Journal for Geometry and Graphics
Volume 29 (2025), No. 2, 223–254

An Efficient Method for Obtaining the Maximum k-Gon in a Closed Contour with Obstacles

Rubén Molano, Mar Ávila, Mohammadhossein Homaei, Pablo G. Rodríguez, Andrés Caro

University of Extremadura, Cáceres, Spain
rmolano@unex.es, mmavila@unex.es, mhomaein@alumnos.unex.es, pablogr@unex.es, andresc@unex.es

Abstract. Geometric optimization has been frequently studied in a recurrent way, where it has been fundamental to developing more complete algorithms. Regions of interest can be obtained as user-defined polygons as a first step toward many practical applications. This article focuses on lattice polygons defined on a regular partition and presents an efficient method for computing all possible polygons contained within regions of interest bounded by arbitrary obstacles such as points, segments, and holes. The developed algorithm calculates all the simple polygons with the maximum area or perimeter contained within the region of interest with $O(n^5k)$ computational time. The user can define the polygon to be calculated (triangle, quadrilateral, pentagon, hexagon, etc.) as well as the desired solution: maximum area or maximum perimeter. The paper presents several practical applications that demonstrate the efficiency and versatility of the algorithm. The pseudocode for the algorithm is presented, as well as the source code (Java and Python) in a GitHub repository for research purposes.

Key Words: k-gon, closed contour, polygon, area, perimeter

MSC 2020: 97G30 (primary), 97N80, 68-04

1 Introduction

The inscribed polygon within a closed contour can be used to detect and classify objects in an image and determine their size and shape in the context of image analysis and pattern recognition [17, 28]. Therefore, to segment an object from its background, or in applications such as edge detection and noise reduction in images, it is often useful to have functions that provide inscribed polygons in some region of the image. Another possible application involves pattern recognition, in which the goal is to identify and classify objects in images based on

patterns or features found in the image. The inscribed contours can extract information about the object's shape and size from the image. Based on this information, the object can be classified and differentiated from other objects in the image. Moreover, it is useful for measuring objects. The size and shape of objects are often measured as part of image analysis. Inscribed polygons are useful for measuring the lengths of their sides and their vertex distances. As a result of this information, it is possible to estimate the size and shape of the object in the image. Additionally, it is helpful in pattern recognition applications for detecting features [19]. The obtained polygon allows the detection of features such as vertex angles and object symmetry.

A function that computes the polygons inscribed in a certain region can be useful in agriculture and livestock for a variety of applications [4, 26]. For instance, crop and pasture analysis (estimating the amount of land available for cultivation and grazing; planning for crop and pasture rotation, land management planning). Additionally, crop growth (study of crop growth and yield), and animal tracking (monitoring their location and movement, prevention of escapes, surveillance, etc.). Inscribed polygons can be used in medicine for image analysis, identifying regions of interest, and validating image processing algorithms [2, 6, 16, 29, 31]. Moreover, satellite images can be analyzed for the optimal placement of solar panels in the photovoltaic industry, including rooftop solar panels and farm solar panels [15].

In many Computer Vision applications, it is common for algorithms to focus on specific areas of an image, known as Regions of Interest (ROI). Often, these regions of interest are obtained by users manually delineating areas or through semi-automatic algorithms that generate irregularly shaped areas. When operations are performed on neighbors or convolutions at the edges of these ROIs, pixels outside the region are sometimes considered, which can lead to undesirable results. To ensure that all pixels belong to the region of interest, it is necessary to find inscribed polygons within these regions. Traditionally, easily recognizable geometric figures are used, such as rectangles, squares or circles. Geometric optimization problems include the problem of packing spheres into a rectangular box [8] or finding the smallest parallelogram that encloses a convex polygon [30]. Thus to calculate the solution, an efficient algorithm is required to explore the entire search space.

In real-world applications, the search for the best inscribed contour is most commonly restricted to closed polygons containing forbidden holes, points or segments, that is, finite groupings of aligned points. Fig. 1 shows the initial problem, the closed contour C (region of interest), in which forbidden points (q_1, q_2, q_3), segments (S_1, S_2) and holes (H_1, H_2) have been defined. This significantly limits the number of inscribed polygons that can be found. Thus, the proposed problem can be formulated as follows: given a fixed number k, compute the simple k-gon with maximum area or perimeter contained in any closed contour C with r-points, $PointsP = \{q_1, \ldots, q_r\}$, t segments, $SegmentsP = \{S_1, \ldots, S_t\}$ and h holes, $HolesP = \{H_1, \ldots, H_h\}$.

Throughout this paper, all polygons under consideration are lattice polygons defined on a regular partition, which makes it possible to ground the algorithmic construction and to guarantee convergence through Pick's theorem.

2 Related work

The problem of locating the largest geometric figure through a set of obstacles in the plane is a well-known problem in the field of geometric optimization. Naaamad et al. [24] introduced the problem commonly known as the maximal empty rectangle problem or MER problem.

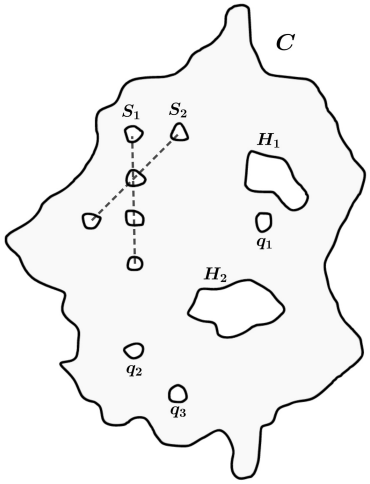


Figure 1: Closed contour C with three points, two segments and two holes

Given a rectangle A and a set S of n points of A, finding a maximum area rectangle (MER) that is totally contained in A, does not contain any point of S in its interior, and its edges are parallel to the edges of A. The problem was solved in $O(\min(n^2, R\log n))$, where R denotes the number of rectangles satisfying the above conditions. Later, the time complexity was improved to $O(R + n\log n)$ by Orlowski [27]. Another procedure to solve the problem that does not require the calculation of all MERs was presented in [1, 11] in $O(n\log^3 n)$ time and $O(n\log^2 n)$ time. Removing the axis-aligned condition, [10, 23] proved that the largest empty rectangle of arbitrary orientation for a given set of n points in the plane can be computed in $O(n^3)$ time. Recently, Bae and Yoon [7] considered the largest empty square of arbitrary orientation and proved that it can be found in $O(n^2\log n)$ time. Finally, Boyce et al. [9] generalized the problem and presented an algorithm for finding maximum area or perimeter convex k-gons with vertices k of the given n points in $O(kn\log n + n\log^2 n)$ time. For the case where the defective zones are segments, Nandy et al. [25] located the empty isothetic rectangle of maximum area given n arbitrary-oriented nonintersecting segments of finite length on a rectangular box in $O(n\log^2 n)$ time. Table 1 shows the computational costs of the previous papers when the obstacles are points or segments. The case of obstacles with holes has already been reviewed in Molano et al. [22], so we will not proceed to conduct any further revisions on this aspect. Beyond these approaches, other computational geometry problems have been explored that complement our work, such as the one proposed by Ausserhofer et al. [5].

In this paper, an open problem is addressed. First, instead of computing the triangle, rectangle or square with the maximum area or perimeter, our algorithm calculates the simple k-gon (k-sided polygon) with the largest area or perimeter. Second, the convex k-gon or rectangle of maximum area or perimeter can also be obtained. As a result, our algorithm enables the user to choose the type of k-gon to be found (triangle, quadrilateral, pentagon, hexagon, etc.) and the desired solution (maximum area or maximum perimeter) in $O(n^5k)$ computational time. Currently, no generic solution has yet been developed to compute k-sided polygons with the maximum area or perimeter contained within a closed contour with arbitrary obstacles. Furthermore, this work presents the algorithms, so that any researcher can easily understand it and adapt it to their favorite programming language (Java, Python). Additionally, a link is provided to a GitHub repository [20] containing all the source code developed in Java and Python.

Table 1: Computational cost.

Reference	Initial	Final	Computational cost
[24]	n-points	axis-parallel rectangle	$O(\min(n^2, R\log n))$
[27]	n-points	axis-parallel rectangle	$O(R + n\log n)$
[11]	n-points	axis-parallel rectangle	$O(n\log^3 n)$
[1]	n-points	axis-parallel rectangle	$O(n\log^2 n)$
[23]	n-points	rectangle	$O(n^3)$
[10]	n-points	rectangle	$O(n^3)$
[7]	n-points	square	$O(n^2\log n)$
[9]	n-points	convex k-gon	$O(kn\log n + n\log^2 n)$
[25]	n non intersecting segments	axis-parallel rectangle	$O(n\log^2 n)$

3 Definitions

3.1 Lattice Polygon

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, with $a, b, c, d \in \mathbb{Z}$, that satisfy:

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

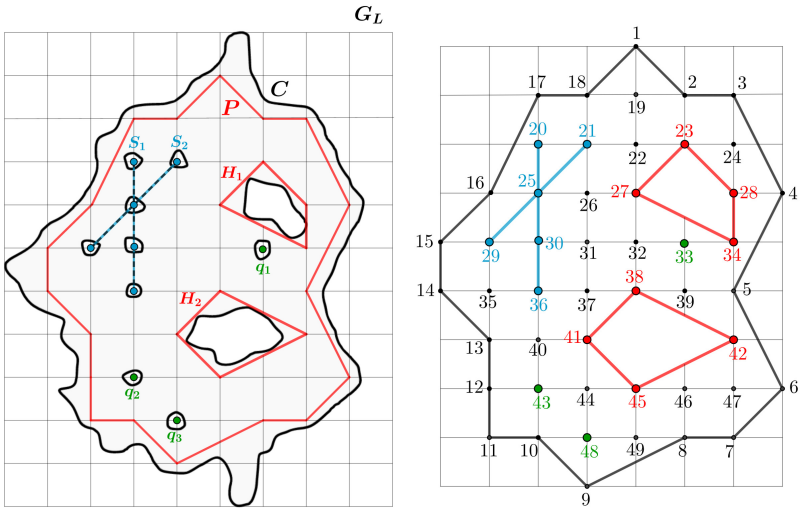
2. The connections between consecutive vertices are not necessarily established in the eight directions, $\pi k/4$, $k = 0, \ldots, 7$.
3. The edges of the polygon do not intersect except at their vertices.

We denote by $G_L = \{(x_i, y_j) \colon 0 \le i \le r, 0 \le j \le s\}$, 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 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 $\Pi \preceq \dot{\Pi}$.

Fig. 2 shows the lattice polygon constructed from the initial problem seen in Fig. 1.

If $\#$ represents the cardinality of the set and $P = \partial P \cup \imath P$ where ∂P is the family consisting of boundary nodes of P and its complementary in P, $\imath P$, the interior points, then $\#(P) = N = n + o \simeq \lambda n, \lambda \in \mathbb{N}$ with $\#(\partial P) = n$ and $\#(\imath P) = o$. In the same way, if $H_i = \partial H_i \cup \imath H_i$ with $\#(\partial H_i) = m_i$, $\#(\imath H_i) = r_i$, then $\#(H_i) = M = m_i + r_i \simeq \mu m_i, \mu \in \mathbb{N}$. Furthermore, let $m = \max\{m_1, m_2, \ldots, m_h\}$ and $s = \max\{s_1, \ldots, s_t\}$ where $\#(S_i) = s_i$, $1 \le i \le t$.

