Optimal Area Polygonization by Triangulation and Visibility Search

JULIEN LEPAGNOT, LAURENT MOALIC, and DOMINIQUE SCHMITT, Université de Haute-Alsace, IRIMAS UR 7499, France

The aim of the "CG:SHOP Challenge 2019" was to generate optimal area polygonizations of a planar point set. We describe here the algorithm that won the challenge. It is a two-phase algorithm based on the node-insertion move technique, which comes from the TSP. In the first phase, we use constrained triangulations to check efficiently the simplicity of the generated polygonizations. In the second phase, we perform visibility searches to be able to generate a wider variety of polygonizations. In both phases, the simulated annealing metaheuristic is implemented to approach the optimum.

CCS Concepts: • **Theory of computation → Design and analysis of algorithms**; *Computational Geometry;*

Additional Key Words and Phrases: Computational Geometry, geometric optimization, algorithm engineering, polygonalization, area optimization

ACM Reference format:
Julien Lepagnot, Laurent Moalic, and Dominique Schmitt. 2023. Optimal Area Polygonization by Triangulation and Visibility Search. *J. Exp. Algorithmics* 27, 2, Article 2.5 (February 2023), 23 pages.
https://doi.org/10.1145/3503953

1 INTRODUCTION

In this paper we address the following two optimization problems which were the subject of the "CG:SHOP Challenge 2019": Given a set P of points in the plane, find a simple polygon with vertex set P and with minimum (resp. maximum) area. As pointed out in the survey paper of the challenge [7], the problem is $\mathcal{NP}$-Hard. Different exact methods have been studied in [12] for sets with up to 25 points. Since the instances of the competition has up to one million points, it is necessary to implement approximation methods. The algorithm presented here is the one that allowed us to win the competition. Earlier related works are given in [7].

The method we used to deal with both the Min-Area and the Max-Area queries comes from the well known **Traveling Salesman Problem (TSP)**, where it is called *node-insertion move* (see e.g. [24]). The idea is to start with a first polygonization of the given point set. A sequence of polygonizations is then generated by local improvements. To pass from one polygonization to another, a vertex is "moved" from its current position to another one in the sequence of vertices. More precisely, the selected vertex is removed from the current polygonization and is inserted

Authors' address: J. Lepagnot, L. Moalic, and D. Schmitt, Université de Haute-Alsace, IRIMAS UR 7499, 68100 Mulhouse, France; emails: {Julien.Lepagnot, Laurent.Moalic, Dominique.Schmitt}@uha.fr.

1084-6654/2023/02-ART2.5 $15.00
https://doi.org/10.1145/3503953

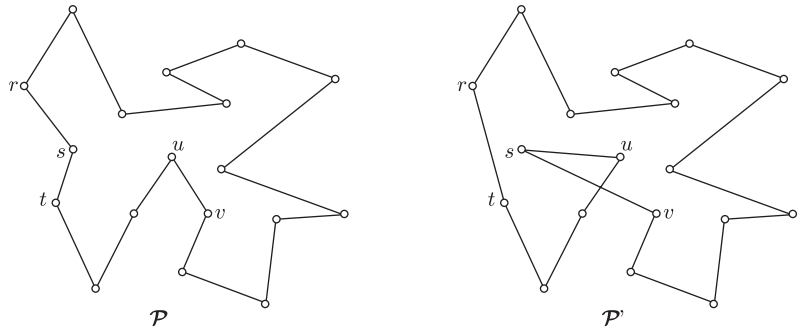


Fig. 1. The node-insertion move of the vertex s of the polygonization $\mathcal{P}$ consists in moving the vertex s from its current position to between the endpoints of an edge uv of $\mathcal{P}$.

between the endpoints of a selected edge of the polygonization. The new polygonization generated that way is in general not simple (see Figure 1).

If we want to handle the area optimization problems by node-insertion moves, we need to find a vertex s and an edge uv in the current polygonization $\mathcal{P}$ with which we can perform a node-insertion move that does not generate intersections. More precisely, if r and t are the neighbors of s on $\mathcal{P}$, the segments rt, su, and sv must intersect no edge in the new polygonization. Finding all such valid pairs (s, uv) is clearly time consuming in large polygonizations. Therefore, for the challenge, we implemented two distinct methods to search for valid pairs. In the first method, only a restricted number of valid pairs is considered. To this aim, we construct a triangulation of the interior and of the exterior of the polygonization $\mathcal{P}$, and consider only pairs (s, uv) such that usv and rst are triangles of the triangulation. In this case, the segments rt, su, and sv cannot generate intersections, implying that no intersection test has to be performed. In the second method, almost all possible valid node-insertion moves are considered. (Figure 3 illustrates the rare valid node-insertion moves that are not considered.) Visibility tests are performed to avoid intersections. It quickly turned out that good results are obtained in a reasonable time by combining the two methods: A first solution is constructed by the first method and is then improved by the second method.

In both phases, if we only perform node-insertion moves that improve the current polygonization, the algorithm falls in a local optimum as shown in Figure 2. Therefore, we must accept node-insertion moves that degrade the current polygonization. We do so by implementing a simulated annealing process. With this scheme, degrading node-insertion moves happen mostly at the beginning of each phase, rarely at the end.

The node-insertion move technique seems to be an appropriate tool to deal with the optimal area polygonization problem since it has been implemented in various forms (and under various names) by all the top five teams of the challenge. However, all of them avoid applying the most general node-insertion move to any polygonization. Like us, Eder et al. [9] use a constrained triangulation to find more easily valid node-insertion moves. But they do not later search for all possible moves. Goren et al. [14] and Ramos et al. [23] introduce a local node-insertion move: only edges issued from the neighbors of the vertex to move are considered for insertion. Ramos et al. apply these local moves to the large instances and limit global moves to instances with at most 30,000 points. Goren et al. decompose the instances in smaller subsets and use global moves to construct good polygonizations of the subsets. The polygonization obtained by merging the sub-polygonizations is then improved by local moves. Crombez et al. [6] construct incrementally a good initial polygonization before applying more general node-insertion moves: not only vertices, but also paths are moved from one location to another. Whereas three teams apply only operations that

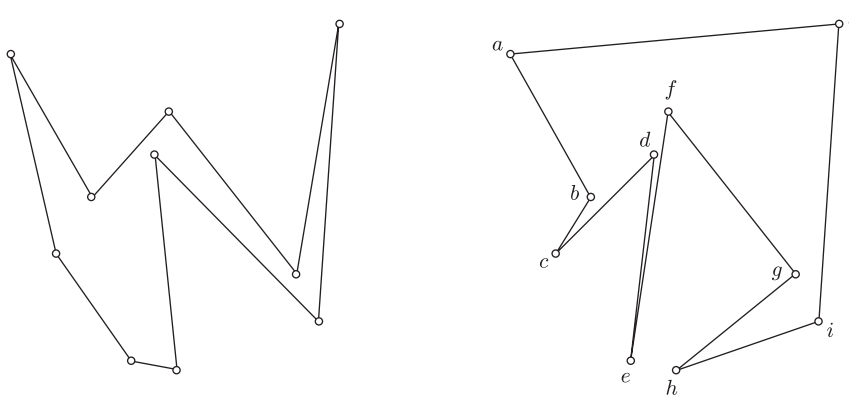


Fig. 2. The figure on the left is the minimal area polygonization of the instance "uniform-0000010-1" of the competition. The figure on the right shows a local optimum: the area of the polygonization is not minimal but it cannot be reduced by a single node-insertion move. Indeed, vertex a cannot be moved, otherwise the segment bj connecting its two neighbors generates intersections. For the same reason, vertices i and j cannot be moved. Moving vertex f leads to adding the triangle efg to the interior of the polygonization. The edges with which the node-insertion move of vertex f can be performed are ja, ab, bc, cd, and de. Thus, one of the triangles jfa, afb, bfc, cfd, and dfe is removed from the interior of the current polygonization. Since each of these triangles has a smaller area than efg, the area of the polygonization increases. The same holds for node-insertion moves of vertices b, d, and g. If we move vertex h, the triangle ghi is removed from the interior of the polygonization. The edges with which this node-insertion move can be performed are ef and fg, leading to adding one of the triangles ehf and fhg to the interior of the polygonization. Since the area of each of these triangles is greater than the area of ghi, the area of the polygonization increases. The same holds for vertices c and e.

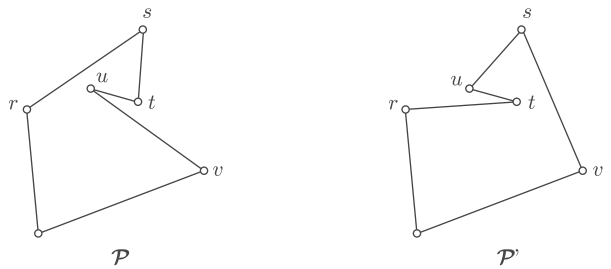


Fig. 3. According to Definition 1, the pair (s, uv) will not be considered as valid in $\mathcal{P}$, even if the node insertion move of (s, uv) generates a polygonization $\mathcal{P}'$ which is simple. Notice also that the triangles rst and usv are neither on both sides of $\mathcal{P}$ nor of $\mathcal{P}'$.

improve the current polygonization, Goren et al. use like us simulated annealing to get out of local optima. A detailed description of simulated annealing applied to the optimal area polygonization problem can be found in their paper.

In the present paper, we first provide, in Section 2, the methods and structures we use to implement the two phases of our algorithm. In Section 3, we show how the different parameters of the simulated annealing have to be set and we study the complexity of the algorithm. In Section 4, we present the results we obtain on different instances of the "CG:SHOP Challenge 2019", both with the parameters used at the competition and with fitted parameters obtained by a grid search after the competition.

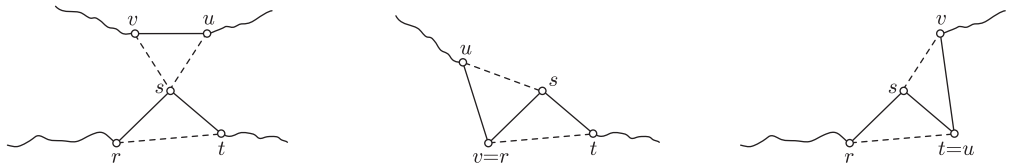


Fig. 4. Three cases where the pairs (s, uv) are valid. The polygonizations (full lines) are not completely drawn.

Source code of our implementations is available on GitHub: https://github.com/lmoalic/SApolygonization.

2 METHODS

2.1 Valid Pairs in Simple Polygonizations

Let P be a set of $n > 3$ distinct points in the plane, not all collinear. A *polygonization* of P is a closed polyline $\mathcal{P}$ whose vertices are the points of P and such that each point of P appears exactly once as a vertex of $\mathcal{P}$. An *edge* of $\mathcal{P}$ is a line segment joining two consecutive vertices of $\mathcal{P}$. An *open edge* is an edge exclusive of its endpoints.

The polygonization $\mathcal{P}$ is said to be *simple* if no point of P belongs to an open edge of $\mathcal{P}$ and if the open edges of $\mathcal{P}$ are pairwise disjoint, i.e., if $\mathcal{P}$ is a Jordan curve. The bounded region of the plane delimited by $\mathcal{P}$ is said to be *inside* $\mathcal{P}$ and the unbounded region is said to be *outside* $\mathcal{P}$. Given any two simple polygonizations $\mathcal{P}$ and $\mathcal{P}'$, and given any point x in the plane that is neither on $\mathcal{P}$ nor on $\mathcal{P}'$, we say that x is on the same side of $\mathcal{P}$ as of $\mathcal{P}'$ if, either x is inside $\mathcal{P}$ and inside $\mathcal{P}'$, or x is outside $\mathcal{P}$ and outside $\mathcal{P}'$. Otherwise, we say that x is not on the same side of $\mathcal{P}$ as of $\mathcal{P}'$ or, indifferently, that x is on the other side of $\mathcal{P}$ as of $\mathcal{P}'$.

Let now $\mathcal{P}$ be a polygonization of P. Let s be a vertex of $\mathcal{P}$, let r, t be the neighbors of s on $\mathcal{P}$, and let uv be an edge of $\mathcal{P}$ such that $s \notin \{u, v\}$. We call *node-insertion move* of the pair (s, uv) in $\mathcal{P}$, the operation that consists in replacing the edges uv, rs, st in $\mathcal{P}$ by the edges rt, us, sv. This operation generates a polygonization $\mathcal{P}'$ of P which is not necessarily simple, even if $\mathcal{P}$ is simple (see Figure 1). In order to find more easily pairs (s, uv) that ensure the simplicity of $\mathcal{P}'$, we impose the following restrictions on these pairs (see also Figure 3).

Definition 1. Given a polygonization $\mathcal{P}$ of P, let s be a vertex of $\mathcal{P}$ with neighbors r, t, and let uv be an edge of $\mathcal{P}$ distinct from rs and st. The pair (s, uv) is called a *valid pair* in $\mathcal{P}$ if rst and usv are non-degenerate triangles whose interiors are disjoint and do not intersect $\mathcal{P}$, and whose open edges either are open edges of $\mathcal{P}$ or do not intersect $\mathcal{P}$.

From this definition, the triangles rst and usv share at most one edge, and the shared edge is neither uv nor rt (see Figure 4).

Furthermore, also by Definition 1, if $\mathcal{P}$ is a simple polygonization of P then the polygonization $\mathcal{P}' = (\mathcal{P} \setminus \{uv, rs, st\}) \cup \{rt, us, sv\}$ obtained by the node-insertion move of (s, uv) in $\mathcal{P}$ is also a simple polygonization of P. The following lemma shows that if (s, uv) is *valid* then the polygon delimited by $\mathcal{P}'$ is obtained from the polygon delimited by $\mathcal{P}$ by removing one of the triangles rst or usv, and by adding the other triangle.

Lemma 2. *Given a simple polygonization $\mathcal{P}$ of P and a valid pair (s, uv) in $\mathcal{P}$, let r, t be the neighbors of s on $\mathcal{P}$ and let $\mathcal{P}'$ be the polygonization obtained by the node-insertion move of (s, uv) in $\mathcal{P}$. Then the following holds:*

(i) The interior of the triangle rst lies entirely either inside or outside $\mathcal{P}$. The interior of the triangle usv lies on the other side of $\mathcal{P}$.

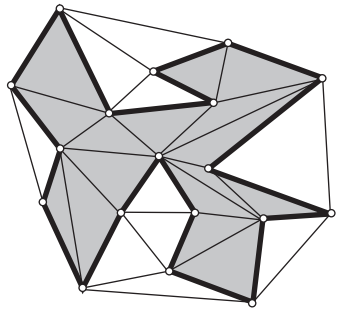


Fig. 5. A colored triangulation (thin lines) constrained by a simple polygonization (thick lines).

(ii) Every point in the interior of the triangle rst or in the interior of the triangle usv is not on the same side of $\mathcal{P}$ as of $\mathcal{P}'$.

(iii) Every point of the plane that belongs neither to rst, nor to usv, nor to $\mathcal{P}$ is on the same side of $\mathcal{P}$ as of $\mathcal{P}'$.

Proof. (i) Since the interiors of rst and of usv are not cut by $\mathcal{P}$, each of them lies entirely either inside or outside $\mathcal{P}$.

Let x be any point in the interior of rst and x' be any point in the interior of usv. The line composed by the two segments xs and sx' intersects $\mathcal{P}$ in precisely one point, the point s. