed the Algorithm 8, *POLYGONS-RECT*, which computes the rectangle of the maximum area.

The algorithm always provides a solution inside the original contour, being closer to the original contour as the order of the adjacency matrix increases. Figures 15 and 16 show this idea, in (a), the order of the adjacency matrix is 26, in (b) it is 112 and in (c) it is 450.

7 Source code

To perform the experiments in this paper, including a practical application, and to verify the efficiency of the algorithm that has been generated, the source code, has been developed from pseudocode into Java and Python. This source code is

Table 5 Solutions example 4: maximum-area simple k-gon

Distance	k-gon	POLYGONS (64, distance, matrix, 0), area
2	Triangle	(2,49,24), (3,6,5), (4,59,5)
3	Quadrilateral	(2,27,26,24), (3,6,5,4), (8,13,11,10), (8,13,12,10)
	Rectangle	(3,35,49,25), (8,37,38,10), (9,55,41,11)
4	Pentagon	(10,55,56,19,18), (10,55,56,44,18)
5	Hexagon	(10,55,48,20,19,18)
6	Heptagon	(3,52,53,35,49,21,5), (4,52,53,35,49,21,5)
		(8,55,48,20,19,18,10), (9,56,19,18,61,13,11)
		(9,56,44,18,61,13,11), (10,55,56,19,18,61,12)
		(10,55,56,44,18,61,12), (10,55,56,57,20,19,18)
		(10,62,61,18,19,56,55), (10,62,61,18,44,56,55)
7	Octagon	(4,59,55,18,10,8,6,5)
8	Nonagon	(3,20,19,18,10,55,56,44,5), (4,59,55,56,18,10,8,6,5)