We compute the maximum area or perimeter simple k-gon in a lattice polygon P with r points, t segments $\{S_1, \ldots, S_t\}$ and h holes $\{H_1, \ldots, H_h\}$ and coordinates belonging to the



(a) Closed contour C on a regular partition with partition size L

(b) Lattice polygon P with three points, two segments and two holes on a regular partition with partition size L

Figure 2: Construction of the lattice polygon

square grid G_L, we use the following sets:

$$\begin{cases} Points = P = \{p_1, p_2, \ldots, p_{49}\},\ \#(P) = N = 49 \\ Polygon = \{p_1, p_2, \ldots, p_{18}\} \\ PointsP = \{p_{33}, p_{43}, p_{48}\} \\ SegmentsP = \{S_1, S_2\},\ S_1 = \{p_{20}, p_{25}, p_{30}, p_{36}\}, \\ S_2 = \{p_{21}, p_{25}, p_{29}\} \\ HolesP = \{H_1, H_2\},\ H_1 = \{p_{23}, p_{28}, p_{34}, p_{27}\}, \\ H_2 = \{p_{38}, p_{42}, p_{45}, p_{41}\}. \end{cases}$$

3.2 Adjacency Matrix

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

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

In this context, an edge is defined as a line segment connecting two lattice points of P that lies entirely inside the polygon, without intersecting any segments, except at their endpoints, or crossing through holes in its interior. These edges constitute the possible candidates to become sides of the k-gons constructed in the subsequent algorithms. For example, in Fig. 2b we have $A(1,3) = 0$, since the segment $\overline{p_1 p_3}$ lies outside the polygon P, and $A(1,47) = 0$, since the segment $\overline{p_1 p_{47}}$ intersects several holes.

For Fig. 2, the adjacency matrix is represented as follows:

$$A = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ \vdots \\ 47 \\ 48 \\ 49 \end{array} \begin{array}{c} \begin{array}{ccccccc} 1 & 2 & 3 & \dots & 47 & 48 & 49 \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 & 0 & 0 \\ 1 & 0 & 1 & \cdots & 0 & 0 & 0 \\ 0 & 1 & 0 & \cdots & 1 & 0 & 0 \\ & \vdots & & \ddots & & \vdots & \\ 0 & 0 & 1 & \cdots & 0 & 1 & 1 \\ 0 & 0 & 0 & \cdots & 1 & 0 & 1 \\ 0 & 0 & 0 & \cdots & 1 & 1 & 0 \end{pmatrix} \end{array}$$

Due to the high number of points that a lattice polygon can have, it is necessary to develop an algorithm to calculate the matrix A.

4 Computing the Adjacency Matrix of a Lattice Polygon with r Points, t Segments and h Holes

Let C be a closed contour constrained by arbitrary obstacles and P a lattice polygon within C with r points, $\#(PointsP) = r$, t segments, $SegmentsP = \{S_1, \dots, S_t\}$ and h holes, $HolesP = \{H_1, \dots, H_h\}$, and coordinates belonging to the square grid G_L.

We calculate the adjacency matrix of the lattice polygon P by considering only the points that define it as parameters, $P = points$. It uses two algorithms, *IntersectionPol (l)* and *Matrix(points)*.

4.1 Segment-Polygon Intersection (Algorithm 1)

We compute by Algorithm 1 when the intersection of a segment l with *Polygon*, *SegmentsP* or *HolesP* is possible. It returns *true* if the intersection is not allowed and *false* otherwise.

To facilitate the understanding of the algorithm, it has been divided into three blocks corresponding to each of the sets mentioned above. Each block explains the algorithm's function and analyzes the associated computational cost. This structure provides a more comprehensive and clear view of the algorithm as a whole.

Block 1: Polygon (Lines 2–6): The algorithm starts by first calculating the *Polygon* sides using the function *SidesPol(poly)* in $O(n)$ time (Line 2) and then applies function *IntersectionPoly(l, s, poly, num)* (Line 4), to find out if the intersection of segment l with any Polygon side is not allowed. If so, the algorithm terminates and returns *true* (Line 5). The computational cost of this block is $O(n^2)$ by the loop (Line 3) computed in $O(n)$ time and the function *IntersectionPoly(l, s, poly, num)* in $O(n)$ time.

If all intersections are possible (Line 7), Block 2 is carried out.

m_1 and m_2 are auxiliary points defined around the intersection point p, obtained by shifting p by a small value ϵ. They are used to determine whether the intersection lies inside or outside the polygon P or one of its holes, and thus to decide if the intersection is admissible. The procedure for constructing these points is part of the algorithm *IntersectionPoly*, described in detail in Molano et al. [22].

Fig. 3 shows two intersections. If the algorithm *PointIn(p, poly)* computes if a given point (p) is inside or on a side of *poly*, and *AlignedPol(p, poly)* is used to check if a point p

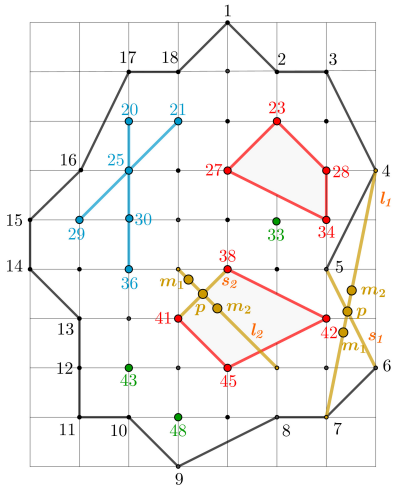


Figure 3: Algorithm 1: Segment-Polygon intersection

lies on one of the sides *poly*, then the first intersection, $l_1 \cap s_1$, is not possible for *Polygon* because *PointIn(m_2, Polygon)* = *false*. The second intersection, $l_2 \cap s_2$, is not possible for *HolesP*[2] because *PointIn(m_2, HolesP*[2]*)* = *true* and *AlignedPol(m_2, HolesP*[2]*)* = *false*. In this situation, *IntersectionPol*(l_1) = *true* and *IntersectionPol*(l_2) = *true*.

Block 2: SegmentsP (Lines 8–17): The algorithm checks if there is an intersection between the segment l and any of the segments of *SegmentsP*, using the functions *Intersection(l, s)* (Line 14), in $O(1)$ time, which returns *true* if the segments intersect and *false* if they do not; function *Intersectionp(l, s)* (Line 15), in $O(1)$ time, which computes the intersection point of the two nonconsecutive segments; and function *Direction(p,q,r: Points)* in $O(1)$ time, which returns the orientation of the three points [12]. The computational cost of this block is $O(t)$ by the loop (Line 9) computed in $O(t)$ time.

Collinear segment intersections are possible and have been taken into account in the algorithms, since a side of a k-gon may coincide with or be contained in the boundary of P, in a segment, or in the side of a hole.

If all intersections are possible (Line 18), Block 3 is carried out.

Block 3: HolesP (Lines 19–30): The algorithm checks segment l with all sides of the holes: *HolesP*[1], *HolesP*[2], ... ,*HolesP*[h] (Lines 20, 24) and perform the same process as done for *Polygon*. If for some *HolesP*[i], $1 \leq i \leq h$, the intersection is not allowed, the algorithm terminates, returns *true* (Line 22) and is not executed for the following holes: $i+1, \ldots, h$. The computational cost is $O(hm^4)$ by loop (Line 20) computed in $O(h)$ time, function *SidesPol(HolesP [i])* (Line 21) in $O(m)$ time, loop (Line 24) in $O(m)$ time, and algorithm *IntersectionPoly(l, s, poly, num)* (Line 25) in time $O(m^2)$. Here, $m = \max\{m_1, m_2, \ldots, m_h\}$, where $m_i = \#(H_i)$ denotes the number of vertices of the i-th hole, with # representing set cardinality.

Furthermore, a special scenario is addressed where the segment l is entirely contained within a hole. This specific case is outlined in Lines 28 to 30 and depicted in Fig. 4. Considering the hole *HolesP*[1] and the segment l_1 defined by the points $(27, 28)$, it is evident that *IntersectionPol* (l_1) = *true*, as the midpoint lies within *HolesP*[1] and is not part of any of its sides. The function *Midpoint*(l) returns the midpoint of the segment l. Similarly, this holds

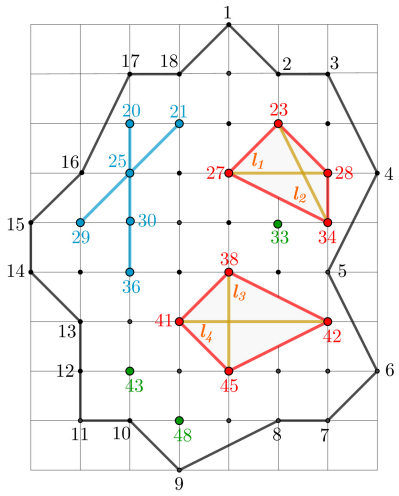


Figure 4: Segment contained in a hole

true for the segments formed by the points: $l_2 = (23, 34)$, $l_3 = (38, 45)$, and $l_4 = (41, 42)$.

The computational cost for this scenario is $O(hm^2)$ by the loop (Line 20) computed in $O(h)$ time, the function *PointIn(p, poly)* (Line 29) in $O(m)$ time, and the function *AlignedPol(p, poly)* (Line 29) in $O(m)$ time. Therefore, the total computational cost of Block 3 is $O(\max(hm^4, hm^2)) = O(hm^4)$.

Combining the computational costs of Blocks 1, 2, and 3, we arrive at the final computational cost of Algorithm 1: $O(\max(n^2, t, hm^4)) = O(n^2)$ if we assume that $\#(P) = N \simeq n \gg m$.

4.2 Computing the Adjacency Matrix (Algorithm 2)

In this algorithm, the input set $Points = P = \partial P \cup \imath P$ consists of all lattice points belonging to the boundary of the polygon P, $\partial P = Polygon$, and its interior, $\imath P$, including those associated with obstacles, whether points (*PointsP*), segments (*SegmentsP*), or holes (*HolesP*). The resulting adjacency matrix identifies all valid candidate edges of the final k-gon.

Algorithm 2, *Matrix(Points)*, is responsible for computing the adjacency matrix of the lattice polygon P. Initially, the adjacency matrix is initialized with all its terms equal to 0 ($matrix[i, j] = 0$) (Line 1), and then it determines all sides of the polygon (Line 2). Subsequently, two loops (Lines 3–4) are used to evaluate all pairs of lattice points of P contained in *Points*. The entry $matrix[i, j] = 1$ indicates that the segment $\overline{p_i p_j}$ is admissible, that is, it can potentially become a side of a k-gon. In particular, the condition `Length(Union(sides,{l})) = Length(sides)` (Line 6) is a set-membership test: it holds exactly when the candidate segment l is already one of the boundary sides of P returned by *SidesPol(Polygon)*. In this case, the algorithm assigns $matrix[i, j] \leftarrow 1$. Lastly, the assignment $matrix[j, i] = matrix[i, j]$ (Line 14) is carried out, as the adjacency matrix is symmetric and only the terms above the main diagonal have been computed thus far.

The computational cost of Algorithm 2 is determined in $O(n^5)$ time by loop (Line 3) computed in $O(n)$ time, loop (Line 4) in $O(n)$ time, Algorithm 1 (Line 9) in $O(n^2)$ time and the function *PointIn(p, poly)* (Line 12) in $O(n)$ time.

Algorithm 1: *IntersectionPol*(l)

Input: l: segment
Output: *true* if the intersection of segment l with *Polygon*, *SegmentsP* or *HolesP* is not allowed

```
1  isIntersecting ← false
2  sides ← SidesPol(Polygon)
3  for i ← 1 to Length(sides) do
4      isIntersecting ← IntersectionPoly(l, sides[i], Polygon, 0)
5      if isIntersecting = true then
6          break
7  if isIntersecting = false then
8      if SegmentsP ≠ ∅ then
9          for i ← 1 to Length(SegmentsP) do
10             if isIntersecting = true then
11                 break
12             n ← Length(SegmentsP[i])
13             s ← (Points[SegmentsP[i][1]], Points[SegmentsP[i][n]])
14             if Intersection(l, s) = true then
15                 p ← Intersectionp(l, s)
16                 if (Direction(s[1], l[1], p) ≠ 0 and Direction(s[1], l[2], p) ≠ 0 and
                     Direction(s[2], l[1], p) ≠ 0 and Direction(s[2], l[2], p) ≠ 0) then
17                     isIntersecting ← true
18 if isIntersecting = false then
19     if HolesP ≠ ∅ then
20         for i ← 1 to Length(HolesP) do
21             hol ← SidesPol(HolesP[i])
22             if isIntersecting = true then
23                 break
24             for j ← 1 to Length(HolesP[i]) do
25                 isIntersecting ← IntersectionPoly(l, hol[j], HolesP[i], 1)
26                 if isIntersecting = true then
27                     break
28             m ← Midpoint(l[0], l[1])
29             if (PointIn(m, HolesP[i]) = true and AlignedPol(m, HolesP[i]) = false then
30                 isIntersecting ← true)
31 return isIntersecting
```

5 Maximum Area or Perimeter Simple k-gon in a Lattice Polygon Among Arbitrary Obstacles

To obtain the maximum area or perimeter simple k-gon in a lattice polygon P with r points, t segments and h holes, we use the framework in Fig. 5 together with the summary of the main algorithms in Table 2.

