SIAM J. Imaging Sciences
Vol. 15, No. 4, pp. 1808–1832

An Algorithm to Compute Any Simple k-gon of a Maximum Area or Perimeter Inscribed in a Region of Interest[*]

Rubén Molano[†], Mar Ávila[‡], José Carlos Sancho[‡], Pablo G. Rodríguez[‡], and Andres Caro[‡]

Abstract. Computational and mathematical models are research subjects for solving engineering, computer science, and computer vision problems. Image preprocessing usually needs to efficiently compute polygons related to some previously delimited region of interest. Most of the solved problems are limited to the search for some type of polygon with k sides (triangles, rectangles, squares, etc.) with maximum area, maximum perimeter, or similar. This paper presents a generic algorithm that computes in $O(n^5k)$ computational time the polygon of any number of sides (any simple k-gon) inscribed in a region of interest (in any closed contour without restrictions). The polygon obtained fulfills the requirements specified by the user: maximum area or perimeter or minimum area or perimeter. No previous work has been proposed to obtain any k-gon inscribed in any unconstrained contour. The algorithms and mathematical models are presented and explained, and the source code is available in a GitHub repository for research purposes.

Key words. k-gon, simple polygon, region of interest, area, perimeter

MSC codes. 68U05, 68U10, 52A38

DOI. 10.1137/22M1482676

1. Introduction.

In the last few decades, several algorithms and solutions based on geometry related to polygons have been proposed in response to problems concerned with the search for the polygon of maximum or minimum area, and maximum or minimum perimeter. This is because polygons are used as a basic figure in computer vision, remote sensing, geographic information systems, robotics, and even in biomedical applications, where polygons are essential structures for path planning.

Thus, Sun et al. [39] optimized information extraction in the building segmentation process, and Li et al. [29] described a framework to extract building footprint polygons from very high-resolution aerial images. Zhao, Persello, and Stein [44] used an improved method based on PolyMapper [28] to predict the contour of buildings in a vector format. Polygon computation has also been used in the biomedical field, in iris recognition systems [36], and to evaluate facial microexpressions [43]. Other studies have also been developed for early detection of cancer by analyzing medical images and determining the best region of interest (ROI) [4, 6]. Additionally, in robotics, many researchers have used polygons in their studies. The authors of [5, 22] solved the polygon decomposition problem or the problem of dividing a polygon into

[*]Received by the editors March 7, 2022; accepted for publication (in revised form) July 19, 2022; published electronically November 14, 2022.
https://doi.org/10.1137/22M1482676
Funding: This work was funded by the Junta de Extremadura (European Regional Development Fund), Consejería de Economía, Ciencia y Agenda Digital, under project GR21099.

[†]Department of Mathematics, University of Extremadura, 10003 Cáceres, Spain (rmolano@unex.es).
[‡]Department of Computer and Telematics Systems Engineering, University of Extremadura, 10003 Cáceres, Spain (mmavila@unex.es, jcsancho@unex.es, pablogr@unex.es, acaro@unex.es).

a set of smaller polygons of a given area. The authors of [42] presented a strategy for a mobile robot to explore an unknown simple polygon. The robot's task was to explore each cell and return to the start, where the number of cell visits was as small as possible.

One of the most important problems is the search for the largest polygon that can be built within an ROI (a closed contour) or that can be contained within another polygon. As presented in section 2, most of the solutions proposed to date are based on a unique type of polygon (triangle, rectangle, square, parallelogram). In other words, the problem is solved for a unique k-gon, where k is the number of sides of the polygon (3-gon for triangles, 4-gon for quadrilaterals, 5-gon for pentagons, etc.). Moreover, the solutions presented are limited only to convex polygons and, less frequently, to simple polygons. In the majority of cases, the solutions do not provide algorithms or pseudocode that could facilitate the reproducibility and user-friendliness of the solutions obtained.

In this paper, a generic solution is presented that is capable of computing the simple k-gon of the maximum area or maximum perimeter inscribed in any closed contour (ROI) which is not necessarily convex; it is also possible to obtain the minimum area or minimum perimeter with a small modification to the algorithm. An algorithmic solution is shown with the aim of finding the simple (nonconvex) polygon of any number of k sides chosen by the user (k is a value that is specified by the parameter in the proposed algorithmic solution). No other paper has proposed a generic solution for any type of polygon. All the proposed approaches are focused on very specific, partial, and nonreconfigurable solutions. In our algorithm, the user can indicate the number of sides desired for the polygon and the solution to be found: the maximum area, the maximum perimeter, or the minimum area and the minimum perimeter. The polygon obtained, in relation to the functionality specified, determines the region of interest (ROI) where the users are interested for their specific purposes. Using this method, any researcher can obtain the most appropriate k-sided ROI for their research interests. There is no need to find one method for finding the largest triangle, another for finding the largest quadrilateral, square, or rectangle, another for finding the largest pentagon, etc. A single method offers all the options. Furthermore, this paper presents the pseudocode of the algorithm, modularized in the 8 subprograms that form it so that any researcher can easily understand and adapt it to their favorite programming language (C/C++, Java, Python, R, etc.). Additionally, a link to a GitHub repository with all the source code developed in C/C++, Java, and Python is included [30].

In particular, this 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

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

Our algorithm solves the inclusion problem $Inc(\mathcal{P}_{all}, \mathcal{P}_{k\,sim}, \mu)$, where $\mathcal{P}_{all}$ is the family of all simple polygons, $\mathcal{P}_{k\,sim}$ denotes the class of all simple k-gons, and μ is the real function with respect to area or perimeter.

The aim of this paper is to develop a generic algorithm that computes any simple k-gon inscribed in an ROI (a closed contour without restrictions). Thus, the polygon obtained fulfills the requirements specified by the user, such as the maximum area or perimeter, as well as the minimum area or perimeter.

The main contributions of this paper are as follows: (i) an efficient algorithm is presented to compute any simple k-gon inscribed in a closed contour; (ii) the obtained simple k-gon can be one of the maximum areas or perimeters and minimum areas or perimeters; (iii) all the pseudocode is presented and explained to facilitate its reproducibility and extension; (iv) all source code, scripts, and documents are available for the scientific community in a GitHub repository.

The algorithm described in this paper is detailed in the following sections. Section 2 presents a review of related works. Section 3 introduces the concept of lattice polygon, computes the maximum-area or perimeter simple k-gon in a lattice polygon, and shows the results with two examples. Then, section 4 expands on the previous sections and describes how to compute the maximum-area or perimeter simple k-gon inscribed in a region of interest. The feasibility of the algorithm is also shown in a real practical application to calculate the maximum-area k-gon in agricultural plots. Section 5 shows where the source code can be downloaded. Finally, section 6 demonstrates how the ideas presented in this paper could be useful for future work and presents the conclusions of our research.

2. Related works.

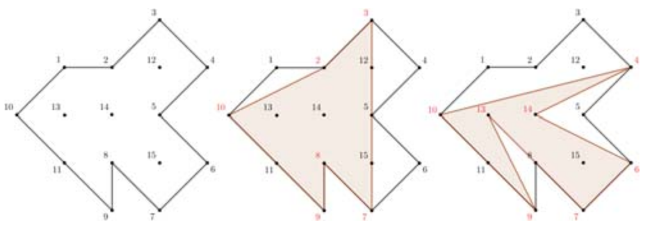
Boyce et al. [10] showed that given a convex n-gon, the maximum-area or perimeter inscribed k-gon can be found in $O(kn \log n + n \log^2 n)$ time. Later, Aggarwal et al. [2] reduced the complexity of the algorithm to $O(kn + n \log n)$. They considered the inclusion problem $Inc(\mathcal{P}_{con}, \mathcal{P}_k, \mu)$, where $\mathcal{P}_{con}$ is the family of all convex polygons, $\mathcal{P}_k$ is the family of all convex k-gons, and μ is the function with respect to the area or perimeter. The difference between this paper and the previous authors' achievements is substantial, since the initial polygon and the k-gon are simple and have greater difficulty than a convex polygon. These restrictions cause the complexity order increase to $O(n^5 k)$. Table 1 shows the computational costs of the previous papers, and Figure 1 shows some results of our algorithm.

Table 1
Computational cost.

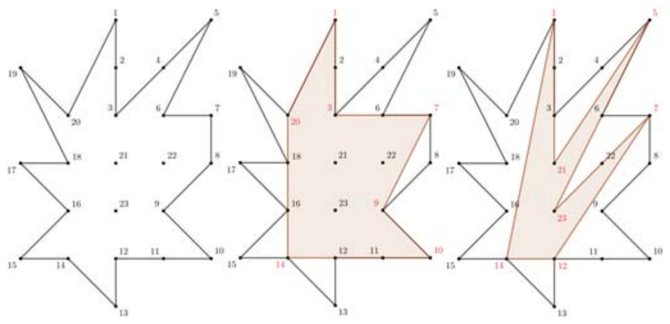
Author	Polygon	k-gon	Computational cost
Boyce et al. [10]	convex	convex	$O(kn \log n + n \log^2 n)$
Aggarwal et al. [2]	convex	convex	$O(kn + n \log n)$
this paper	simple	simple	$O(n^5 k)$

Inclusion problems have been studied by researchers for several years. For triangles, Van Der Hoog et al. [41] showed how to compute the maximum-area triangle in a convex polygon in $O(n \log n)$ time. Later, Kallus [24] solved this problem in $O(n)$ time. For simple polygons the solution became more complex, and thus, Melissaratos and Souvaine [31] presented an algorithm for finding the maximum-area triangle inscribed in a simple n-gon in $O(n^4)$ time.