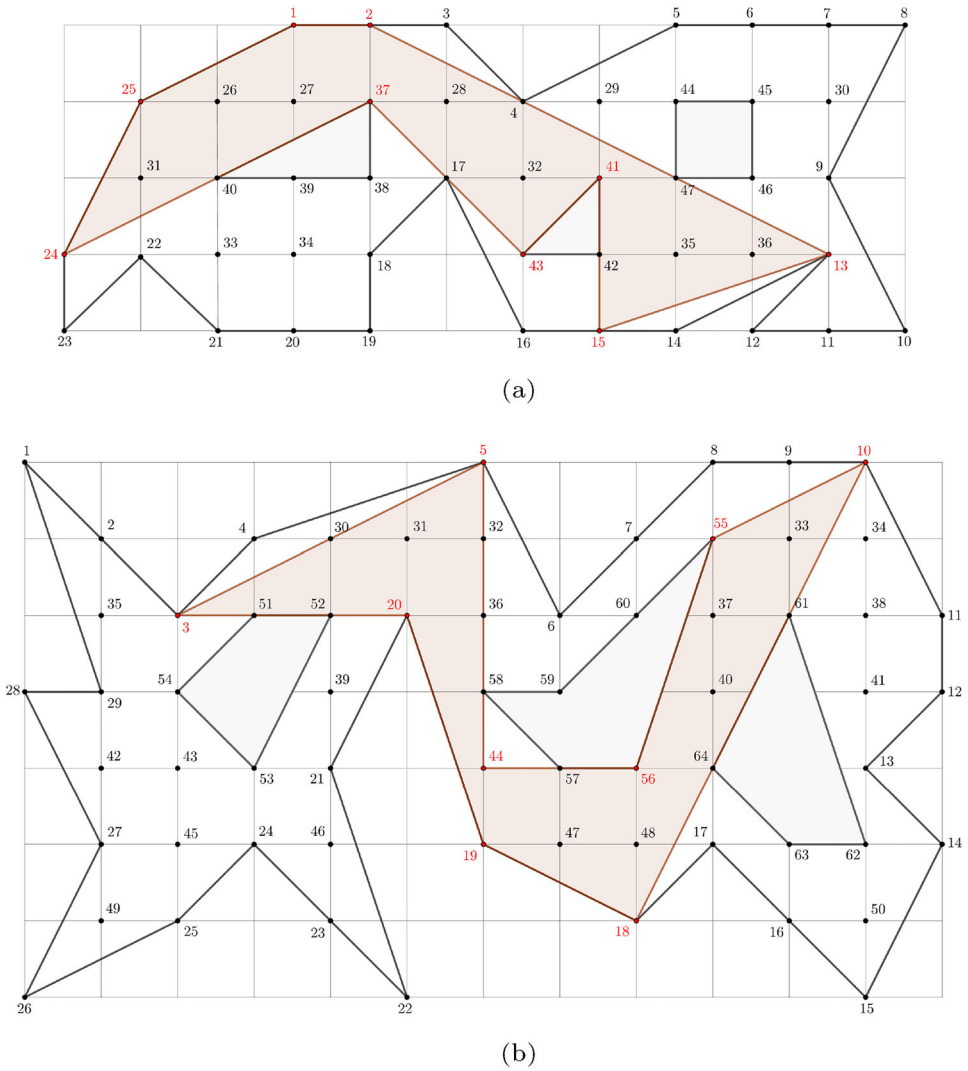


Fig. 14 Maximum area simple 9-gon

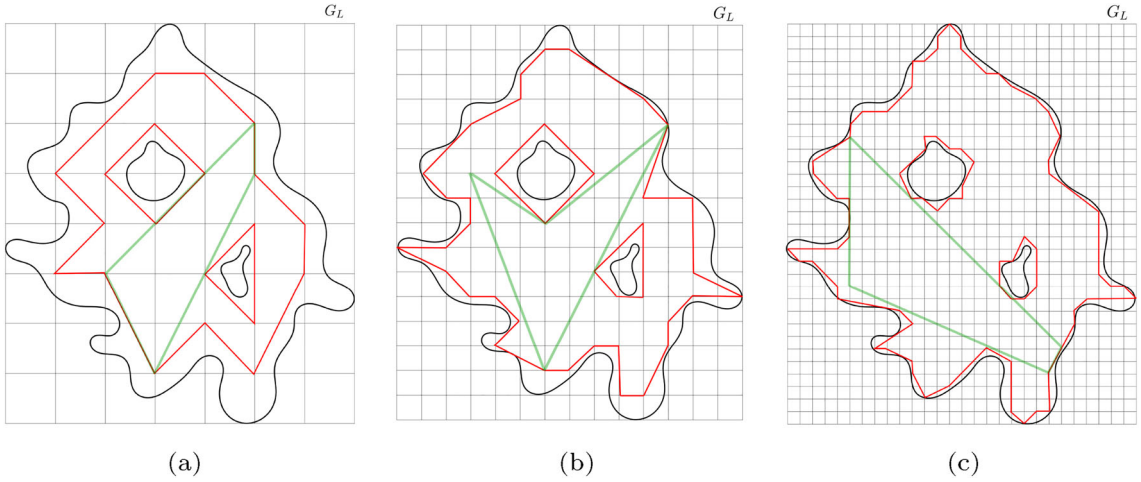


Fig. 15 Maximum area simple 4-gon with different partition sizes

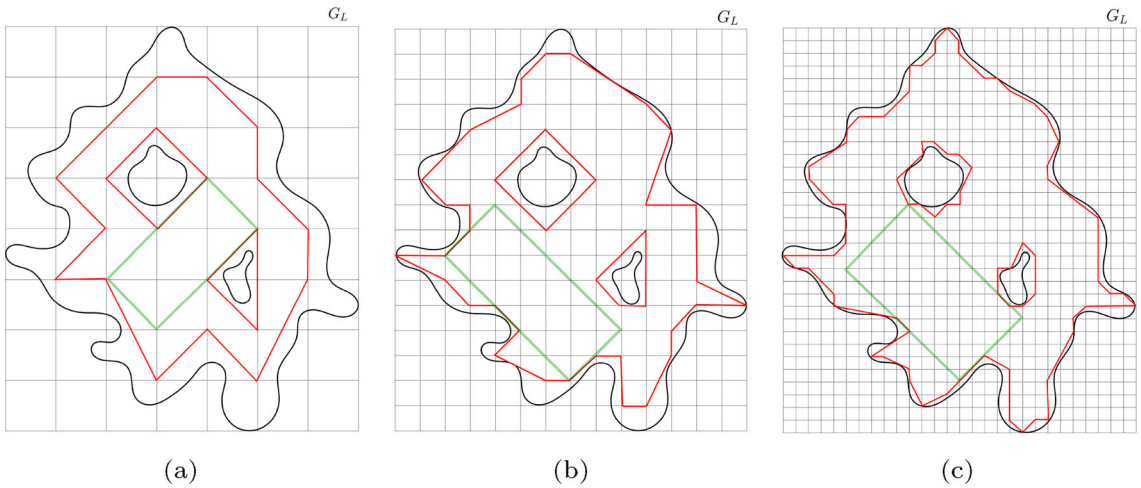


Fig. 16 Maximum area rectangle with different partition sizes

available on GitHub [28] so that any researcher can adapt and configure it to their desired programming language.

8 Conclusions

This paper investigates the problem of finding the simple k-gon with maximum area or perimeter contained in a region of interest with holes. Regions of interest can be irregular and may have inaccessible areas called holes. We developed an algorithm to address these regions of interest that might have been discarded if this algorithm were not available. To compute solutions, the user decides what type of polygon is required (triangle, quadrilateral, pentagon, hexagon, etc.), and the required solution: maximum area or perimeter. The polyvalence of the algorithm is evident, and the user is the one who decides what to calculate. All pseudocode is explained, defined in 8 subprograms, and tested in a practical application. Finally, the source code was developed in Java and Python and has been made available in a public repository on GitHub.

Data availability Data sharing not applicable to this article as no datasets were generated or analysed during the current study. Source code is available at [28].

Declarations

Conflict of Interest The authors declare that they have no conflict of interest.

References

1. Girard, N., Smirnov, D., Solomon, J., Tarabalka, Y.: Polygonal building extraction by frame field learning. In: 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR),

pp. 5887–5896. https://doi.org/10.1109/CVPR46437.2021.00583 (2021)
2. Xu, Z., Liu, Y., Gan, L., Hu, X., Sun, Y., Liu, M., Wang, L.: csboundary: city-scale road-boundary detection in aerial images for high-definition maps. IEEE Robot. Autom. Lett. **7**(2), 5063–5070 (2022). https://doi.org/10.1109/LRA.2022.3154052
3. Bast, H., Hert, S.: The area partitioning problem. https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.29.7473 (2000)
4. Wei, Q., Sun, J., Tan, X., Yao, X., Ren, Y.: The simple grid polygon exploration problem. J. Comb. Optim. **41**(3), 625–639 (2021). https://doi.org/10.1007/s10878-021-00705-5
5. Ali, H., Faisal, S., Chen, K., Rada, L.: Image-selective segmentation model for multi-regions within the object of interest with application to medical disease. Vis. Comput. **37**(5), 939–955 (2021). https://doi.org/10.1007/s00371-020-01845-1
6. Qasmieh, I.A., Alquran, H., Alqudah, A.M.: Occluded iris classification and segmentation using self-customized artificial intelligence models and iterative randomized hough transform. Int. J. Electr. Comput. Eng. **11**(5), 4037 (2021). https://doi.org/10.1049/iet-ipr.2017.0509
7. Gibert, G., D'Alessandro, D., Lance, F.: Face detection method based on photoplethysmography. In: 2013 10th IEEE International Conference on Advanced Video and Signal Based Surveillance, pp. 449–453. https://doi.org/10.1109/AVSS.2013.6636681 (2013)
8. Oksanen, T.: Shape-describing indices for agricultural field plots and their relationship to operational efficiency. Comput. Electron. Agric. **98**(1), 252–259 (2013). https://doi.org/10.1016/j.compag.2013.08.014
9. Anderson, S.L., Murray, S.C.: R/uastools: plotshpcreate: create multi-polygon shapefiles for extraction of research plot scale agriculture remote sensing data. Front. Plant Sci. **11**, 511768 (2020). https://doi.org/10.3389/fpls.2020.511768
10. Ministry of Farming, Fishing and Food of Spain. https://www.mapa.gob.es/es/cartografia-y-sig (2023)
11. Visor SigPac v4.8. https://sigpac.mapa.gob.es/fega/visor (2023)
12. Lee, S., Eom, T., Ahn, H.-K.: Largest triangles in a polygon. Comput. Geom. **98**, 101792 (2021). https://doi.org/10.1016/j.comgeo.2021.101792
13. Kallus, Y.: A linear-time algorithm for the maximum-area inscribed triangle in a convex polygon. arXiv:1706.03049 (2017)
14. Melissaratos, E.A., Souvaine, D.L.: Shortest paths help solve geometric optimization problems in planar regions. SIAM J. Comput. **21**(4), 601–638 (1992). https://doi.org/10.1137/0221038
15. Alt, H., Hsu, D., Snoeyink, J.: Computing the largest inscribed isothetic rectangle. In: CCCG, pp. 67–72. https://www.cs.ubc.ca/sites/default/files/tr/1994/TR-94-28_0.pdf (1995)
16. Boland, R.P., Urrutia, J.: Finding the largest axis-aligned rectangle in a polygon in... In: Proceedings of the 13th the Canadian Conference on Computational Geometry. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.20.7210 (2001)
17. Daniels, K., Milenkovic, V., Roth, D.: Finding the largest area axis-parallel rectangle in a polygon. Comput. Geom. **7**(1–2), 125–148 (1997). https://doi.org/10.1016/0925-7721(95)00041-0
18. Knauer, C., Schlipf, L., Schmidt, J.M., Tiwary, H.R.: Largest inscribed rectangles in convex polygons. J. Discrete Algorithms **13**, 78–85 (2012). https://doi.org/10.1016/j.jda.2012.01.002
19. Molano, R., Rodríguez, P.G., Caro, A., Durán, M.L.: Finding the largest area rectangle of arbitrary orientation in a closed contour. Appl. Math. Comput. **218**(19), 9866–9874 (2012). https://doi.org/10.1016/j.amc.2012.03.063
20. Choi, Y., Lee, S., Ahn, H.-K.: Maximum-area and maximum-perimeter rectangles in polygons. Comput. Geom. **94**, 101710 (2021). https://doi.org/10.1016/j.comgeo.2020.101710
21. Jin, K.: Maximal parallelograms in convex polygons. arXiv:1512.03897 **376**. https://iiis.tsinghua.edu.cn/uploadfile/2015/0605/20150605101629915.pdf (2015)
22. Molano, R., Caballero, D., Rodríguez, P.G., Ávila, M.D.M., Torres, J.P., Durán, M.L., Sancho, J.C., Caro, A.: Finding the largest volume parallelepipedon of arbitrary orientation in a solid. IEEE Access **9**, 103600–103609 (2021). https://doi.org/10.1109/ACCESS.2021.3098234
23. Keikha, V.: Linear-time algorithms for largest inscribed quadrilateral. https://invenio.nusl.cz/record/432419 (2020)
24. Rote, G.: The largest quadrilateral in a convex polygon. arXiv:1905.11203 **381**. arXiv:1905.11203v2 (2019)
25. Goodman, J.E.: On the largest convex polygon contained in a non-convex n-gon, or how to peel a potato. Geom. Dedicata. **11**(1), 99–106 (1981). https://doi.org/10.1007/BF00183192
26. Chang, J.-S., Yap, C.-K.: A polynomial solution for the potato-peeling problem. Discrete Comput. Geom. **1**(2), 155–182 (1986). https://doi.org/10.1007/BF02187692
27. Cabello, S., Cibulka, J., Kyncl, J., Saumell, M., Valtr, P.: Peeling potatoes near-optimally in near-linear time. SIAM J. Comput. **46**(5), 1574–1602 (2017). https://doi.org/10.1137/16M1079695
28. Media Engineering Group (GIM): Source code, Scripts, and Documentation. https://github.com/UniversidadExtremadura/user-defined-polygons-inside-a-contour. Accessed 2023/02/20 (2023)
29. Varberg, D.E.: Pick's theorem revisited. Am. Math. Mon. **92**(8), 584–587 (1985). https://doi.org/10.1080/00029890.1985.11971689
30. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to algorithms, third. New York (2009)
31. Freeman, H., Shapira, R.: Determining the minimum-area encasing rectangle for an arbitrary closed curve. Commun. ACM **18**(7), 409–413 (1975). https://doi.org/10.1145/360881.360919

R. Molano received the B.Sc. degree in mathematics, in 1998, and the B.Sc. degree in computer science, in 2009. He is currently pursuing the Ph.D. degree with the Department of Mathematics, University of Extremadura. He has been an Assistant Professor with the Department of Mathematics, University of Extremadura, since 2008. His research interests include the development of statistical analysis in the field of pattern recognition and image analysis.

J. C. Sancho received the M.Sc. and Ph.D. degrees in computer science from the University of Extremadura, Spain, in 2014 and 2021, respectively. He has been an Assistant Professor with the Department of Computer and Telematics Systems Engineering, University of Extremadura, since 2018. He is the coauthor of several research SCI journal articles. His research interest includes audit, and security software development.

M. M. Ávila received the B.Sc. and M.Sc. degrees in computer science from the University of Extremadura, in 1997 and 1999, respectively, and the Ph.D. degree in computer science, in 2018. She has been an Assistant Professor with the Department of Computer and Telematics Systems Engineering, University of Extremadura, since 2002. She has participated in several research projects, being the coauthor of several SCI journal articles. Her research interests include pattern recognition, data mining and machine learning, information retrieval, and cybersecurity.

P. G. Rodríguez received the Doctor (Ph.D.) degree in computer science, in 2000. He belongs to the Research Group of Media Engineering (GIM) in the area of Computer Languages and Systems in the University of Extremadura (Spain). He was the Director of the School of Technology (Escuela Politécnica) in Cáceres. His teaching is mainly centered on subjects of Programming and Databases. His research interests include the Internet of Things (IoT), bigdata and pattern recognition, and image analysis.

A. Caro received the B.Sc., M.Sc., and Ph.D. degrees in computer science from the University of Extremadura, Spain, in 1993, 1998, and 2006, respectively. He has been an Associate Professor with the Department of Computer and Telematics Systems Engineering, University of Extremadura, since 1999. He is currently the Laboratory Head of the Media Engineering Group. He has participated in the management of both national and international research and development projects and private financing. He is the coauthor of numerous research SCI journal articles and book chapters. His research interests include cybersecurity, big data, machine learning, and pattern recognition.