Algorithm 2: *Matrix(Points)*

Input: Points $= P = \partial P \cup \imath P$
Output: Adjacency matrix

```
1  Initialize matrix, matrix[i,j] ← 0
2  sides ← SidesPol(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.1
10             then
11                 m ← Midpoint(Points[i], Points[j])
12                 if PointIn(m, Polygon) = true then
13                     matrix[i,j] ← 1
14 Symmetric matrix, matrix[j,i] ← matrix[i,j]
15 return matrix
```

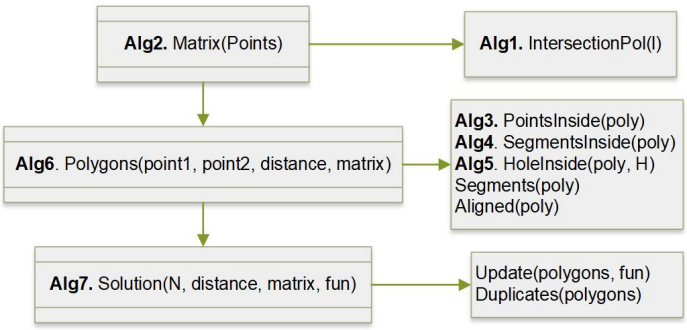


Figure 5: General framework

5.1 Obtaining Polygons (Algorithm 6)

We compute the sides of the polygons that are a certain distance from *point1* to *point2*. It has four parameters: *point1*, initial point; *point2*, final point; *distance*, which is used to determine all sides with this value (distance + 1 indicates the number k of sides of the polygon to be inscribed); and matrix, which is the adjacency matrix.

5.1.1 Algorithm 3. PointsInside(poly)

Given a k-sided polygon, Algorithm 3 determines whether all points of *PointsP* are outside the polygon *poly* or on one of its sides. It utilizes the functions *PointIn(p, poly)* and *AlignedPol(p, poly)* which returns *true* if a point belongs to one of the sides of the polygon, computed in $O(k)$ time. If the first function is verified, *PointIn(p, poly)* = *true*, and the previous condition is not, *AlignedPol(p, poly)* = *false*, the algorithm terminates and returns the value *true*. The computational cost is $O(k^2 r)$ by loop (Line 2) computed in $O(r)$ time, the function *PointIn*

Table 2: Summary of the main algorithms and their functionality

Algorithm	Functionality
Alg. 1	Segment–obstacles intersection
Alg. 2	Computation of the adjacency matrix
Alg. 3	Position of all points (*PointsP*) with respect to polygon *poly*
Alg. 4	Position of all segments (*SegmentsP*) with respect to polygon *poly*
Alg. 5	Position of the hole H with respect to polygon *poly*
Alg. 6	Sides between point1 and point2 for a certain distance
Alg. 7	Simple k-gons with maximum area or maximum perimeter contained in P

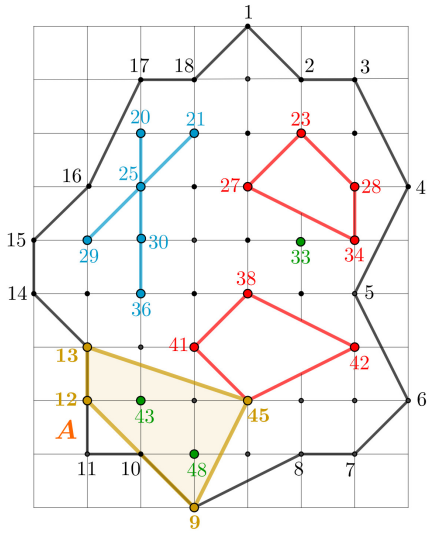


Figure 6: Algorithm 3. PointsInside(poly)

(Line 3) in $O(k)$ time and the function *AlignedPol* (Line 4) in $O(k)$ time.

In Fig. 6, polygon $A = (13, 45, 9, 12)$ has been constructed. The result obtained by applying Algorithm 3 is *PointsInside*(A) = true, as points 43 and 48 are located inside A, but not on its boundary.

Algorithm 3: PointsInside(poly)

Input: *poly*: k-sided polygon
Output: *false* if all the points of *PointsP* are outside the *poly* or on one of its sides

```
1 isPointInside ← false
2 for i ← 1 to Length(PointsP) do
3     if PointIn(Points[PointsP [i]], poly) = true then
4         if AlignedPol(Points[PointsP [i]], poly) = false then
5             isPointInside ← true
6             break
7 return isPointInside
```

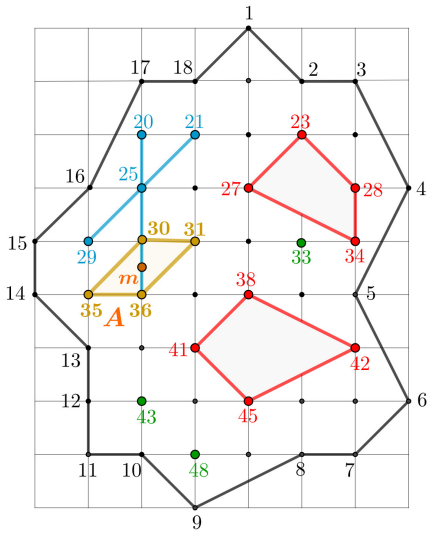


Figure 7: Algorithm 4. SegmentsInside(poly)

5.1.2 Algorithm 4. SegmentsInside(poly)

Algorithm 4 determines whether there is any intersection between the segments of *SegmentsP* and the polygon *poly*. To achieve this, the algorithm examines each segment (Line 2) and checks if the midpoint of any section of that segment lies exclusively inside *poly* (Lines 7–9). If this condition is met for any segment, the algorithm terminates and returns *true* (Line 10). The computational cost is $O(k^2st)$ by the loops in Lines 2 and 7 computed in $O(t)$ and $O(s)$, respectively, the function *PointIn* (Line 9) in $O(k)$ time and the function *AlignedPol* (Line 9) in $O(k)$ time.

Fig. 7 shows a polygon A with vertices $(30, 31, 36, 35)$. By analyzing the figure, we can observe that polygon A intersects segment S_1. When evaluating the algorithm *SegmentsInside(A)*, we obtain the value *true*, since the midpoint m is in the interior of polygon A but not in its boundary.

Algorithm 4: SegmentsInside(poly)

Input: *poly*: k-sided polygon
Output: *false* if all the segments of *SegmentsP* are outside the *poly* or on one of its sides

```
1  isSegmentInside ← false;
2  for i ← 1 to Length(SegmentsP) do
3      if isSegmentInside = true then
4          break
5      SegmP ← SegmentsP [i]
6      Insert(SegmP, SegmentsP [i][1])
7      for j ← 1 to Length(SegmP[i])-1 do
8          m ← Midpoint(Points[SegmP[i][j]], Points[SegmP[i][j+1]])
9          if (PointIn(m, poly) = true and AlignedPol(m, poly) = false) then
10             isSegmentInside ← true
11             break
12 return isSegmentInside
```

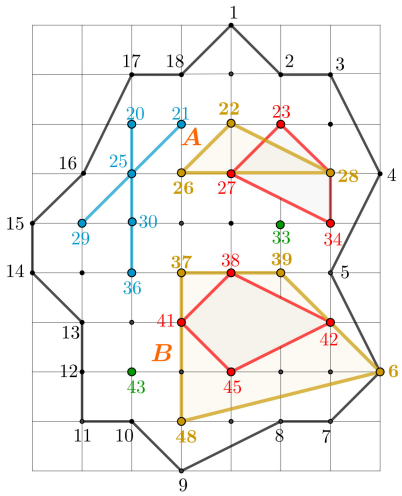


Figure 8: Algorithm 5: Hole contained in a k-sided polygon

5.1.3 Algorithm 5. HoleInside(poly, H)

Algorithm 5 determines whether a hole H is contained within or intersects with the polygon *poly*. To achieve this, it calculates the midpoint of each side and determines if that value is inside *poly* but not on its boundary. If so, the algorithm terminates and returns *true*. The computational cost is $O(k^2m)$ time by loop (Line 4) computed in $O(m)$ time, the function *PointIn* (Line 6) in $O(k)$ time and the function *AlignedPol* (Line 6) in $O(k)$ time.

Algorithm 5: HoleInside(poly, H)

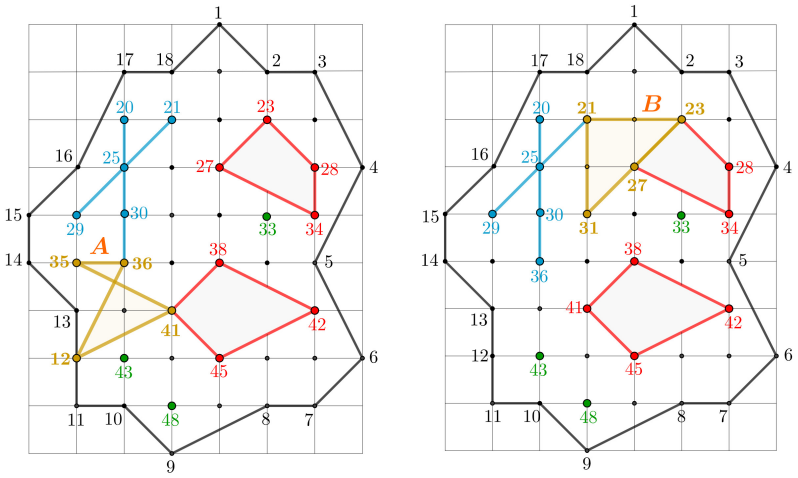
Input: *poly*: k-sided polygon, H: *HolesP*
Output: *false* if the hole H is not contained or not intersect the k-sided polygon and *true* otherwise

```
1 isHoleInside ← false
2 hol ← H
3 Insert(hol, H[1])
4 for i ← 1 to Length(hol) − 1 do
5 |   m ← Midpoint(Points[hol[i]], Points[hol[i+1]])
6 |   if (PointIn(m, poly) = true and AlignedPol(m, poly) = false) then
7 |   |   isHoleInside ← true
8 |   |   break
9 return isHoleInside
```

Fig. 8 shows two polygons. The first one, $A = (22, 28, 26)$, cannot be included in the final solution as it intersects with hole H_1. Similarly, polygon $B = (37, 39, 6, 48)$ cannot be as a valid solution since hole H_2 lies within it.

5.1.4 Algorithm 6. Polygons(point1, point2, distance, matrix)

To enhance comprehension of Algorithm 6, it has been organized into four distinct blocks. These blocks elucidate the algorithm's functionality and assess the associated computational cost.



(a) Intersection of segments (b) Consecutive aligned points

Figure 9: *Segments(poly)* and *Aligned(poly)* functions

Block 1: (Lines 3–13): This block generates a complete set of edges connecting two given points, with *point1* as the starting point and *point2* as the endpoint. The process entails identifying all edges that are at a distance of one unit from *point1*. Subsequently, the function iteratively repeats this process until all edges up to the desired distance are computed. The computational cost of this block is determined by three loops (Lines 3, 5, 8), each with a different time complexity: $O(k)$, $O(n^2)$, and $O(n)$, respectively. Hence, the computational cost of Block 1 is $O(n^3k)$.

Block 2: (Lines 14–17): The "temp1" set yields multiple solutions, and only those solutions whose final point coincides with *point2* are selected. To ensure that the chosen solutions form a valid polygon, two algorithms are employed: *Segments(poly)* and *Aligned(poly)*. The first algorithm determines whether two segments (edges) intersect or not, in $O(k^2)$ time, while the second algorithm, in $O(k)$ time, returns 0 if three consecutive points of the *poly* are aligned and otherwise returns a non-zero value. The computational cost is determined by the loop in Line 15 in $O(n^2)$ time and the above algorithms, therefore obtaining a computational cost of $O(n^2k^3)$ time.

In Fig. 9a, it can be observed that the polygon $A = (35, 36, 12, 41)$ cannot form a quadrilateral. Similarly, in Fig. 9b, it is shown that $B = (21, 23, 27, 31)$ also cannot be a quadrilateral, as points 23, 27, and 31 are aligned, indicating that it is in fact a triangle.

Block 3: (Lines 18–20): All solutions that do not satisfy the conditions set by the algorithms *PointsInside(poly)* and *SegmentsInside(poly)* are eliminated. The first algorithm returns *false* if all points of *PointsP* are outside a polygon of *temp2* or lie on any of its sides, while the second algorithm also returns *false* if there is no intersection between any of the segments of *SegmentsP* with a polygon of *temp2*. These algorithms have a computational cost of $O(k^2r)$ and $O(k^2st)$, respectively. The computational cost is determined by the loop in Line 18, which runs in $O(n^2)$ time, in addition to the mentioned algorithms. Therefore, the total cost is $O(n^2k^4rst)$.

Block 4: (Lines 21–27): From all the polygons obtained in Line 20, we remove those containing one or more holes. The computational cost is determined by the following loops: Line 21 computed in $O(n^2)$ time; Line 23 computed in $O(h)$, where P is a lattice polygon with h holes; Line 24 computed in $O(k^2m)$ by Algorithm 5. Thus, the computational cost is $O(n^2k^2hm)$.

Therefore, Algorithm 6 can be solved in $O(n^3k)$ time, as $\max(n^3k, n^2k^3, n^2k^4rst, n^2k^2hm) = n^3k$ if $n \gg k$ is assumed.

Algorithm 6: Polygons(point1, point2, distance, matrix)

Input: point1, point2 $\in$ *Points*, distance $\in \mathbb{N}$, matrix: adjacency matrix
Output: sides between point1 and point2 for a certain distance

```
1  temp1 ← {{point1}};  temp3, sides ← ∅
2  if Matrix(point1, point2) = 1 then
3      for i ← 1 to distance do
4          temp2 ← ∅
5          for j ← 1 to Length(temp1) do
6              last ← temp1[j][i]
7              if last ≠ point2 then
8                  for k ← 1 to N do
9                      if (Matrix(last,k) = 1 and Length(Union(temp1[j],k)) ≠
                         Length(temp1[j])) then
10                         Insert(temp2, Insert(temp1[j],k))
11                         Delete(temp1[j], temp1[j][Length(temp1[j])])
12         temp1 ← ∅
13         temp1 ← temp2
14     temp2 ← ∅
15     for i ← 1 to Length(temp1) do
16         if (temp1[i][distance + 1] = point2 and Segments(temp1[i]) = false and
             Aligned(temp1[i]) ≠ 0) then
17             Insert(temp2, temp1[i])
18     for i ← 1 to Length(temp2) do
19         if (PointsInside(temp2[i]) = false and SegmentsInside(temp2[i]) = false) then
20             Insert(temp3, temp2[i])
21     for i ← 1 to Length(temp3) do
22         h ← 0
23         for j ← 1 to Length(HolesP) do
24             if Holes(temp3[i], HolesP [j]) = false then
25                 h ← h+1
26         if h = Length(HolesP) then
27             Insert(sides, temp3[i])
28 return sides
```

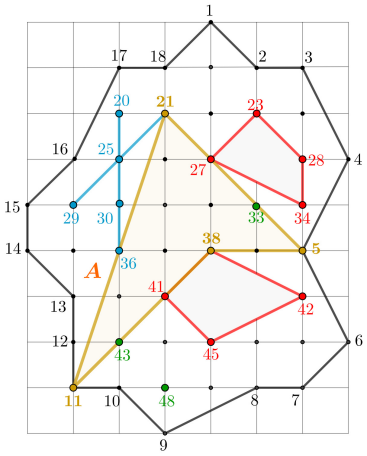


Figure 10: Maximum area 4-gon; $A = (5, 38, 11, 21)$

5.2 Maximum Area or Perimeter (Algorithm 7)

We compute the maximum area or perimeter simple k-gon inscribed in a lattice polygon P with r points, t segments and h holes. It is the main program of the proposal. It has four parameters: $N = \#(P)$, number of points of the lattice polygon P; $k = distance + 1$, where k represents the number of sides of the polygon to be computed; matrix, which is the adjacency matrix; and fun, which indicates whether to calculate the area (fun = 0) or the perimeter (fun = 1). Additionally, with some minor modifications to the algorithm, it is possible to calculate all the solutions for convex polygons, *SolutionConvex(N, distance, matrix, fun)* and for rectangles, *Rectangle(N, matrix, fun)*.

It utilizes Algorithm 6, *Polygons(point1, point2, distance, matrix)*, and two auxiliary algorithms, *Update(polygons, fun)* and *Duplicates(polygons)*. The first one computes the largest area or perimeter polygons, depending on the value of the *fun* parameter, 0 for the largest area and 1 for the perimeter, in $O(n^2k)$ time. The second algorithm ensures that no duplicate solutions occur during polygon analysis and can be solved in $O(n^4k \log k)$ time.

We examine the case of a simple 4-gon with the solution $(5, 38, 11, 21)$, as depicted in Fig. 10. It is apparent that for each k-gon, there are $2k$ solutions with the same area and perimeter. To address this challenge, the *Duplicates* algorithm selects a representative solution from each set of identical solutions.

The computational cost of the algorithm *Solution(N, distance, matrix, fun)* is initially determined by two nested loops (Lines 2–3) that run in $O(n^2)$ time and the algorithm *Polygons(point1, point2, distance, matrix)*, which takes $O(n^3k)$ time. This results in a computational cost of $O(n^5k)$. Finally, the algorithms *Update(polygons, fun)* and *Duplicates(polygons)* are employed to obtain the desired solution. Therefore, the computational cost of our proposed algorithm for computing the maximum area or perimeter of a simple k-gon inscribed in a lattice polygon P with r points, t segments, and h holes is $O(n^5k)$, as $\max(n^5k, n^2k, n^4k \log k) = n^5k$ if $n \gg k$.

The source code for the seven algorithms used has been developed in Python and Java and is available in a GitHub repository [20]. This allows any researcher to examine the code in detail, understand how the algorithms and functions work, and modify or improve the code as needed.

Algorithm 7: *Solution(N, distance, matrix, fun)*

Input: N: number of points of P, distance $\in \mathbb{N}$, matrix: adjacency matrix, fun = $\{0, 1\}$

Output: Simple k-gons with maximum area or maximum perimeter contained in P

```
1 polig, solution ← ∅
2 for i ← 1 to N-1 do
3 |   for j ← i + 1 to N do
4 |   |   if Polygons(i, j, distance, matriz) ≠ ∅ then
5 |   |   |   Insert(polig, Polygons(i, j, distance, matrix))
6 solution ← Duplicates(Update(polig, fun))
7 return solution
```

5.3 Experimental Results

Through the algorithm *Solution(N, distance, matrix, fun)*, we have demonstrated the capability to calculate the maximum area or perimeter of a simple k-gon contained within a lattice polygon P with r points, t segments, and h holes. Now, let's explore some potential solutions (Fig. 12, Fig. 13 and Fig. 14) for the problem depicted in Fig. 2 (refer to Fig. 11 and Table 3). Additionally, we'll consider a new example (Fig. 15 and Table 4), showcasing the solutions of the simple and convex k-gons with maximum area and perimeter.

In order to facilitate the interpretation of the results, the solutions highlighted in red in Tables 3 and 4 indicate the specific k-gons that are illustrated in Figs. 12–14 and 16–17. This convention allows readers to easily connect the numerical data with the corresponding graphical representations.

Upon analyzing the results presented in Tables 3 and 4, an increase in both the area and perimeter of simple polygons is observed as the number of sides increases. However, in certain specific cases of convex polygons, solutions could not be found. This is illustrated by the calculations for the heptagon and octagon, as shown in Tables 3 and 4. These tables also include a column with computational time in seconds. All experiments were performed on a computer with an Intel Core i9 processor, 64 GB RAM, and an Nvidia GeForce RTX 3070 8 GB graphics card.

Fig. 11. $PointsP = \{p_{33}, p_{43}, p_{48}\}$, $SegmentsP = \{S_1, S_2\}$, where $S_1 = \{p_{20}, p_{25}, p_{30}, p_{36}\}$, $S_2 = \{p_{21}, p_{25}, p_{29}\}$ and $HolesP = \{H_1, H_2\}$, where $H_1 = \{p_{23}, p_{28}, p_{34}, p_{27}\}$, $H_2 = \{p_{38}, p_{42}, p_{45}, p_{41}\}$.

Fig. 14 illustrates how Algorithm 7, *Solution(N, distance, matrix, fun)*, can be utilized to compute the k-convex polygon with maximum area or perimeter. This adapted algorithm, denoted as *SolutionConvex(N, distance, matrix, fun)*, enables us to compare solutions for both simple and convex polygons.

Fig. 15. $PointsP = \{p_{35}, p_{36}, p_{40}, p_{41}, p_{53}\}$, $SegmentsP = \{S_1, S_2, S_3, S_4, S_5\}$, where $S_1 = \{p_{27}, p_{34}\}$, $S_2 = \{p_{30}, p_{38}, p_{45}\}$, $S_3 = \{p_{32}, p_{44}, p_{54}\}$, $S_4 = \{p_{44}, p_{52}, p_{56}\}$, $S_5 = \{p_{43}, p_{47}\}$ and $HolesP = \varnothing$.

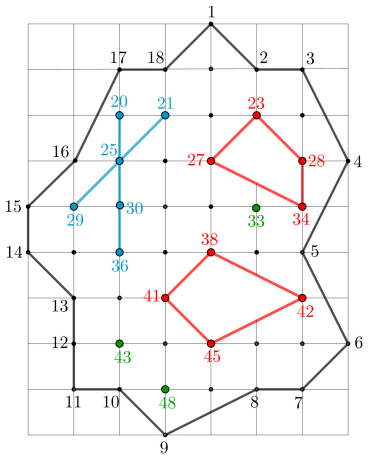
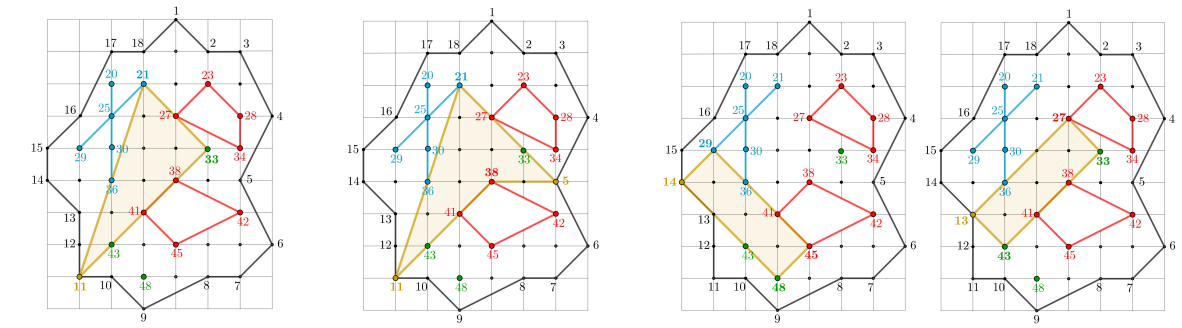
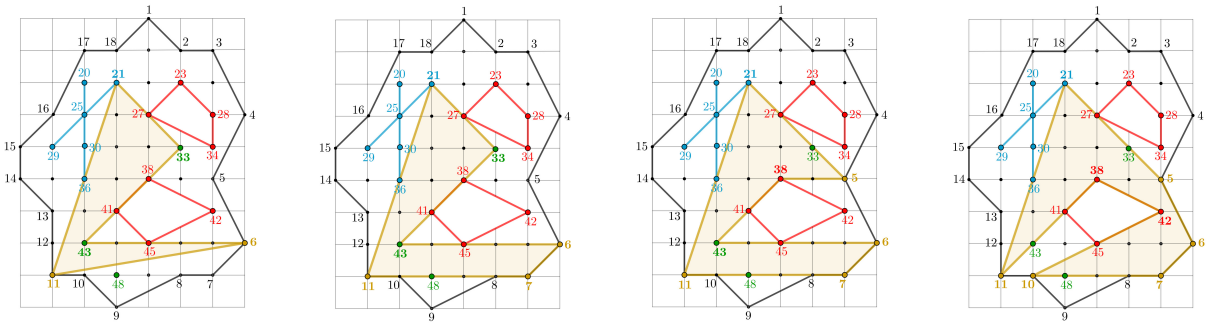


Figure 11: Example: Lattice polygon P, $N = 49$



(a) Triangle (b) Quadrilateral (c) Rectangle

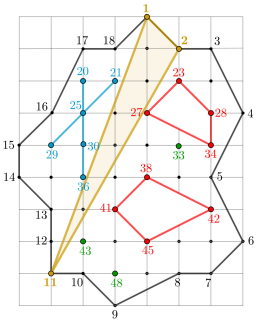


(d) Pentagon (e) Hexagon (f) Heptagon (g) Octagon

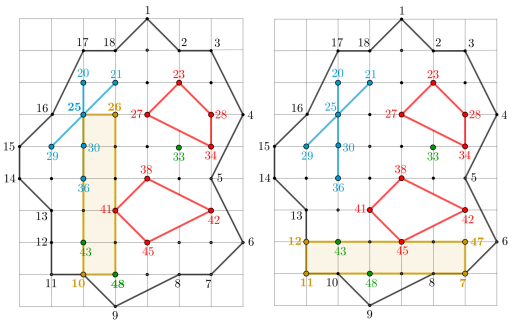
Figure 12: Example 1: Maximum-area simple k-gon

6 Maximum Area or Perimeter Simple k-gon in a Closed Contour Constrained by Arbitrary Obstacles

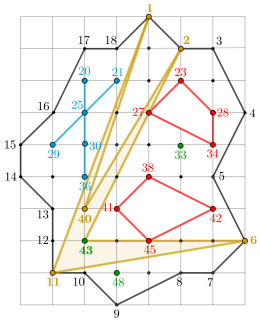
By algorithm *Solution(N, distance, matrix, fun)*, we demonstrate the feasibility of computing the maximum-area or perimeter simple k-gon contained in a lattice polygon P with r points, t segments and h holes. Now, through Theorem 6.1, we observe that the area of a closed contour C constrained by arbitrary obstacles can be calculated by the inscribed limit area within the lattice polygon P, constructing finer partitions. Moreover, we achieve the paper's objective, as the simple k-gon contained in P is the maximal in area or perimeter within C



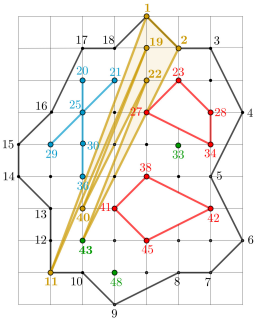
(a) Triangle



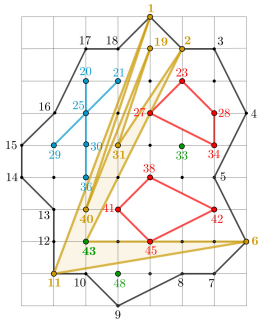
(b) Rectangle



(c) Hexagon

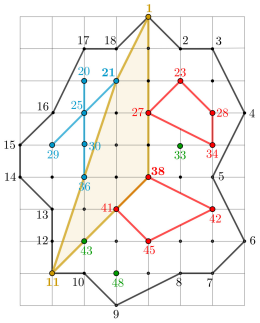


(d) Heptagon

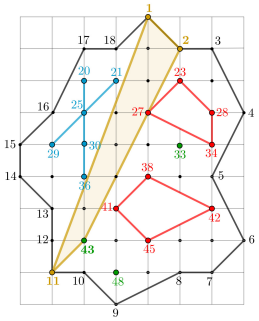


(e) Octagon

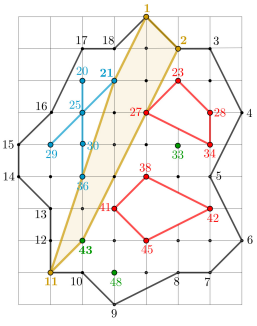
Figure 13: Example 1: Maximum-perimeter simple k-gon



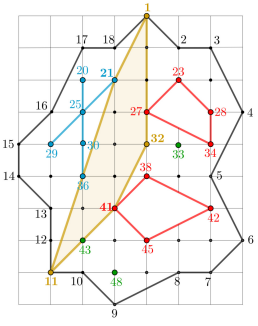
(a) Quadrilateral area



(b) Quadrilateral perimeter



(c) Pentagon area



(d) Pentagon perimeter

Figure 14: Example 1: Maximum-area and perimeter convex k-gon

Table 3: Solutions Example 1: Maximum-area and perimeter simple and convex k-gon, $N = 49$

***k*-gon**	**Maximum-area simple (Fig. 12)**	**Area**	**Time (s)**
Triangle	(11,33,21)	8	0.01
Quadril.	(5,38,11,21)	9	0.23
Rectangle	(14,48,45,29), (13,27,33,43)	6	0.32
Pentagon	(6,43,33,21,11), (7,43,33,21,11)	10.49	1.88
Hexagon	(6,43,33,21,11,7)	13	11,71
Heptagon	(5,38,43,6,7,11,21), (6,21,11,38,42,10,7)	14	91.10
Octagon	(5,21,11,38,42,10,7,6)	15.50	830.83
***k*-gon**	**Maximum-perimeter simple (Fig. 13)**	**Area**	**Time (s)**
Triangle	(1,11,2)	18.01	0.01
Quadril.	(1,40,2,11)	28.75	0.21
Rectangle	(10,25,26,48), (12,11,7,47)	12	0.33
Pentagon	(1,11,19,43,2)	30.63	1.89
Hexagon	(1,40,2,43,6,11)	38.48	10.99
Heptagon	(1,11,19,40,22,43,2)	39.55	89.68
Octagon	(1,40,19,31,2,43,6,11)	44.84	831.67
***k*-gon**	**Maximum-area convex (Fig. 14)**	**Area**	**Time (s)**
Quadril.	(1,38,11,21)	8.50	0.17
Pentagon	(1,21,11,43,2), (1,32,41,11,21)	8	1.45
Hexagon	(1,27,41,10,36,21), (1,32,41,43,36,21)	7	10.46
	(1,38,41,40,36,21), (8,49,43,14,15,29), (10,41,27,22,21,25)		
	(10,41,27,19,21,30), (14,48,49,45,29,15), (19,32,41,43,30,21)		
	(19,38,41,40,30,21), (21,25,40,41,38,22), (21,25,43,41,32,22)		
Heptagon	∅	–	89.74
Octagon	∅	–	813.90
***k*-gon**	**Maximum-perimeter convex (Fig. 14)**	**Area**	**Time (s)**
Quadril.	(1,11,43,2)	18.07	0.17
Pentagon	(1,21,11,43,2)	18.09	1.15
Hexagon	(1,22,37,43,11,21)	17.35	10.21
Heptagon	∅	–	90.34
Octagon	∅	–	815.26

when P approaches C with finer partitions.

If $A(C)$ denotes the area of C, $A(C_0)$ the area of the outer lattice polygon C_0 and $A(H_j)$ is the area of each hole with $1 \leq j \leq h$, then:

$$A(C) = A(C_0) - \sum_{j=1}^{h} A(H_j)$$

Moreover, let R the rectangle of the minimum area that encloses the closed contour C_0 [13] and let Π be a regular partition of R with *partition size* L. We define *lower area* $\underline{A}(C_0, \Pi)$ and *upper area* $\overline{A}(H_j, \Pi)$ as the largest area lattice polygon P_0 contained in C_0 and the smallest area lattice polygon Q_j containing each H_j, $1 \leq j \leq h$, respectively, and both are built by

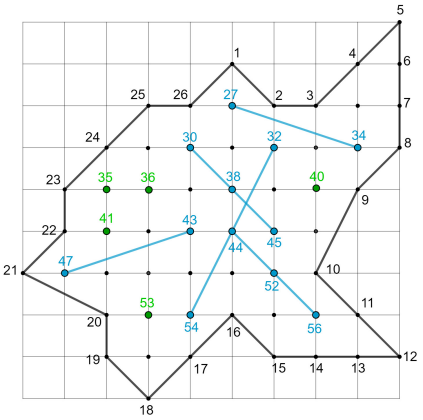
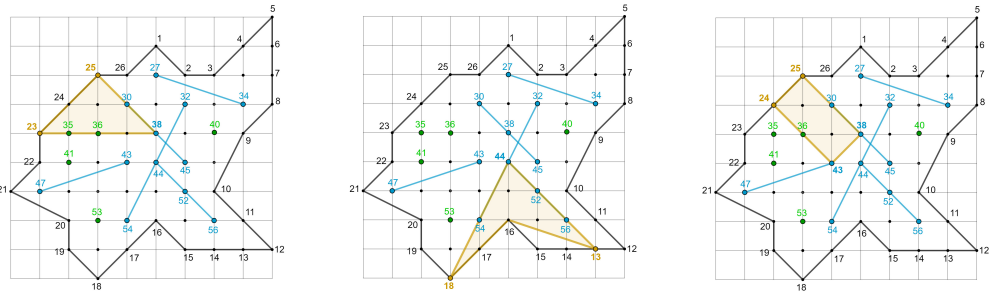
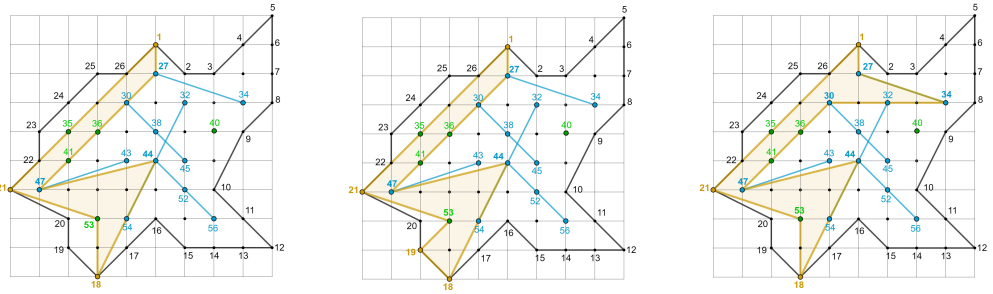


Figure 15: Example 2: Lattice polygon P, $N = 57$



(a) Triangle (b) Quadrilateral (c) Rectangle



(d) Heptagon (e) Octagon (f) Nonagon

Figure 16: Example 2: Maximum-area simple k-gon

points of G_L. By Pick's theorem [32],

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

Theorem 6.1. *Let C be a closed contour constrained by arbitrary obstacles 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.*

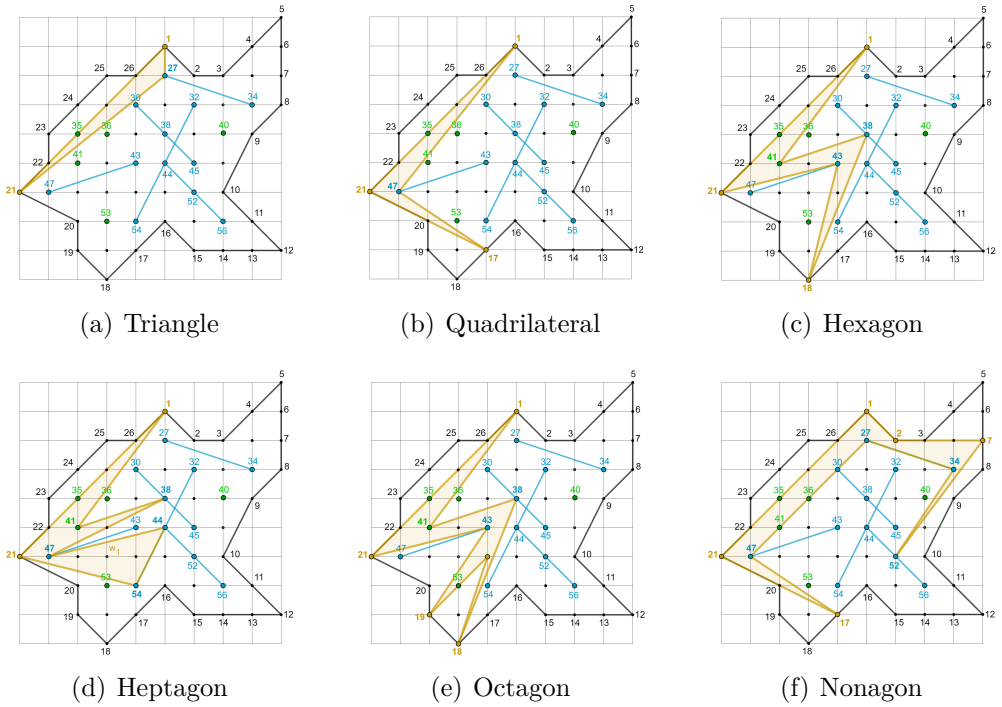
Proof. We consider a regular partition $\dot{\Pi}$ as finer than Π. Then, $\underline{A}(C_0, \Pi) \leq \underline{A}(C_0, \dot{\Pi})$ and $\overline{A}(H_j, \Pi) \geq \overline{A}(H_j, \dot{\Pi})$ with $1 \leq j \leq h$. Therefore, $\underline{A}(C_0, \Pi) - \sum_{j=1}^{h} \overline{A}(H_j, \Pi) \leq \underline{A}(C_0, \dot{\Pi}) -$

Table 4: Solutions Example 2: Maximum-area and perimeter simple and convex k-gon, $N = 57$

***k*-gon**	**Maximum-area simple (Fig. 16)**	**Area**	**Time (s)**
Triangle	(23,38,25), (25,47,27)	4	0.01
Quadril.	(13,44,18,16), (18,53,47,44), (25,47,30,34)	5	0.21
Rectangle	(24,43,38,25), (38,23,22,44)	4	0.27
Pentagon	(1,47,44,54,21)	6.50	1.03
Hexagon	(1,27,47,44,54,21)	8.50	4.70
Heptagon	(1,27,47,44,18,53,21)	10	25.17
Octagon	(1,27,47,44,18,19,53,21), (1,27,47,44,18,20,53,21) (1,38,30,47,44,18,53,21), (18,53,21,26,34,30,47,44)	11	160.62
Nonagon	(1,27,34,30,47,44,18,53,21), (1,38,30,47,44,18,19,53,21) (1,38,30,47,44,18,20,53,21), (1,41,30,44,18,53,47,43,21) (18,44,47,30,34,26,21,53,19), (18,44,47,30,34,26,21,53,20)	12	1125.91
***k*-gon**	**Maximum-perimeter simple (Fig. 17)**	**Perimeter**	**Time (s)**
Triangle	(1,27,21), (1,47,21)	14.47	0.01
Quadril.	(1,47,17,21)	21.55	0.23
Rectangle	(22,44,38,23)	10	0.32
Pentagon	(1,47,44,54,21)	23,96	1.01
Hexagon	(1,41,38,18,43,21)	28.86	4.26
Heptagon	(1,41,38,47,44,54,21)	30.18	26.12
Octagon	(1,41,38,18,50,19,43,21)	34.34	155.34
Nonagon	(1,21,17,47,27,34,52,7,2)	36.99	1098.23
***k*-gon**	**Maximum-area convex**	**Area**	**Time (s)**
Quadril.	(1,21,47,27), (20,54,44,47)	4.50	0.14
Pentagon	(20,54,44,43,47)	5	0.88
Hexagon	(15,14,56,52,51,16) (23,35,30,27,26,24), (24,35,36,30,26,25) (29,36,43,44,38,30), (38,43,53,57,54,44)	3	4.03
Heptagon	$\varnothing$	–	23.68
Octagon	$\varnothing$	–	151.19
***k*-gon**	**Maximum-perimeter convex**	**Area**	**Time (s)**
Quadril.	(1,27,47,21)	14.73	0.11
Pentagon	(18,44,38,43,53)	11.12	0.75
Hexagon	(23,35,30,27,26,24),(53,57,54,44,38,43)	9.30	3.99
Heptagon	$\varnothing$	–	22.67
Octagon	$\varnothing$	–	149.45

$\sum_{j=1}^{h} \overline{A}(H_j, \dot{\Pi})$. 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 Fig. 18 and Fig. 19, we present three partitions for the same closed contour C with three points, two segments, and two holes, with partition sizes: $L_1 = 1$, $L_2 = 1/2$, $L_3 = 1/4$, numbers of points: 49, 197, 804, and adjacency matrices: A, $\dot{A}$, $\ddot{A}$, respectively. It is evident that the area of the maximum simple 3-gon, 4-gon and rectangle increases as the partition becomes finer.



(a) Triangle (b) Quadrilateral (c) Hexagon

(d) Heptagon (e) Octagon (f) Nonagon

Figure 17: Example 2: Maximum-perimeter simple k-gon

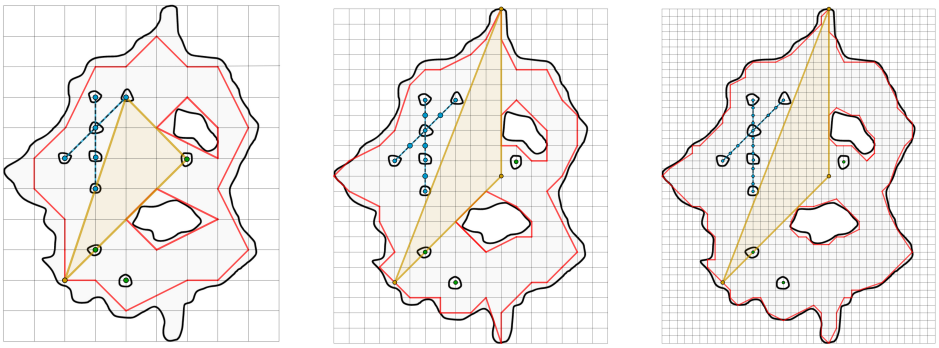
In addition to the increase in polygonal area, Figs. 18 and 19 also reflect the computational effort required as the resolution becomes finer. For the case $N = 49$, execution times are negligible (below one second). For $N = 197$, runtimes grow to the order of seconds, while for $N = 804$ they can reach several hours depending on the instance (e.g., up to 22044 s in Fig. 18f). A similar growth is observed in Fig. 19, where runtimes range from a few seconds to slightly over one thousand seconds. These values are consistent with the theoretical complexity $O(n^5k)$ and highlight the combinatorial nature of the problem. Moreover, memory usage follows the same trend.

7 Results and Discussion

Table 1 shows the computational cost of other scientific proposals that inscribe polygons in Regions of Interest. Although Algorithm 7 provides the final procedure for computing the maximum k-gon, it must be acknowledged that its computational cost is high. The algorithm systematically explores all pairs of points, which results in a number of operations that grows rapidly with the size of the dataset. This is not a methodological limitation, but rather an inherent feature of the problem when polygons with arbitrary numbers of sides are allowed.

In the existing literature, approaches with lower complexity achieve this by imposing strong restrictions on the type of polygons considered, most often limiting the search to axis-aligned rectangles, squares, or highly symmetric figures. Such simplifications undoubtedly reduce execution times, but they also compromise the generality that real-world applications frequently require.

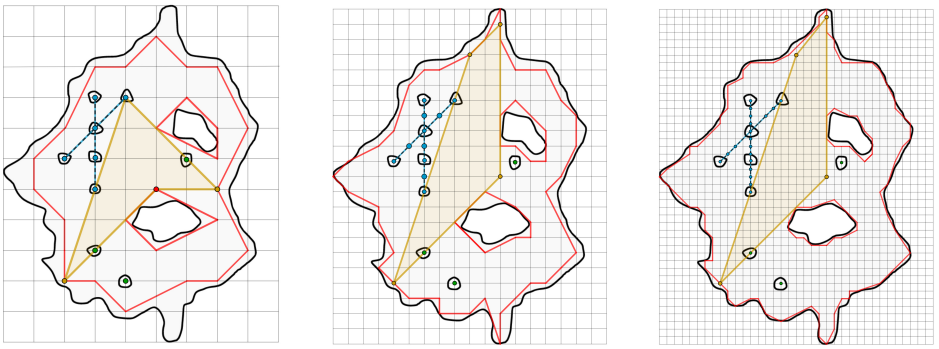
In medical contexts, for example, regions of interest often need to adapt to irregular anatomical shapes, such as tumors, which cannot be represented adequately by rectangles. In military and security applications, strategic zones in maps or satellite imagery typically



(a) Solution(49, 2, A, 0), Area = 8

(b) Solution(197, 2, $\dot{A}$, 0), Area = 9,63

(c) Solution(804, 2, $\ddot{A}$, 0), Area = 9,63

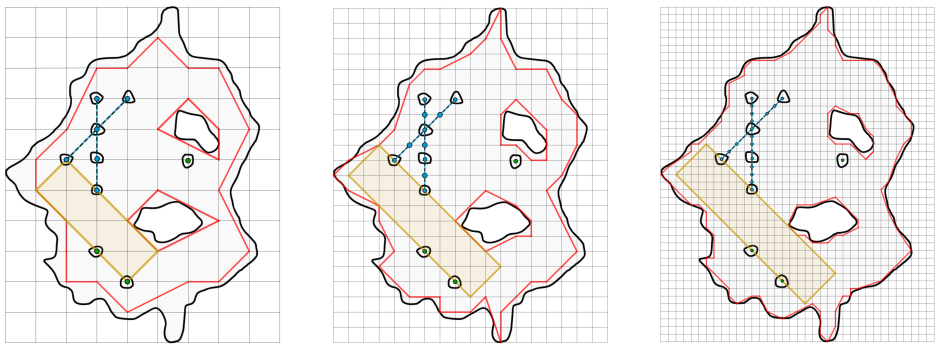


(d) Solution(49, 3, A, 0), Area = 9

(e) Solution(197, 3, $\dot{A}$, 0), Area = 11,25

(f) Solution(804, 3, $\ddot{A}$, 0), Area = 11,38

Figure 18: Maximum area simple 3-gon and 4-gon with different partition sizes



(a) Rectangle(49, A, 0), Area = 6

(b) Rectangle(197, $\dot{A}$, 0), Area = 8

(c) Rectangle(804, $\ddot{A}$, 0), Area = 8,50

Figure 19: Maximum area rectangle with different partition sizes

follow irregular boundaries shaped by natural geography or urban infrastructure. Likewise, in scientific fields ranging from geological exploration to robotics, the data to be analyzed rarely conform to simple predefined figures.

For these reasons, although Algorithm 7 entails a higher computational cost, it remains the only exact and general solution currently available. It addresses the problem in its full generality and provides a rigorous framework for applications where flexibility and precision are essential.

In this section, we present two illustrative examples to show how the proposed algorithm can be applied to different scenarios involving lattice polygons with obstacles. These cases are intended to illustrate the flexibility and broad applicability of the method.

It should be noted that the representation of obstacles, whether as points, segments, or holes, depends on the adopted convention and can affect both the possible results and the computational time. In this paper, we adhere to a consistent criterion: regions with appreciable area are treated as holes, finite aligned sets of points as segments, and negligible regions as isolated points. Without such rules, two users could model the same configuration differently, leading to distinct outcomes.

Furthermore, the type and number of obstacles directly influence the adjacency matrix used in Algorithm 7. For instance, in the case study analyzed here, the adjacency matrix is of order 49. In general, as the problem becomes more complex and the number of obstacles increases, the adjacency matrix tends to become sparser (with many zero entries), which reduces the number of candidate edges and may shorten the total computation time.

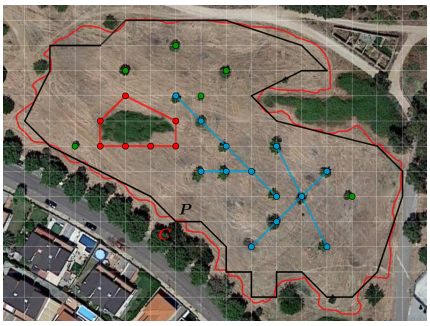
7.1 Construction of a Shopping Center

Constructing a building of any polygonal shape on a plot of land can be an interesting challenge for architects [18]. Although this shape can make the structure unique and attractive, it can also present problems during the design and building process. In Fig. 20a, the closed contour C of a plot of land has been extracted and then, the lattice polygon P on closed contour C has been constructed (Fig. 20b). Finally we construct the lattice polygon P (Fig. 20c) with $N = \#(P) = 111$, $Points = \{p_1, \ldots, p_{111}\}$, $Polygon = \{p_1, \ldots, p_{33}\}$, $PointsP = \{p_{34}, p_{35}, p_{36}, p_{37}, p_{38}, p_{39}\}$, $SegmentsP = \{S_1, S_2, S_3, S_4\}$ and $HolesP = \{H_1\}$. The objective was to design a shopping center of any polygonal shape with the maximum area. Upon the algorithm *Solution(N, distance, matrix, fun)* is applied to this problem, several potential solutions are discovered, which are illustrated in Fig. 21. These solutions demonstrate the effectiveness of the algorithm and the possibilities that can be achieved when designing buildings on polygonal plots.

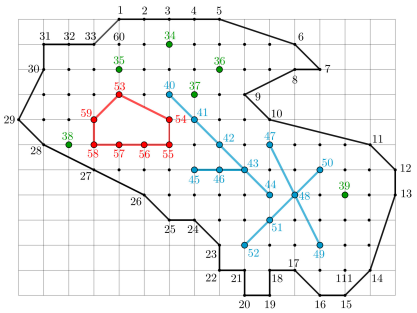
7.2 Medical Imaging

The acquisition of medical images is an important component in the diagnosis and treatment of various diseases, including brain tumors [3, 21]. In this particular case, an image of a primitive neuroectodermal tumor (PNET) [14] (Fig. 22a), a primary tumor of the central nervous system, was obtained. After obtaining the image of the PNET, the lattice polygon P on the closed contour C was created, as shown in Fig. 22b. Finally, the lattice polygon P (Fig. 22c) necessary to apply the algorithm *Solution(N, distance, matrix, fun)* was built, where $N = \#(P) = 86$, $Points = \{p_1, \ldots, p_{86}\}$, $Polygon = \{p_1, \ldots, p_{36}\}$, $PointsP = \{p_{37}, p_{41}, p_{75}, p_{86}\}$ and $SegmentsP = \{S_1, S_2, S_3, S_4, S_5\}$. In this context, the main objective was to compute the largest polygonal region with the maximum area to perform an accurate and detailed evaluation of the brain images. By applying the algorithm *Solution(N, distance, matrix, fun)*, it was possible to obtain different geometric solutions for simple and convex polygons. Fig. 23 provides an illustration of some of these solutions. Table 5 has been constructed to compare the areas obtained with simple and convex polygons, enhancing the understanding of the capabilities of the algorithm. This table further demonstrates that, for convex polygons, an increase in the number of sides does not imply an increase in area or perimeter.

(a) Closed contour C



(b) Lattice polygon P on closed contour C



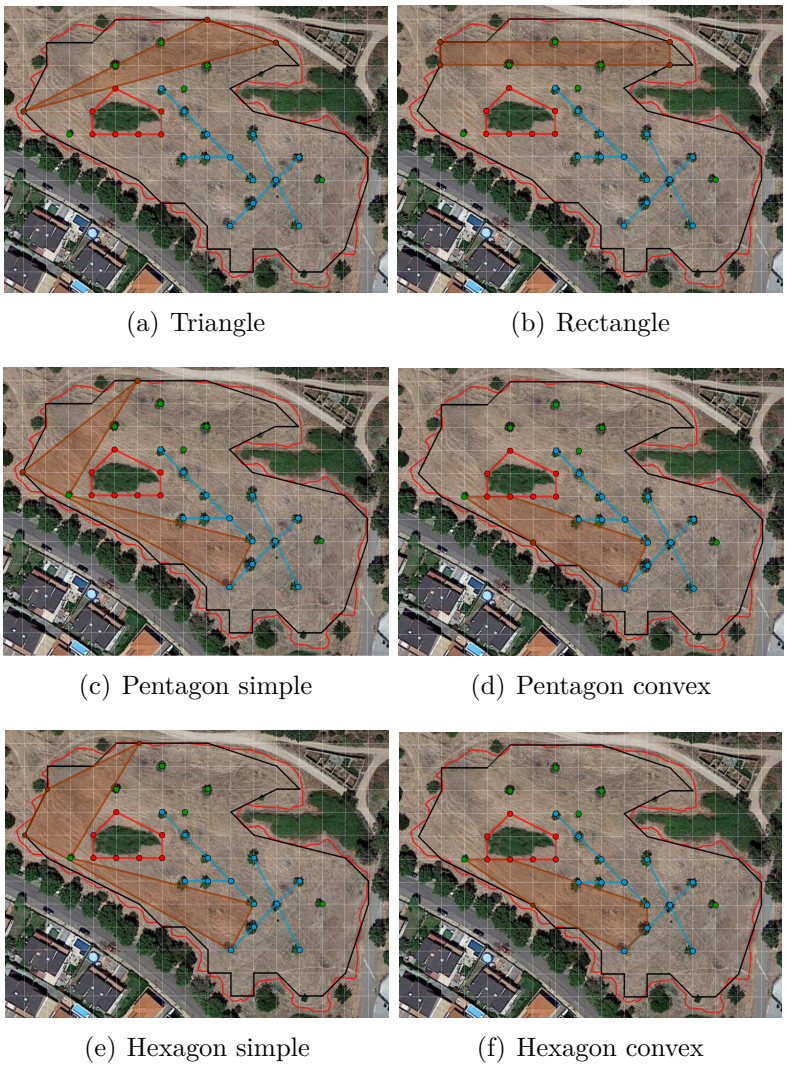
(c) Lattice polygon P

Figure 20: Practical application 1: Construction of a shopping center

Table 5: Solutions Practical application 2: Maximum-area simple and convex k-gon

***k*-gon**	**Maximum-area simple (Fig. 23)**	**Area**
Triangle	(3,85,18), (11,42,17), (35,82,57)	10
Rectangle	(8,15,19,59)	10
Quadrilateral	(14,52,35,42), (26,75,35,57)	11.50
Pentagon	(2,72,66,17,11), (11,42,39,59,17)	14.50
Hexagon	(4,60,44,57,82,35)	16.50
Heptagon	(14,52,44,57,82,35,42)	22.50
***k*-gon**	**Maximum-area convex (Fig. 23)**	**Area**
Quadrilateral	(14,42,40,15), (35,75,82,57)	11
Pentagon	(2,46,84,14,42), (24,56,36,30,25), (35,75,82,57,44)	11.50
Hexagon	(24,56,36,35,75,25)	12
Heptagon	(24,56,36,35,32,82,25), (24,69,50,36,35,75,25)	11.50

Additionally, a new partition finer than the original partition has been created to graphically demonstrate how a smaller partition size L results in a larger lattice polygon contained within the closed curve C (Fig. 24), which directly influences the final solution. This approach is particularly important in applications that demand high accuracy, such as evaluating the extent of areas of interest. As shown in the results presented in Table 6 and Fig. 25, using finer partitions leads to larger polygonal areas, thereby significantly improving the accuracy



(a) Triangle (b) Rectangle

(c) Pentagon simple (d) Pentagon convex

(e) Hexagon simple (f) Hexagon convex

Figure 21: Practical application 1: Maximum-area simple and convex k-gon

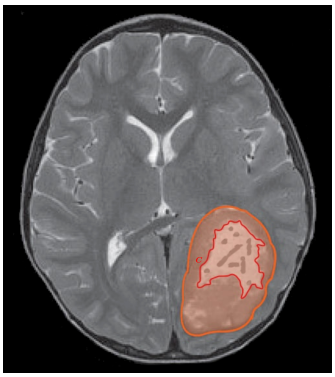
in identifying and quantifying relevant regions. The selected partition sizes are $L_1 = 1$ and $L_2 = 1/2$, resulting in 86 and 325 points, respectively. The values reported in Table 6 correspond to polygonal areas. No execution times are included in this table; for runtime measurements, see Tables 3–4.

8 Source Code

All the examples included in Section 5.3 and Section 7 were initially generated using pseudocode and later developed in Java and Python, which is available on GitHub [20]. In this way, any researcher can verify our solutions or include new examples to satisfy their needs.

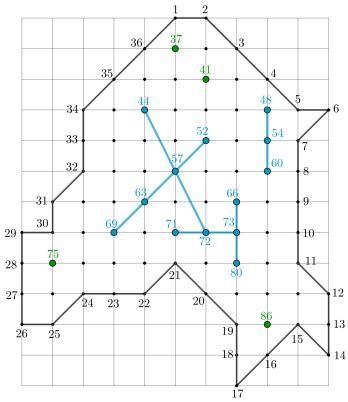
9 Conclusions

This paper presents a novel method for calculating any simple k-gon within a closed contour without restrictions, including forbidden points, segments and holes in the interior. Any



(a) Closed contour C

(b) Lattice polygon P on closed contour C



(c) Lattice polygon P

Figure 22: Practical application 2: Medical imaging

Table 6: Practical application 2: Maximum-area with partition sizes $L_1 = 1$ and $L_2 = 1/2$

$k-$**gon**	$\boldsymbol{L_1 = 1}$ $\boldsymbol{N = 86}$	$\boldsymbol{L_2 = 1/2}$ $\boldsymbol{N = 325}$
Triangle	10	11.25
Rectangle	10	10
Quadrilateral Simple	11.50	13.50
Quadrilateral Convex	11	12.50
Pentagon Simple	14.50	16.75
Pentagon Convex	11.50	12.38

researcher can compute the largest simple k-gon of maximum area or perimeter contained in a simple polygon with the proposed algorithm by first determining the type of k-gon (triangle, quadrilateral, pentagon, hexagon, etc.). Additionally, the algorithm can also be modified to compute convex k-gons of maximum area or perimeter. No other algorithm is as complete, generic, and versatile as the one proposed in this paper, since it is the user who determines which type of solution is needed. In addition to explaining all pseudocode, the source code (Java and Python) is available on GitHub for research purposes.

(a) Triangle (b) Rectangle (c) Quadrilateral simple

(d) Quadrilateral convex (e) Hexagon simple (f) Hexagon convex

Figure 23: Practical application 2: Maximum-area simple and convex k-gon

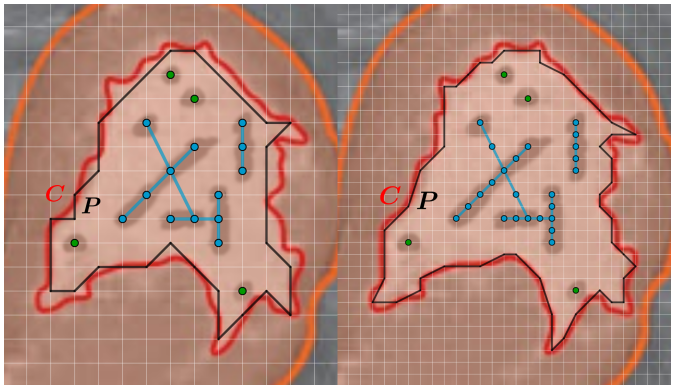


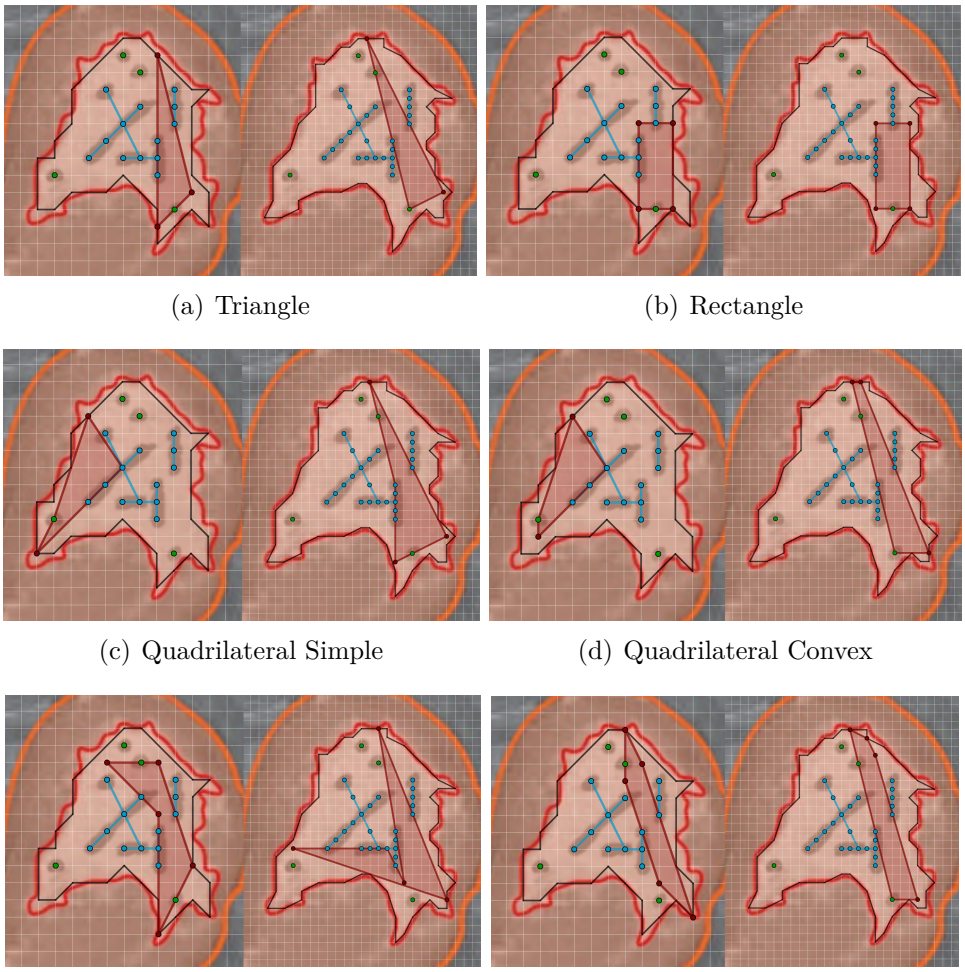
Figure 24: Lattice polygon P with different partition sizes: $L_1 = 1$ and $L_2 = 1/2$

References

[1] A. Aggarwal and S. Suri: *Fast algorithms for computing the largest empty rectangle.* In *Proceedings of the Third Annual Symposium on Computational Geometry*, 278–290. 1987. doi: 10.1145/41958.41988.

[2] H. Ali, S. Faisal, K. Chen, and L. Rada: *Image-selective segmentation model for multi-regions within the object of interest with application to medical disease.* The Visual Computer **37**(5), 939–955, 2021. doi: 10.1007/s00371-020-01845-1.

[3] S. Alqazzaz, X. Sun, X. Yang, and L. Nokes: *Automated brain tumor segmentation*



(a) Triangle (b) Rectangle

(c) Quadrilateral Simple (d) Quadrilateral Convex

(e) Pentagon Simple (f) Pentagon Convex

Figure 25: Practical application 2: Maximum-area with partition sizes $L_1 = 1$ and $L_2 = 1/2$

on multi-modal MR image using SegNet. Computational Visual Media **5**, 209–219, 2019. doi: 10.1007/s41095-019-0139-y.

[4] S. L. Anderson and S. C. Murray: *R/UAStools::plotshpcreate: Create Multi-Polygon Shapefiles for Extraction of Research Plot Scale Agriculture Remote Sensing Data*. Frontiers in Plant Science **11**, 511768, 2020. doi: 10.3389/fpls.2020.511768.

[5] M. Ausserhofer, S. Dann, Z. Lángi, and G. Tóth: *Corrigendum to "An algorithm to find maximum area polygons circumscribed about a convex polygon" [Discrete Appl. Math. 255 (2019) 98–108]*. Discrete Applied Mathematics **353**, 222–226, 2024. doi: 10.1016/j.dam.2024.04.013.

[6] E. Baco, O. Ukimura, E. Rud, L. Vlatkovic, A. Svindland, M. Aron, S. Palmer, T. Matsugasumi, A. Marien, J.-C. Bernhard, et al.: *Magnetic resonance imaging–transectal ultrasound image-fusion biopsies accurately characterize the index tumor: correlation with step-sectioned radical prostatectomy specimens in 135 patients*. European Urology **67**(4), 787–794, 2015. doi: 10.1016/j.eururo.2014.08.077.

[7] S. W. Bae and S. D. Yoon: *Empty squares in arbitrary orientation among points.* Algorithmica 1–46, 2022. doi: 10.1007/s00453-022-01002-1.

[8] E. G. Birgin and F. N. C. Sobral: *Minimizing the object dimensions in circle and sphere packing problems.* Computers & Operations Research **35**(7), 2357–2375, 2008. doi: 10.1016/j.cor.2006.11.002.

[9] J. E. Boyce, D. P. Dobkin, R. L. Drysdale III, and L. J. Guibas: *Finding extremal polygons.* In *Proceedings of the fourteenth annual ACM symposium on Theory of computing*, 282–289. 1982. doi: 10.1145/800070.802202.

[10] J. Chaudhuri, S. C. Nandy, and S. Das: *Largest empty rectangle among a point set.* Journal of algorithms **46**(1), 54–78, 2003. doi: 10.1016/S0196-6774(02)00285-7.

[11] B. Chazelle, R. Drysdale, and D. Lee: *Computing the largest empty rectangle.* SIAM Journal on Computing **15**(1), 300–315, 1986. doi: 10.1137/0215022.

[12] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein: *Introduction to Algorithms.* The MIT Press, New York, 3 ed., 2009.

[13] H. Freeman and R. Shapira: *Determining the minimum-area encasing rectangle for an arbitrary closed curve.* Communications of the ACM **18**(7), 409–413, 1975. doi: 10.1145/360881.360919.

[14] N. Friedrichs, R. Vorreuther, C. Poremba, K.-L. Schafer, A. Böcking, R. Buettner, and H. Zhou: *Primitive neuroectodermal tumor (PNET) in the differential diagnosis of malignant kidney tumors.* Pathology-Research and Practice **198**(8), 563–569, 2002. doi: 10.1078/0344-0338-00303.

[15] A. Gagliano, F. Patania, F. Nocera, A. Capizzi, and A. Galesi: *GIS-based decision support for solar photovoltaic planning in urban environment.* In *Sustainability in Energy and Buildings: Proceedings of the 4th International Conference in Sustainability in Energy and Buildings*, 865–874. 2013. doi: 10.1007/978-3-642-36645-1_77.

[16] G. Gibert, D. D'Alessandro, and F. Lance: *Face detection method based on photoplethysmography.* In *2013 10th IEEE International Conference on Advanced Video and Signal Based Surveillance*, 449–453. 2013. doi: 10.1109/AVSS.2013.6636681.

[17] Y. He and A. Kundu: *2-D shape classification using hidden Markov model.* IEEE Transactions on Pattern Analysis & Machine Intelligence **13**(11), 1172–1184, 1991. doi: 10.1109/34.103276.

[18] M. Izadi and P. Saeedi: *Three-dimensional polygonal building model estimation from single satellite images.* IEEE Transactions on Geoscience and Remote Sensing **50**(6), 2254–2272, 2011. doi: 10.1109/TGRS.2011.2172995.

[19] B. Manjunath, C. Shekhar, and R. Chellappa: *A new approach to image feature detection with applications.* Pattern Recognition **29**(4), 627–640, 1996. doi: 10.1016/0031-3203(95)00115-8.

[20] Media Engineering Group (GIM): *Source Code, Scripts, and Documentation*, 2025, May 14. https://github.com/UniversidadExtremadura/maximum-k-gon-in-a-closed-contour-with-obstacles.

[21] B. H. Menze, A. Jakab, S. Bauer, J. Kalpathy-Cramer, K. Farahani, J. Kirby, Y. Burren, N. Porz, J. Slotboom, R. Wiest, et al.: *The multimodal brain tumor image segmentation benchmark (BRATS)*. IEEE transactions on medical imaging **34**(10), 1993–2024, 2014. doi: 10.1109/TMI.2014.2377694.

[22] R. Molano, J. Sancho, M. Ávila, P. Rodríguez, and A. Caro: *Obtaining the user-defined polygons inside a closed contour with holes*. The Visual Computer 1–19, 2023. doi: 10.1007/s00371-023-03170-9.

[23] A. Mukhopadhyay and S. Rao: *Computing a largest empty arbitrary oriented rectangle: theory and implementation*. In *International Conference on Computational Science and Its Applications*, 797–806. 2003. doi: 10.1007/3-540-44842-X_81.

[24] A. Naamad, D. Lee, and W.-L. Hsu: *On the maximum empty rectangle problem*. Discrete Applied Mathematics **8**(3), 267–277, 1984. doi: 10.1016/0166-218X(84)90124-0.

[25] S. C. Nandy, A. Sinha, and B. B. Bhattacharya: *Location of the largest empty rectangle among arbitrary obstacles*. In *International Conference on Foundations of Software Technology and Theoretical Computer Science*, 159–170. 1994. doi: 10.1007/3-540-58715-2_122.

[26] T. Oksanen: *Shape-describing indices for agricultural field plots and their relationship to operational efficiency*. Computers and Electronics in Agriculture **98**(1), 252–259, 2013. ISSN 0168-1699. doi: 10.1016/j.compag.2013.08.014.

[27] M. Orlowski: *A new algorithm for the largest empty rectangle problem*. Algorithmica **5**(1), 65–73, 1990. doi: 10.1007/BF01840377.

[28] E. Persoon and K.-S. Fu: *Shape discrimination using Fourier descriptors*. IEEE Transactions on Pattern Analysis and Machine Intelligence (3), 388–397, 1986. doi: 10.1109/TPAMI.1986.4767799.

[29] I. A. Qasmieh, H. Alquran, and A. M. Alqudah: *Occluded iris classification and segmentation using self-customized artificial intelligence models and iterative randomized Hough transform*. International Journal of Electrical and Computer Engineering **11**(5), 4037, 2021. doi: 10.11591/ijece.v11i5.pp4037-4049.

[30] G. Rote: *The largest contained quadrilateral and the smallest enclosing parallelogram of a convex polygon*, 2019. arXiv: 10.48550/arXiv.1905.11203.

[31] W. Sun, H. Zhao, and Z. Jin: *A visual attention based ROI detection method for facial expression recognition*. Neurocomputing **296**, 12–22, 2018. doi: 10.1016/j.neucom.2018.03.034.

[32] D. E. Varberg: *Pick's theorem revisited*. The American Mathematical Monthly **92**(8), 584–587, 1985. doi: 10.1080/00029890.1985.11971689.

Received August 10, 2025; final form September 4, 2025.