For rectangles, Alt, Hsu, and Snoeyink [3] computed the largest area axis-parallel rectangle in a convex polygon in $O(\log n)$ time, and when the constraint for convex polygons was removed, Daniels, Milenkovic, and Roth [16] solved the problem in $O(n \log^2 n)$ time first, and then Boland and Urrutia [9] solved it in $O(n \log n)$ time. In addition, Sarkar et al. [38] presented a combinatorial algorithm to find the largest area rectangle inside a digital object in $O(k.n/g + (n/g) \log(n/g))$ time, building the inner isothetic cover first [8]. If it was not



(a) Maximum-area and perimeter simple 6-gon contained in a simple 10-gon.



(b) Maximum-area and perimeter simple 7-gon contained in a simple 17-gon.

Figure 1. *Maximum-area and perimeter simple k-gon contained in a simple n-gon.*

an axis-aligned rectangle, Knauer et al. [27] considered approximation algorithms and proved that the rectangle with the largest area of arbitrary orientation in a convex polygon could be computed in $O(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon}\log n)$. Molano et al. [34] extended the problem, and the authors showed how to compute the largest area rectangle of arbitrary orientation in a closed contour in $O(n^3)$ time working with ROIs. Again the extension from convex polygons to simple polygons was important for obtaining a higher computational cost. Jin [23] considered the problem for parallelograms and showed how to compute in $O(n\log^2 n)$ time the maximum area parallelogram in a convex polygon. In the same way, Molano et al. [33] obtained the largest parallelogram in $O(n^3)$ time for simple polygons. Finally, Keikha et al. [25] and Rote [37] computed a linear-time algorithm for finding the quadrilateral of the largest area contained in a convex polygon.

The "potato peeling problem," or the problem of finding the largest convex polygon contained in a simple polygon, is the inclusion problem $Inc(\mathcal{P}_{all}, \mathcal{P}_{con}, \mu)$. The problem was introduced by Goodman [20] and solved by Chang and Yap [14] in $O(n^7)$ time under the area measure and in $O(n^6)$ time for the perimeter. Later, Hall-Holt et al. [21] gave an approximation algorithm in $O(n\log n)$ time.

This problem is closely related to the problem of finding the convex hull [15]: Given a set P of n points in the plane, compute the smallest convex polygon Q such that each point is on the boundary Q or in its interior. Both problems are complementary to each other if the first one belongs to the class of *inclusion problems*, and the second one is defined by *enclosure problems*, as follows: $Enc(\mathcal{P}, \mathcal{Q}, \mu)$: Given $P \in \mathcal{P}$, find the μ-smallest $Q \in \mathcal{Q}$ that encloses P.

In this sense, enclosure problems have also been studied by researchers in many cases. Klee and Laskowski [26] considered the enclosure problem $Enc(\mathcal{P}_{con}, \mathcal{P}_3, area)$, where $\mathcal{P}_3$ denotes the class of all triangles, which was solved in $O(n \log^2 n)$ time. Later, O'Rourke et al. [35] improved it to linear time, which was optimal. DePano [17] extended the method described in [26] to solve $Enc(\mathcal{P}_{con}, \mathcal{P}_k, area)$ for all k in $O(n^{k-2} \log^2 n)$ time. Chang and Yap [13] improved DePano's result to $O(n^3 \log k)$ time, and finally, Aggarwal, Chang, and Yap [1] refined the latter solution to $O(n^2 \log n \log k)$. For rectangles, the problem of finding the smallest rectangle containing a convex polygon was solved by Toussaint [40] in a linear time solution. Previously, Freeman and Shapira [19] showed that the minimum area enclosing a rectangle for an arbitrary closed curve can be computed in $O(n^2)$ time.

The first solution to the enclosure problem $Enc(\mathcal{P}_{con}, \mathcal{P}_3, perimeter)$ was proposed by DePano [18] and calculated in $O(n^3)$ time. Later, Bhattacharya and Mukhopadhyay [7] obtained a linear-time algorithm. In addition, the enclosure problem $Enc(\mathcal{P}_{con}, \mathcal{P}_k, perimeter)$, or the problem of finding a minimum-perimeter k-gon that enclosed a given n-gon, was solved by Mitchell and Polishchuk [32] in $O(nk \log k)$ time.

As seen in all the articles reviewed, there was no algorithm that could be generalized for the whole class of polygons. In most of the articles, the solution was given for one type of polygon (triangle, rectangle, parallelogram, quadrilateral, etc.) and sometimes with restrictions (axis-parallel rectangle), with the main difference being when the problems are trying to be solved from a convex polygon to a simple polygon, since the computational cost increases considerably. Table 2 shows the computational cost for the most important inclusion problems.

Table 2
Computational cost (inclusion problems).

Reference	Initial polygon	Final polygon	Comput. cost
[24]	convex	triangle	$O(n)$
[31]	simple	triangle	$O(n^4)$
[3]	convex	axis-parallel rectangle	$O(\log n)$
[9]	simple	axis-parallel rectangle	$O(n \log n)$
[27]	convex	rectangle	$O(\frac{1}{\epsilon} \log \frac{1}{\epsilon} \log n)$
[34]	simple	rectangle	$O(n^3)$
[23]	convex	parallelogram	$O(n \log^2 n)$
[33]	simple	parallelogram	$O(n^3)$
[25, 37]	convex	quadrilateral	$O(n)$
[21]	simple	convex k-gon	$O(n \log n)$

3. Maximum-area or perimeter simple k-gon in a lattice polygon.

Given the rectangle $[a, b] \times [c, d]$, $a, b, c, d \in \mathbb{Z}$, a *regular partition* $\Pi = \Pi_x \times \Pi_y$ of order $r \times s$ is two ordered collections of $r + 1$, $s + 1$ equally spaced points that satisfy

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

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, where $L = |x_{i+1} - x_i| = |y_{j+1} - y_j|$ is the length of the side of each square formed by the square grid (also called *partition size*). 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 $\Pi \preceq \dot{\Pi}$.

Let P be a polygon whose vertices belong to the square grid G_L for a regular partition Π on the rectangle circumscribed to P. A polygon P defined in this way is said to be a *lattice polygon* (Figure 2). As can be seen in Figure 1(b), connections between consecutive vertices are not necessarily established in the eight directions, $0°, 45°, 90°, \dots, 315°$, as are, for example, the connections between vertices 5-6, 18-19, and 20-1.

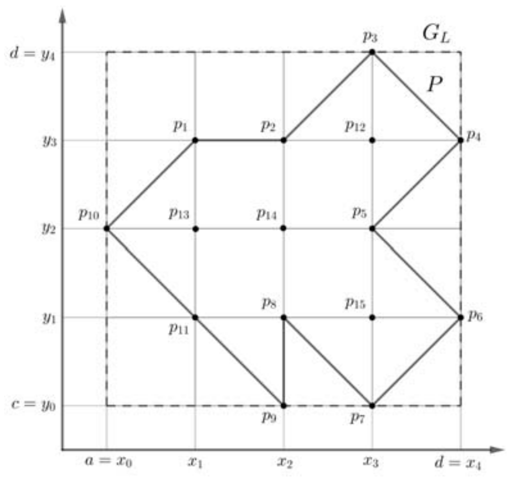


Figure 2. *Lattice polygon P on a regular partition of order 4×4 with partition size L.*

We denote ∂P as the family consisting of boundary nodes of P, and its complementary in P, $\imath P$, the interior points, i.e., $P = \partial P \cup \imath P$. By Pick's theorem [12],

$$A(P) = \left(\#(\imath P) + \frac{\#(\partial P)}{2} - 1 \right) \cdot L^2,$$

where $A(P)$ denotes the area of lattice polygon P and $\#$ represents the cardinality of the set. Similarly, we denote V as the family consisting of vertices of ∂P, and its complementary in ∂P we denote $\imath\partial P$, i.e., $\partial P = V \cup \imath\partial P$. Then we decompose the lattice polygon P as follows:

$$P = V \cup \imath\partial P \cup \imath P = \{p_1, p_2, \dots, p_{n+m+o}\}$$

with $\#(V) = n$, $\#(\imath\partial P) = m$, $\#(\imath P) = o$, and $\#(P) = N = n + m + o \simeq kn, k \in \mathbb{N}$.

Thus, for Figure 2,

$$\left\{ \begin{array}{rcl} V & = & \{p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8, p_9, p_{10}\}, \\ \imath\partial P & = & \{p_{11}\}, \\ \imath P & = & \{p_{12}, p_{13}, p_{14}, p_{15}\}. \end{array} \right.$$

3.1. Adjacency matrix. Given N points of the polygon $P = V \cup \imath\partial P \cup \imath P = \{p_1, p_2, \dots, p_{n+m+o}\}$, the adjacency matrix $A = (a_{ij})$ is a square matrix of order N such that

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

For Figure 2, the adjacency matrix is represented as follows:

$$A=\begin{pmatrix}
0&1&0&1&1&1&1&1&1&1&1&1&1&1&1\\
1&0&1&1&1&1&1&1&1&1&1&1&1&1&1\\
0&1&0&1&1&0&1&1&0&0&1&1&1&1&1\\
1&1&1&0&1&0&0&1&0&1&1&1&1&1&0\\
1&1&1&1&0&1&1&1&0&1&1&1&1&1&1\\
1&1&0&0&1&0&1&1&0&1&1&0&1&1&1\\
1&1&1&0&1&1&0&1&0&0&0&1&1&1&1\\
1&1&1&1&1&1&1&0&1&1&1&1&1&1&1\\
1&1&0&0&0&0&0&1&0&1&1&0&1&1&0\\
1&1&0&1&1&1&0&1&1&0&1&1&1&1&1\\
1&1&1&1&1&1&0&1&1&1&0&1&1&1&1\\
1&1&1&1&1&0&1&1&0&1&1&0&1&1&1\\
1&1&1&1&1&1&1&1&1&1&1&1&0&1&1\\
1&1&1&1&1&1&1&1&1&1&1&1&1&0&1\\
1&1&1&0&1&1&1&1&0&1&1&1&1&1&0
\end{pmatrix}.$$

3.2. Main algorithm. We compute the maximum-area o perimeter simple k-gon in a lattice polygon $P = V\cup\imath\partial P\cup\imath P = \{p_1, p_2, \dots, p_{n+m+o}\}$, $\#(P) = n+m+o = N \simeq \lambda n, \lambda \in \mathbb{N}$, with coordinates belonging to the square grid G_L; we follow the next process defined in three steps.

STEP 1: We compute the possible sides of the simple k-gon by Algorithm 3.1 in $O(n^3k)$ time and Algorithm 3.2 in $O(n^5k)$ time.

Algorithm 3.1 computes all edges that are a certain distance from point 1 to point 2. The algorithm starts by assuming that the two nodes are connected (Line 2). If so, it first computes all the edges contained in polygon P at distance 1 from point 1. The algorithm successively stores all the edges until it reaches the chosen distance where the initial solution is at "temp" (Line 15). Of all the solutions that appear in "temp," we keep those where the final node coincides with point 2 (Line 19). The computational cost is determined by the loops:

- Line 3: $O(k)$, because k =distance +1.
- Line 5: $O(n^2)$, since in the worst case the maximum number of edges coincides with the combinatorial number $\binom{N}{2} = \frac{N\cdot(N-1)}{2} \simeq N^2 \simeq n^2$.
- Line 8: $O(n)$, since the order of the matrix A is $N \simeq n$.

Therefore, the computational cost of Algorithm 3.1 is $O(n^3k)$.

Algorithm 3.2 computes all sides that are within a certain distance. Since $\#(P) = N \simeq n$, the computational cost is $O(n^5k)$.

For Figure 2, the solutions (sides) calculated by Algorithm 3.2 with points = 15, distance = 3, and matrix = A are as follows:

$$\text{SIDES}(15, 3, A) = \{(1, 2, 3, 4), (1, 2, 3, 5), \dots, (15, 14, 13, 11), (15, 14, 13, 12)\}$$

STEP 2: We eliminate all those solutions of Algorithm 3.2 that cannot form a polygon. To do this, we include two conditions, one shown in Algorithm 3.3 and the other in Algorithm 3.4.

Algorithm 3.1 EDGES (point1, point2, distance, matrix).

```
1:  temp ← {point1}
2:  if matrix(point1, point2) = 1 then
3:     for $i$ ← 1 to distance do
4:        edges ← ∅
5:        for $k$ ← 1 to Length(temp) do
6:           last ← temp[$k$][$i$]
7:           if last ≠ point2 then
8:              for $j$ ← 1 to $N$ do
9:                 if matrix(last,$j$) = 1 and Length(Union(temp[$k$],$j$)) ≠ Length(temp[$k$]) then
10:                   Insert(edges, Insert(temp[$k$],$j$))
11:                end if
12:             end for
13:          end if
14:       end for
15:       temp ← edges
16:    end for
17:    edges ← ∅
18:    for $k$ ← 1 to Length(temp) do
19:       if temp[$k$][distance + 1] = point2 then
20:          Insert(edges, temp[$k$])
21:       end if
22:    end for
23: end if
24: return edges
```

Algorithm 3.2 SIDES (points, distance, matrix).

```
1: sides ← ∅
2: for $i$ ← 1 to points-1 do
3:    for $j$ ← $i + 1$ to points do
4:       if EDGES ($i$, $j$, distance, matrix) ≠ ∅ then
5:          Insert(sides, EDGES ($i$, $j$, distance, matrix))
6:       end if
7:    end for
8: end for
9: return sides
```

Algorithm 3.5 finally computes all simple k-gons contained in a lattice polygon.

Algorithm 3.3 allows us to determine when three consecutive points are aligned and returns 0 if they are aligned and other than 0 otherwise. This can be solved in $O(k)$ time.

Definition 3.1. *Let $A = (x_1, y_1)$, $B = (x_2, y_2)$, and $C = (x_3, y_3) \in \mathbb{R}^2$. Then, A, B, and C*

are aligned if

$$\frac{x_2 - x_1}{x_3 - x_2} = \frac{y_2 - y_1}{y_3 - y_2}.$$

In other words, $(x_2 - x_1) \cdot (y_3 - y_2) - (x_3 - x_2) \cdot (y_2 - y_1) = 0$.

Figure 3 shows two polygons: $A = (1, 9, 11, 10)$ and $B = (2, 3, 12, 14)$, solutions of SIDES$(15, 3, A)$. While B is a quadrilateral, A is actually a triangle, because points 9, 11, and 10 are aligned.

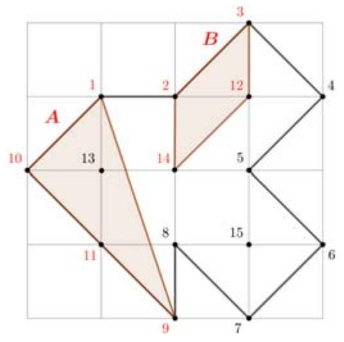


Figure 3. *A nonquadrilateral; B quadrilateral.*

Algorithm 3.3 ALIGNED (points, path).

```
1: temp ← path
2: Insert(temp, path[1])
3: Insert(temp, path[2])
4: for $i$ ← 1 to Length(temp)-2 do
5:     {// Let $P[j] = (x_j, y_j)$, we define $P[j]_x = x_j$ and $P[j]_y = y_j$}
6:     $alig \leftarrow (P[temp[i+1]]_x - P[temp[i]]_x) \cdot (P[temp[i+2]]_y - P[temp[i+1]]_y) - (P[temp[i+2]]_x - P[temp[i+1]]_x) \cdot (P[temp[i+1]]_y - P[temp[i]]_y) = 0$
7: end for
8: if alig = 0 then
9:     Break
10: end if
11: return alig
```

Algorithm 3.4 calculates when two segments (sides) intersect or not and returns `true` if they intersect and `false` otherwise. Figure 4 contains two solutions of SIDES$(15, 3, A)$: $A = (1, 8, 11, 2)$ and $B = (3, 5, 4, 12)$. We observe that none of them can form a quadrilateral.

The computational cost is determined by Lines 7 and 11 in $O(k)$ time, and the function *INTERSECTION (seg1, seg2: segments)* is computed in $O(1)$ time. Except for minor modifications, this function can be found in Cormen et al. [15]. The computational cost of Algorithm 3.4 is $O(k^2)$.

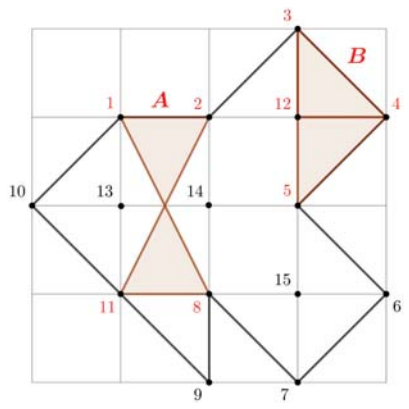


Figure 4. *Intersection of segments.*

Algorithm 3.4 SEGMENTS (points, path).

```
1:  temp $\leftarrow \emptyset$
2:  **for** $i \leftarrow 1$ **to** Length(path)-1 **do**
3:      Insert(temp, {P[path[i]], P[path[i+1]]})
4:  **end for**
5:  Insert(temp, {P[path[Length(path)]], P[path[1]]})
6:  seg $\leftarrow$ False
7:  **for** $i \leftarrow 1$ **to** Length(temp)-1 **do**
8:      **if** seg = True **then**
9:          Break
10:     **end if**
11:     **for** $j \leftarrow i+1$ **to** Length(temp) **do**
12:         seg $\leftarrow$ INTERSECTION (temp[i], temp[j])
13:         **if** seg = True **then**
14:             Break
15:         **end if**
16:     **end for**
17: **end for**
18: **return** seg
```

Algorithm 3.5 computes the simple k-gons contained in the lattice polygon P at a given distance and considers the adjacency matrix. The algorithm starts by storing all the sides (Line 1) described in Algorithm 3.2, and then uses a loop (Line 3) in $O(n^2)$ time to include only those solutions that do not satisfy the two previous conditions described in Algorithm 3.3 ($O(k)$ time) and Algorithm 3.4 ($O(k^2)$ time). The computational cost is $O(n^2k^3)$ time.

STEP 3: We compute the maximum-area or perimeter k-gon contained in a lattice polygon P with coordinates belonging to the square grid G_L. Algorithm 3.5 calculates all simple k-gons; however, to obtain the exact solution we must eliminate those repeated solutions by Algorithm 3.6 and obtain the largest area or perimeter by Algorithm 3.7. Finally, Algo-

Algorithm 3.5 POLYGONS (points, distance, matrix).

```
 1: sides ← SIDES (points, distance, matrix)
 2: polygons ← ∅
 3: for $i$ ← 1 to Length(sides) do
 4:     alig ← ALIGNED (points, sides[i])                // Algorithm 3.3
 5:     seg ← SEGMENTS (points, sides[i])                // Algorithm 3.4
 6:     if alig $\neq$ 0 and seg = False then
 7:         Insert(polygons, sides[i])
 8:     end if
 9: end for
10: return polygons
```

rithm 3.8 gives us the solution of the proposed problem by Boyce et al. [10], which is also for simple polygons to solve the inclusion problem $Inc(\mathcal{P}_{all}, \mathcal{P}_{k\,sim}, \mu)$.

Algorithm 3.6. Figure 5 shows one of the solutions for the simple 4-gon, Q = (1,4,8,10). Other solutions could have been (1,10,8,4), (4,1,10,8), (4,8,10,1), (8,4,1,10), (8,10,1,4), (10,1,4,8), and (10,8,4,1). Algorithm 3.6 chooses a representative of the above solutions and then applies Algorithm 3.7. The computational cost is determined by two loops, Line 2 and Line 4, in $O(n^2)$ time for each of them and performs a sorting algorithm (Line 5), which can be computed in $O(k \log k)$ time [15]. Thus, Algorithm 3.6 can be solved in $O(n^4 k \log k)$ time.

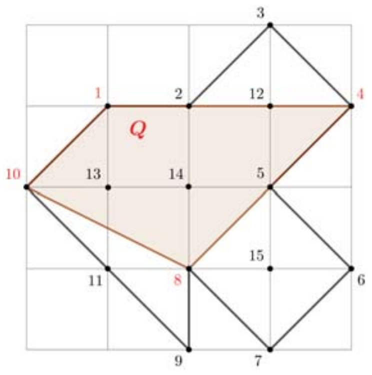


Figure 5. *Maximum-area* 4*-gon;* $Q = (1, 4, 8, 10)$.

Algorithm 3.7 computes the largest area or perimeter polygon depending on the value of the "function" parameter, 0 for the largest area and 1 for the perimeter, in $O(n^2 k)$ time. The algorithm uses a loop (Line 3) in $O(n^2)$ time and the function *AREA-PERIMETER (polygon, function)* in $O(k)$ time that calculates the area [11] or perimeter of a polygon.

Algorithm 3.8 computes the maximum-area or perimeter simple k-gon inscribed in a lattice polygon P with coordinates belonging to the square grid G_L. The computational cost is given by Algorithm 3.5, Algorithm 3.6, and Algorithm 3.7, i.e., $Max(n^2 k^3, n^4 k \log k, n^2 k) = n^4 k \log k$.

Table 3 shows a summary of all the algorithms we used. We can see that the highest

Algorithm 3.6 DUPLICATES (polygons).

```
aux1, aux2 ← ∅
for $i$ ← 1 to Length(polygons) do
    Insert(aux2, i)
    for $j$ ← $i + 1$ to Length(polygons) do
        if Sort(polygons[i]) = Sort(polygons[j]) then
            Insert(aux1, j)
        end if
    end for
end for
Sort(DeleteDuplicates(aux1))
aux3 ← Complement(aux2, aux1)                // We define $Complement(A, B) = A - B$
duplicates ← ∅
for $i$ ← 1 to Length(aux3) do
    Insert(duplicates, polygons[aux3[i]])
end for
return duplicates
```

Algorithm 3.7 UPDATE (polygons, function).

```
update ← ∅
max ← 0
for $i$ ← 1 to Length(polygons) do
    $\mu$ ← AREA-PERIMETER (polygons[i], function)
    if $\mu$ > max then
        max ← $\mu$
        update.clear()
        insert(update, polygons[i])
    else if $\mu$ = max then
        insert(update, polygons[i])
    end if
end for
return update
```

computational cost is obtained by Algorithm 3.2. in $O(n^5 k)$ time. This is the final time of our algorithm.

The Algorithm 3.8 SOLUTION is the main program of the proposal. As can be seen, it has four parameters: points, which indicates the number of points in the lattice polygon; distance, which is used to compute all sides that are within this value (distance + 1 indicates the number k of sides of the polygon to be inscribed); matrix, which is the adjacency matrix; and function, a boolean value indicating whether the maximum area or perimeter should be calculated (function = 0 or function = 1).

Algorithm 3.8 SOLUTION (points, distance, matrix, function).

```
1: solution ← $\emptyset$
2: polygons ← POLYGONS (points, distance, matrix)        // Algorithm 3.5
3: duplicates ← DUPLICATES (polygons)                    // Algorithm 3.6
4: solution ← UPDATE (duplicates, function)              // Algorithm 3.7
5: return solution
```

Table 3
Computational cost.

Algorithm 3.x	Name	Computational cost
1	EDGES (point1, point2, distance, matrix)	$O(n^3k)$
2	SIDES (points, distance, matrix)	$O(n^5k)$
5	POLYGONS (points, distance, matrix)	$O(n^2k^3)$
3	ALIGNED (points, path)	$O(k)$
4	SEGMENTS (points, path)	$O(k^2)$
8	SOLUTION (points, distance, matrix, function)	$O(n^4k\log k)$
6	DUPLICATES (polygons)	$O(n^4k\log k)$
7	UPDATE (polygons, function)	$O(n^2k)$

3.3. Experimental results. Algorithm 3.8 always computes the simple k-gon. To clarify exactly what our algorithm does, we present the two solutions of Figure 1 (Figure 6 and Figure 7) and a new figure, Figure 12, that show more results on random polygons to illustrate the possibility of finding k-gons of maximum area on the same contour.

- **Figure 6**. points = 15, k = distance +1, A = adjacency matrix, function = {0,1}.

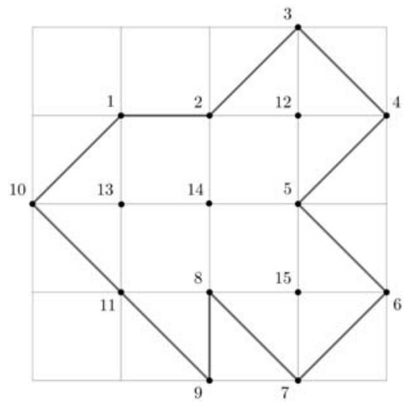


Figure 6. *Example* 1*: Maximum-area and perimeter simple k-gon.*

Area, function = {0}.

$k = 3$	Triangle	$\Rightarrow$	SOLUTION (15,2,A,0) = {(3,7,13)},
$k = 4$	Quadrilateral	$\Rightarrow$	SOLUTION (15,3,A,0) = {(1,4,8,10), (2,6,11,10)},
$k = 5$	Pentagon	$\Rightarrow$	SOLUTION (15,4,A,0) = {(1,4,8,9,10)},
$k = 6$	Hexagon	$\Rightarrow$	SOLUTION (15,5,A,0) = {(1,10,9,8,7,12), (1,4,5,6,11,10), (2,3,4,8,9,10), (2,3,7,8,9,10), (2,6,7,8,9,10), (3,4,5,6,7,13)},
$k = 7$	Heptagon	$\Rightarrow$	SOLUTION (15,6,A,0) = {(1,2,3,7,8,9,10),(1,4,5,6,8,9,10), (1,4,5,7,8,9,10),(1,10,9,8,4,3,2),(1,10,9,8,7,6,2), (2,10,11,6,5,4,3),(7,6,5,4,1,10,8)},
$k = 8$	Octagon	$\Rightarrow$	SOLUTION (15,7,A,0) = {(1,4,5,6,7,8,9,10)}.

Perimeter, function = {1}.

$k = 3$	Triangle	$\Rightarrow$	SOLUTION (15,2,A,1) = {(3,7,13)},
$k = 4$	Quadrilateral	$\Rightarrow$	SOLUTION (15,3,A,1) = {(4,10,6,13)},
$k = 5$	Pentagon	$\Rightarrow$	SOLUTION (15,4,A,1) = {(1,4,13,6,10),(4,10,11,6,13)},
$k = 6$	Hexagon	$\Rightarrow$	SOLUTION (15,5,A,1) = {(4,10,15,13,6,14)},
$k = 7$	Heptagon	$\Rightarrow$	SOLUTION (15,6,A,1) = {(4,10,9,13,7,6,14)},
$k = 8$	Octagon	$\Rightarrow$	SOLUTION (15,7,A,1) = {(4,10,9,13,7,14,6,5)}.

- **Figure 7**. points = 23, k = distance +1, A = adjacency matrix, function = {0,1}.

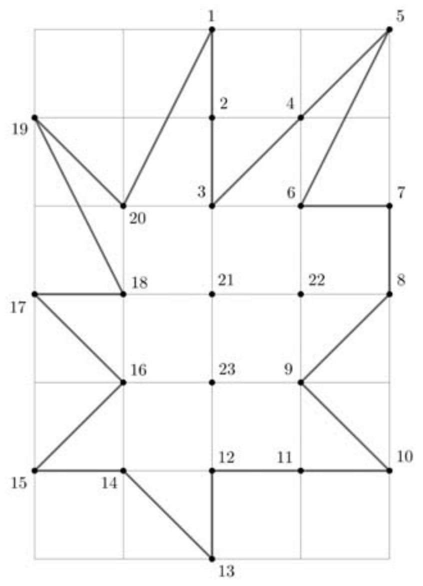


Figure 7. *Example 2: Maximum-area and perimeter simple k-gon.*

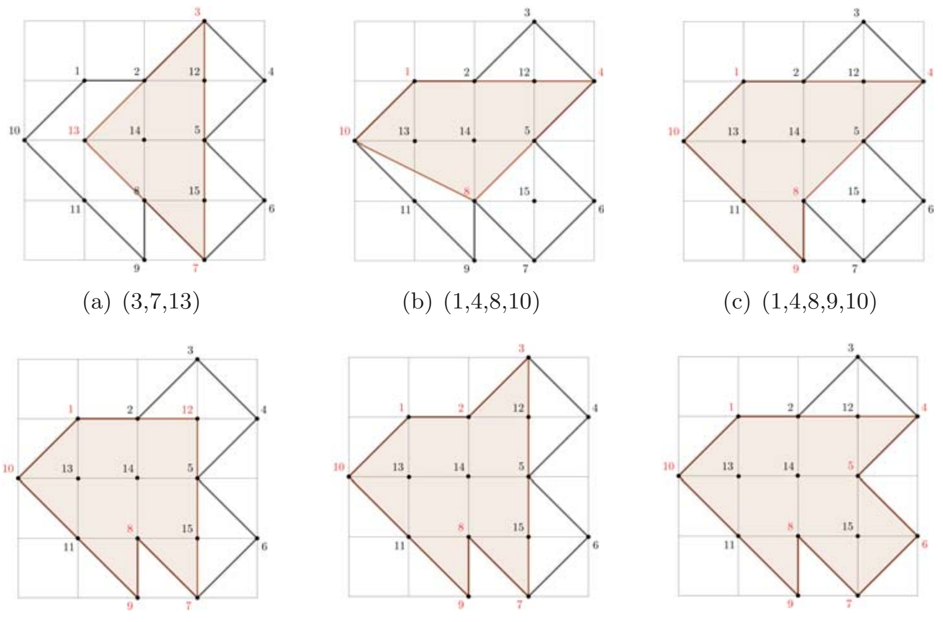
Area, function = {0}.

$k=3$	Triangle	$\Rightarrow$	SOLUTION (23,2,A,0) = {(7,12,20), (7,14,20), (8,14,20), (14,10,20)},
$k=4$	Quadrilateral	$\Rightarrow$	SOLUTION (23,3,A,0) = {(4,11,14,18), (6,20,14,11), (7,20,14,8),(7,20,14,12),(9,7,20,14)},
$k=5$	Pentagon	$\Rightarrow$	SOLUTION (23,4,A,0) = {(9,7,20,14,10)},
$k=6$	Hexagon	$\Rightarrow$	SOLUTION (23,5,A,0) = {(9,8,7,20,14,10), (15,10,9,7,20,16)},
$k=7$	Heptagon	$\Rightarrow$	SOLUTION (23,6,A,0) = {(3,1,20,14,10,9,7), (15,10,9,8,7,20,16)},
$k=8$	Octagon	$\Rightarrow$	SOLUTION (23,7,A,0) = {(9,7,3,1,20,16,15,10),(14,10,9,8,7,3,1,20)}.

Perimeter, function = {1}.

$k=3$	Triangle	$\Rightarrow$	SOLUTION (23,2,A,1) = {(13,1,14)},
$k=4$	Quadrilateral	$\Rightarrow$	SOLUTION (23,3,A,1) = {(1,14,5,21)},
$k=5$	Pentagon	$\Rightarrow$	SOLUTION (23,4,A,1) = {(11,4,23,1,14)},
$k=6$	Hexagon	$\Rightarrow$	SOLUTION (23,5,A,1) = {(5,23,8,14,1,21)},
$k=7$	Heptagon	$\Rightarrow$	SOLUTION (23,6,A,1) = {(5,23,7,12,14,1,21),(7,8,14,1,21,5,23)},
$k=8$	Octagon	$\Rightarrow$	SOLUTION (23,7,A,1) = {(10,9,7,23,5,21,1,14)}.

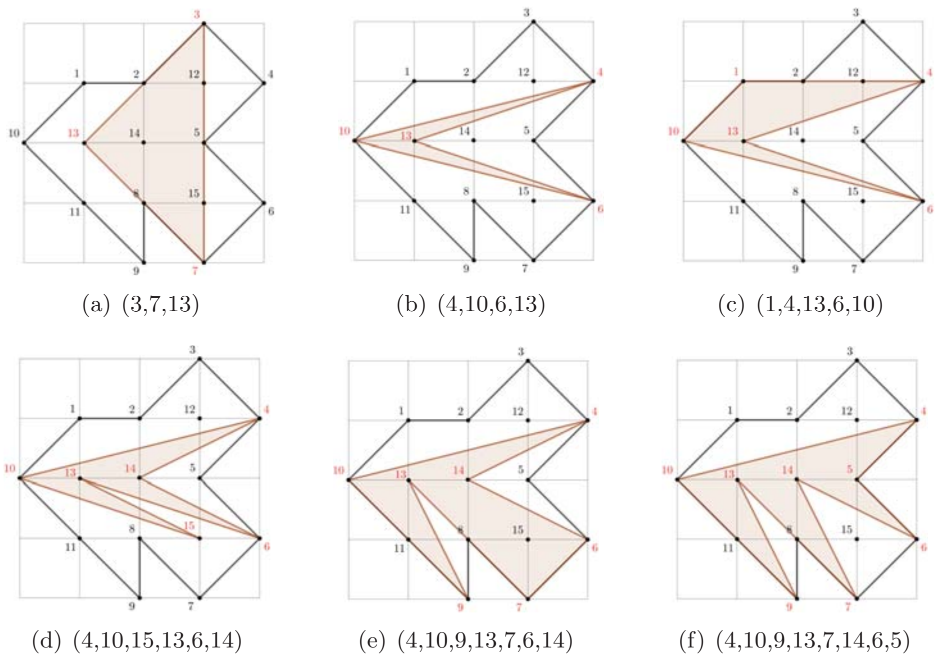
Figure 8, Figure 9 and Figure 10, Figure 11 show some of the solutions of Example 1 and Example 2, respectively:



(a) (3,7,13) (b) (1,4,8,10) (c) (1,4,8,9,10)

(d) (1,10,9,8,7,12) (e) (1,2,3,7,8,9,10) (f) (1,4,5,6,7,8,9,10)

Figure 8. *Example* 1*: Maximum-area simple* k*-gon.*



(a) (3,7,13) (b) (4,10,6,13) (c) (1,4,13,6,10)

(d) (4,10,15,13,6,14) (e) (4,10,9,13,7,6,14) (f) (4,10,9,13,7,14,6,5)

Figure 9. *Example* 1*: Maximum-perimeter simple k-gon.*

- **Figure 12**. points = 35 , k = distance +1, A = adjacency matrix, function = {0,1}.

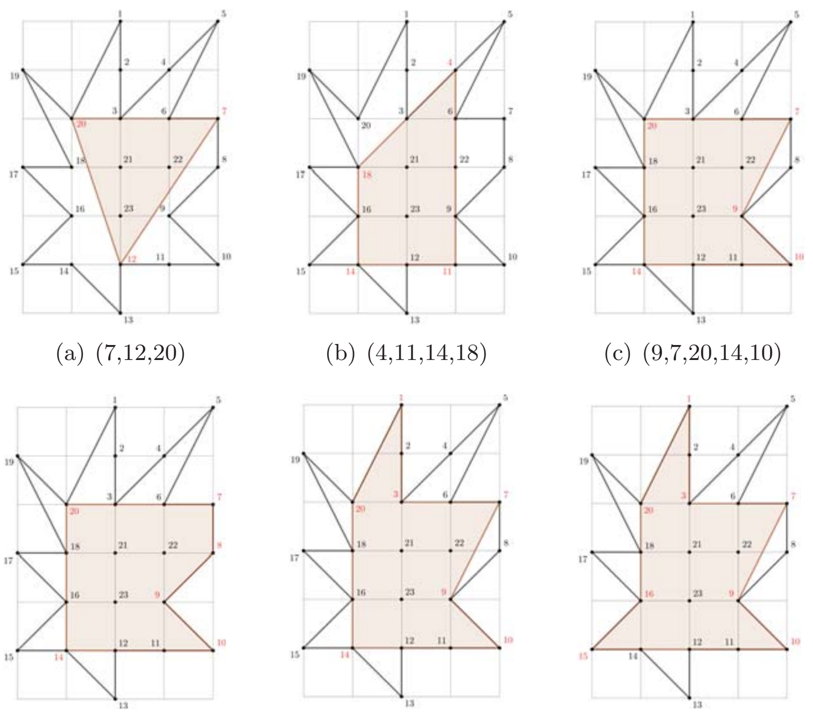
Area, function = {0}.

$k = 3$	Triangle	$\Rightarrow$	SOLUTION (35,2,A,0) = {(7,13,10), (12,31,14)},
$k = 4$	Quadrilateral	$\Rightarrow$	SOLUTION (35,3,A,0) = {(7,16,14,10), (7,31,12,13), (9,15,13,10), (9,15,14,12), (10,29,16,13), (10,29,16,14)},
$k = 5$	Pentagon	$\Rightarrow$	SOLUTION (35,4,A,0) = {(1,33,19,25,28), (7,13,10,9,8)},
$k = 6$	Hexagon	$\Rightarrow$	SOLUTION (35,5,A,0) = {(1,33,19,20,25,28)},
$k = 7$	Heptagon	$\Rightarrow$	SOLUTION (35,6,A,0) = {(1,33,19,25,26,27,28)},
$k = 8$	Octagon	$\Rightarrow$	SOLUTION (35,7,A,0) = {(1,33,19,20,25,26,27,28)}.

Given the above solutions, Figure 13 presents geometrically the solutions for the 5-gon and the 8-gon.

3.4. Practical applications. As previously explained, the algorithm receives as input a closed contour, a value k indicating the number of sides of the polygon, and a value to determine whether to compute the maximum area or the maximum perimeter. Thus, the practical applications of the proposed algorithm are numerous.

This section shows a useful application, oriented to the identification of the maximum area of agricultural plots, either for the installation of solar panels or agricultural greenhouse structures. In both cases, it is of interest to easily identify the surface of the largest area inscribed in a plot (closed contour). This closed contour is previously determined by applications such as Geographic Information Systems (GIS), Cadastral Information Systems (CIS), and similar



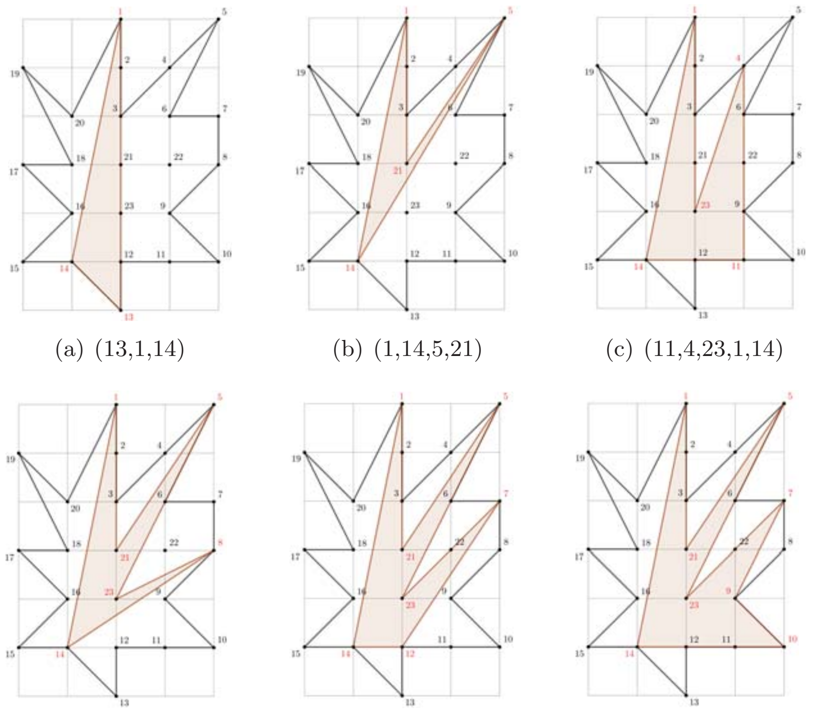
(a) (7,12,20) (b) (4,11,14,18) (c) (9,7,20,14,10)

(d) (9,8,7,20,14,10) (e) (3,1,20,14,10,9,7) (f) (9,7,3,1,20,16,15,10)

Figure 10. *Example 2: Maximum-area simple k-gon.*

approaches. In that case, an ROI was obtained by an external system, which created a vector of closed contour points for the algorithm. Figure 14 shows the location of two agricultural plots used as examples.

The algorithm easily identifies different options from which to choose the best one. This application is illustrated as an example, using only a few nodes for simplicity (Figure 15 and Figure 16).



(a) (13,1,14) (b) (1,14,5,21) (c) (11,4,23,1,14)

(d) (5,23,8,14,1,21) (e) (5,23,7,12,14,1,21) (f) (10,9,7,23,5,21,1,14)

Figure 11. *Example 2: Maximum-perimeter simple k-gon.*

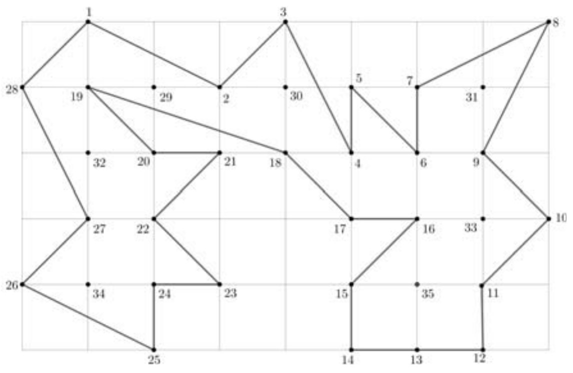
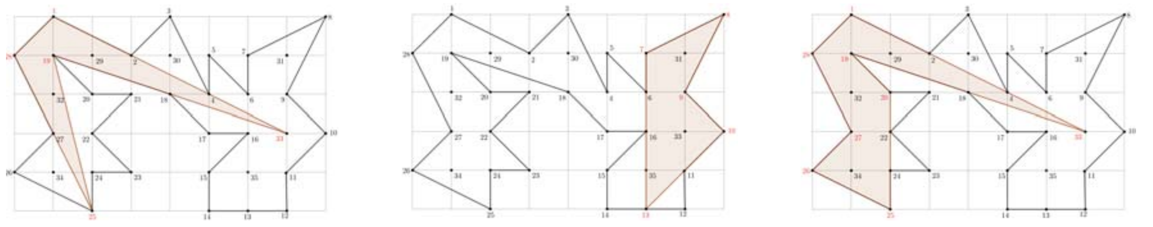
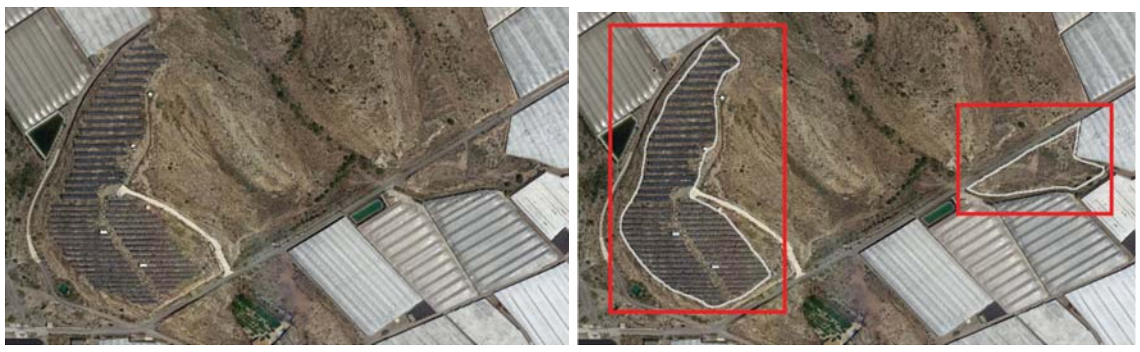


Figure 12. *Example 3: Maximum-area and perimeter simple k-gon.*



(a) (1,33,19,25,28) (b) (7,13,10,9,8) (c) (1,33,19,20,25,26,27,28)

Figure 13. *Example 3: Maximum-area simple 5-gon and 8-gon.*



(a) Original image (b) Closed contours

Figure 14. *Practical application to identify maximum-area plots.*

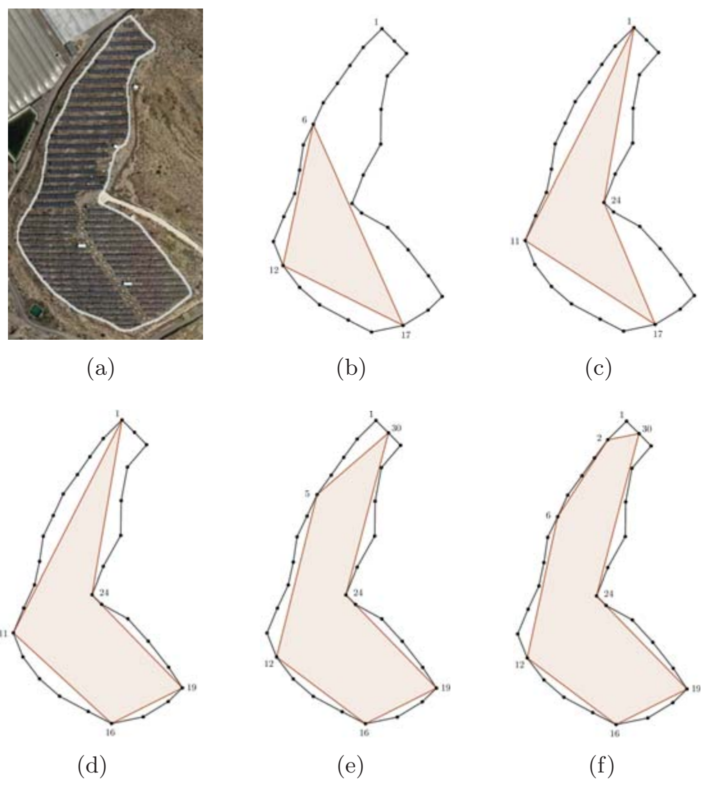


Figure 15. *Closed contour image* (a), *and k-gons of maximum area for* (b) *triangles,* (c) *quadrilaterals,* (d) *pentagons,* (e) *hexagons, and* (f) *heptagons.*

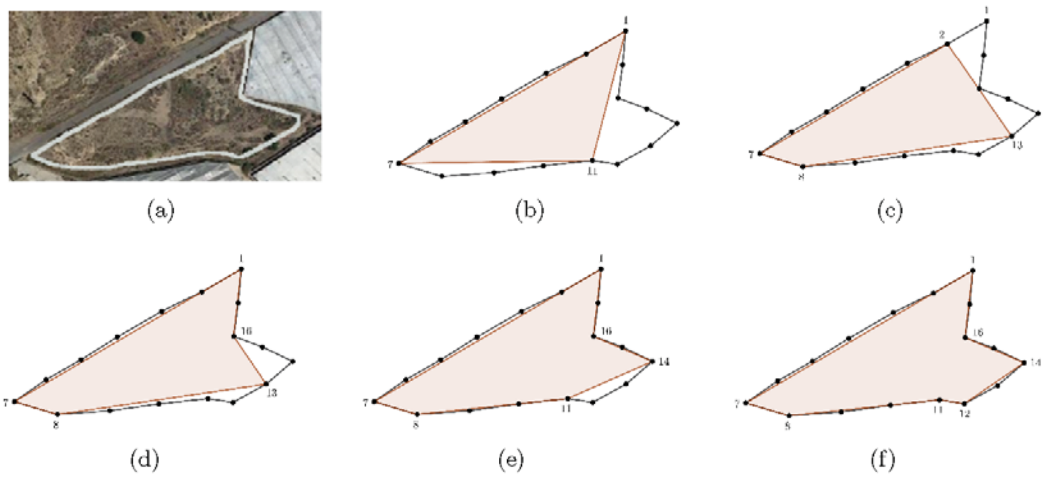


Figure 16. *Closed contour image* (a), *and k-gons of maximum area for* (b) *triangles,* (c) *quadrilaterals,* (d) *pentagons,* (e) *hexagons, and* (f) *heptagons.*

Table 4 shows the output of the algorithm for different configurations. In both cases, adding new points results in larger area surfaces.

Table 4
Result of the algorithm to locate solar panels and greenhouse structures.

Polygon	**Figure 15**		**Figure 16**	
	Solution	**Area**	**Solution**	**Area**
Triangle	(12,6,17)	54.422,5	(7,1,11)	21.800
Quadrilateral	(11,1,24,17)	70.008	(7,2,13,8)	24.776,5
Pentagon	(19,16,11,1,24)	96.480	(8,7,1,16,13)	26.945
Hexagon	(5,30,24,19,16,12)	108.886,5	(11,8,7,1,16,14)	28.952
Heptagon	(6,2,30,24,19,16,12)	114.485,5	(1,7,8,11,12,14,16)	30.040,5

This algorithm could be used for many other purposes, given its versatility and ease of configuration.

4. Approximation for the maximum-area or perimeter simple k-gon in a closed contour.

In Algorithm 3.8, we prove that it is possible to compute the maximum-area or perimeter simple k-gon contained in a lattice polygon P. We now see that the area of a closed contour C can be calculated by the inscribed limit area within the lattice polygon P, constructing finer partitions. With this mathematical proof, we achieve the goal of the paper, since if the lattice polygon P approaches C with finer partitions, the simple k-gon contained in P is also the maximal in area or perimeter within C (Theorem 4.2).

Let R be the rectangle of minimum area that encloses the closed contour C [19], and let Π be a regular partition of R with *partition size* L. We define *lower area* $\underline{A}(C,\Pi)$ and *upper area* $\overline{A}(C,\Pi)$ as the largest area lattice polygon $\underline{P}$ contained in C and the smallest area lattice polygon $\overline{P}$ that contain C, respectively, and both are built by points of G_L (Figure 17). By Pick's theorem [12],

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

We define $\underline{A_k}(C,\Pi)$ and $\overline{A_k}(C,\Pi)$ as the maximum-area simple k-gon contained in $\underline{P}$ and $\overline{P}$, respectively, and compute them with Algorithm 3.8.

Proposition 4.1. *Let C be a closed contour. Then $\underline{A}(C) = \overline{A}(C)$, where*

$$\underline{A}(C) = \sup\{\underline{A}(C,\Pi) : \Pi \textit{ regular partition}\},$$
$$\overline{A}(C) = \inf\{\overline{A}(C,\Pi) : \Pi \textit{ regular partition}\}.$$

The common number is called the Area of C and is denoted by $A(C)$.

Proof. $\underline{A}(C) \leq \overline{A}(C,\Pi)$, and as $\overline{A}(C) \leq \overline{A}(C,\Pi)$ for all Π regular partitions, $\underline{A}(C) \leq \overline{A}(C)$. Moreover, as C is closed and bounded, $\underline{A}(C) = \overline{A}(C)$. ■

Theorem 4.2. *Let C be a closed contour. 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*

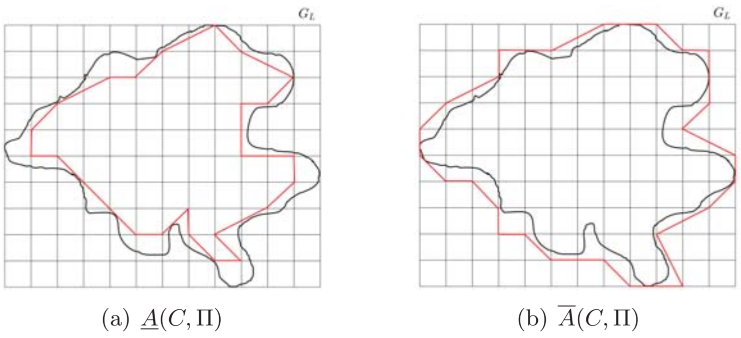


Figure 17. *Lower area and upper area for the regular partition* Π *with partition size* L.

(a) $\lim_{n\to\infty}(\underline{A}(C,\Pi_n)) = A(C)$;

(b) $\lim_{n\to\infty}(\underline{A_k}(C,\Pi_n)) = A_k(C)$, *where* $A_k(C)$ *is the maximum area simple* k*-gon contained in the closed contour* C.

Proof. By Proposition 4.1, $\underline{A}(C) = \overline{A}(C)$; then there exist regular partitions $\dot{\Pi}, \ddot{\Pi}$ such that $|\overline{A}(C,\ddot{\Pi}) - \underline{A}(C,\dot{\Pi})| < \varepsilon$ for all $\varepsilon > 0$. We consider a regular partition Π as finer than $\dot{\Pi}, \ddot{\Pi}$ at once. Then, $\overline{A}(C,\Pi) \leq \overline{A}(C,\ddot{\Pi})$ and $\underline{A}(C,\Pi) \geq \underline{A}(C,\dot{\Pi})$. Therefore, $|\overline{A}(C,\Pi) - \underline{A}(C,\Pi)| \leq |\overline{A}(C,\ddot{\Pi}) - \underline{A}(C,\dot{\Pi})| < \varepsilon$. 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 $\lim_{n\to\infty}(\underline{A}(C,\Pi_n)) = A(C)$. This proves (a).

Moreover, as $\underline{A_k}(C,\Pi) \leq \underline{A}(C,\Pi)$ and $\overline{A_k}(C,\Pi) \leq \overline{A}(C,\Pi)$ for any regular partition, $|\overline{A_k}(C,\Pi) - \underline{A_k}(C,\Pi)| < \varepsilon$. 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 $\lim_{n\to\infty}(\overline{A_k}(C,\Pi_n) - \underline{A_k}(C,\Pi_n)) = 0$, and so $\lim_{n\to\infty}(\underline{A_k}(C,\Pi_n)) = A_k(C)$. ■

Figure 18 shows how the area of a closed contour C can be calculated by the inscribed limit area within the lattice polygon P, constructing finer partitions with $L_{i+1} = L_i/2$ if $\Pi_i \preceq \Pi_{i+1}$ for all i.

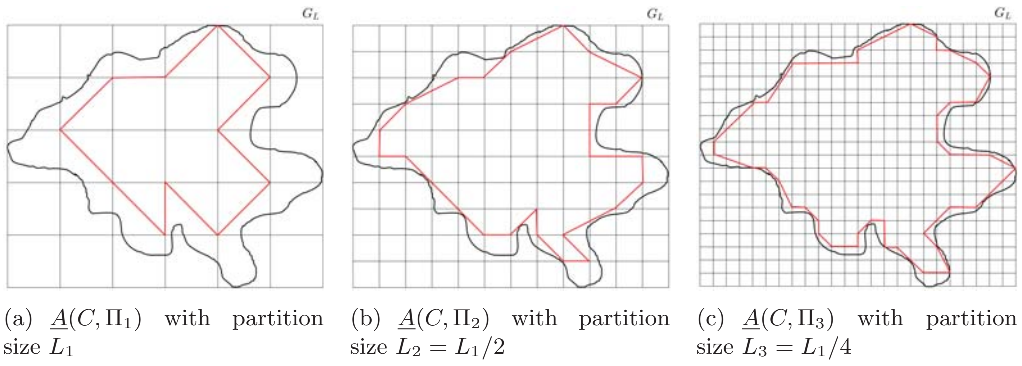

G_L
G_L
G_L
(a) $\underline{A}(C,\Pi_1)$ with partition size L_1
(b) $\underline{A}(C,\Pi_2)$ with partition size $L_2 = L_1/2$
(c) $\underline{A}(C,\Pi_3)$ with partition size $L_3 = L_1/4$

Figure 18. *Lower area for different regular partitions,* $\Pi_1 \preceq \Pi_2 \preceq \Pi_3$.

5. Source code.

All the experiments of this work have been carried out in scripts developed in C/C++, Java, and Python. The source code of this research is fully available for the scientific community on GitHub (C/C++, Java, and Python versions of the proposal) [30], allowing the reproduction of the research presented in this article and the adaptation of the code for other research purposes.

6. Conclusions.

In this article, computational and mathematical models are developed to solve computational geometry problems that can be applied in some areas of engineering, computer science, and computer vision. A new algorithm to compute the maximum-area or perimeter simple k-gon inscribed in a region of interest (a closed contour) and maximum-area or perimeter simple k-gon contained in a simple n-gon is presented. The proposed algorithm can obtain any simple k-gon inscribed on any simple contour. Results of the algorithm have been presented on synthetic examples and in a real-world, practical application to compute the maximum area k-gon in agricultural plots. There is no other algorithm as complete, generic, and constraint-free as the one proposed in this paper. All the pseudocode is presented and explained to allow other scientists to reproduce the algorithm and adapt it for their purposes. Finally, the C/C++, Java, and Python source code of the algorithms, scripts, and technical documents is available for the scientific community in a public GitHub repository.

REFERENCES

[1] A. Aggarwal, J.-S. Chang, and C. K. Yap, *Minimum area circumscribing polygons*, The Visual Computer, 1 (1985), pp. 112–117.

[2] A. Aggarwal, M. M. Klawe, S. Moran, P. Shor, and R. Wilber, *Geometric applications of a matrix-searching algorithm*, Algorithmica, 2 (1987), pp. 195–208.

[3] H. Alt, D. Hsu, and J. Snoeyink, *Computing the largest inscribed isothetic rectangle*, in Proceedings of the 7th Canadian Conference on Computational Geometry (CCCG), Quebec City, Quebec, Canada, 1995, pp. 67–72.

[4] R. Ashraf, S. Afzal, A. U. Rehman, S. Gul, J. Baber, M. Bakhtyar, I. Mehmood, O.-Y. Song, and M. Maqsood, *Region-of-interest based transfer learning assisted framework for skin cancer detection*, IEEE Access, 8 (2020), pp. 147858–147871.

[5] H. Bast and S. Hert, *The area partitioning problem*, in Proceedings of the 12th Canadian Conference on Computational Geometry (CCCG), New Brunswick, Canada, 2000.

[6] V. Bhateja, M. Misra, and S. Urooj, *Human visual system based unsharp masking for enhancement of mammographic images*, J. Comput. Sci., 21 (2017), pp. 387–393.

[7] B. Bhattacharya and A. Mukhopadhyay, *On the minimum perimeter triangle enclosing a convex polygon*, in Japanese Conference on Discrete and Computational Geometry, Springer, Berlin, 2002, pp. 84–96.

[8] A. Biswas, P. Bhowmick, and B. B. Bhattacharya, *Construction of isothetic covers of a digital object: A combinatorial approach*, J. Visual Commun. Image Represent., 21 (2010), pp. 295–310.

[9] R. P. Boland and J. Urrutia, *Finding the largest axis-aligned rectangle in a polygon in $o(nlogn)$ time*, in Proceedings of the 13th Canadian Conference on Computational Geometry (CCCG), Waterloo, Ontario, Canada, 2001, pp. 41–44.

[10] J. E. Boyce, D. P. Dobkin, R. L. Drysdale III, and L. J. Guibas, *Finding extremal polygons*, SIAM J. Comput., 14 (1985), pp. 134–147, https://doi.org/10.1137/0214011.

[11] B. Braden, *The surveyor's area formula*, College Math. J., 17 (1986), pp. 326–337.

[12] M. Bruckheimer and A. Arcavi, *Farey series and Pick's area theorem*, Math. Intelligencer, 17 (1995), pp. 64–67.

[13] J.-S. Chang and C.-K. Yap, *A polynomial solution for potato-peeling and other polygon inclusion and enclosure problems*, in Proceedings of the 25th Annual Symposium on Foundations of Computer

Science, IEEE, Washington, DC, 1984, pp. 408–416.

[14] J.-S. Chang and C.-K. Yap, *A polynomial solution for the potato-peeling problem*, Discrete Comput. Geom., 1 (1986), pp. 155–182.

[15] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, MIT Press, Cambridge, MA, 2009.

[16] K. Daniels, V. Milenkovic, and D. Roth, *Finding the largest area axis-parallel rectangle in a polygon*, Comput. Geom., 7 (1997), pp. 125–148.

[17] A. DePano, *Approximations of Polygons and Polyhedra: Potentials for Research*, manuscript, 1984.

[18] A. DePano, *Polygon Approximation with Optimized Polygonal Enclosures: Applications and Algorithms*, Ph.D. thesis, The Johns Hopkins University, Baltimore, MD, 1987.

[19] H. Freeman and R. Shapira, *Determining the minimum-area encasing rectangle for an arbitrary closed curve*, Comm. ACM, 18 (1975), pp. 409–413.

[20] J. E. Goodman, *On the largest convex polygon contained in a nonconvex n-gon, or how to peel a potato*, Geom. Dedicata, 11 (1981), pp. 99–106.

[21] O. Hall-Holt, M. J. Katz, P. Kumar, J. S. Mitchell, and A. Sityon, *Finding large sticks and potatoes in polygons*, in Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithm, ACM, New York, SIAM, Philadelphia, 2006, pp. 474–483.

[22] S. Hert and V. Lumelsky, *Polygon area decomposition for multiple-robot workspace division*, Internat. J. Comput. Geom. Appl., 8 (1998), pp. 437–466.

[23] K. Jin, *Maximal Parallelograms in Convex Polygons*, preprint, https://arxiv.org/abs/1512.03897v1, 2015.

[24] Y. Kallus, *A Linear-Time Algorithm for the Maximum-Area Inscribed Triangle in a Convex Polygon*, preprint, https://arxiv.org/abs/1706.03049, 2017.

[25] V. Keikha, M. Löffler, A. Mohades, J. Urhausen, and I. van der Hoog, *Maximum-Area Quadrilateral in a Convex Polygon, Revisited*, preprint, https://arxiv.org/abs/1708.00681, 2017.

[26] V. Klee and M. C. Laskowski, *Finding the smallest triangles containing a given convex polygon*, J. Algorithms, 6 (1985), pp. 359–375.

[27] C. Knauer, L. Schlipf, J. M. Schmidt, and H. R. Tiwary, *Largest inscribed rectangles in convex polygons*, J. Discrete Algorithms, 13 (2012), pp. 78–85.

[28] Z. Li, J. D. Wegner, and A. Lucchi, *Topological map extraction from overhead images*, in Proceedings of the IEEE/CVF International Conference on Computer Vision, IEEE, Washington, DC, 2019, pp. 1715–1724.

[29] Z. Li, Q. Xin, Y. Sun, and M. Cao, *A deep learning-based framework for automated extraction of building footprint polygons from very high-resolution aerial imagery*, Remote Sensing, 13 (2021), 3630.

[30] Media Engineering Group (GIM), *Source Code, Scripts, and Documentation*, 2022, https://github.com/UniversidadExtremadura/k-gon (accessed 2022/05/24).

[31] E. A. Melissaratos and D. L. Souvaine, *Shortest paths help solve geometric optimization problems in planar regions*, SIAM J. Comput., 21 (1992), pp. 601–638, https://doi.org/10.1137/0221038.

[32] J. S. Mitchell and V. Polishchuk, *Minimum-perimeter enclosures*, Inform. Process. Lett., 107 (2008), pp. 120–124.

[33] R. Molano, D. Caballero, P. G. Rodríguez, M. D. M. Ávila, J. P. Torres, M. L. Durán, J. C. Sancho, and A. Caro, *Finding the largest volume parallelepipedon of arbitrary orientation in a solid*, IEEE Access, 9 (2021), pp. 103600–103609.

[34] R. Molano, P. G. Rodríguez, A. Caro, and M. L. Durán, *Finding the largest area rectangle of arbitrary orientation in a closed contour*, Appl. Math. Comput., 218 (2012), pp. 9866–9874.

[35] J. O'Rourke, A. Aggarwal, S. Maddila, and M. Baldwin, *An optimal algorithm for finding minimal enclosing triangles*, J. Algorithms, 7 (1986), pp. 258–269.

[36] 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*, Internat. J. Electrical Comput. Engrg., 11 (2021), pp. 4037–4049.

[37] G. Rote, *The Largest Quadrilateral in a Convex Polygon*, preprint, https://arxiv.org/abs/1905.11203v1, 2019.

[38] A. Sarkar, A. Biswas, M. Dutt, and A. Bhattacharya, *Finding a largest rectangle inside a digital object and rectangularization*, J. Comput. System Sci., 95 (2018), pp. 204–217.

[39] X. Sun, W. Zhao, R. V. Maretto, and C. Persello, *Building polygon extraction from aerial images*

and digital surface models with a frame field learning framework, Remote Sensing, 13 (2021), 4700.

[40] G. T. Toussaint, *Pattern recognition and geometrical complexity*, in Proceedings of the 5th International Conference on Pattern Recognition, Miami, FL, 1980, pp. 1324–1347.

[41] I. van der Hoog, V. Keikha, M. Löffler, A. Mohades, and J. Urhausen, *Maximum-area triangle in a convex polygon, revisited*, Inform. Process. Lett., 161 (2020), 105943.

[42] Q. Wei, J. Sun, X. Tan, X. Yao, and Y. Ren, *The simple grid polygon exploration problem*, J. Combin. Optim., 41 (2021), pp. 625–639.

[43] W.-J. Yan and Y.-H. Chen, *Measuring dynamic micro-expressions via feature extraction methods*, J. Comput. Sci., 25 (2018), pp. 318–326.

[44] W. Zhao, C. Persello, and A. Stein, *Building outline delineation: From aerial images to polygons with an improved end-to-end learning framework*, ISPRS J. Photogrammetry Remote Sensing, 175 (2021), pp. 119–131.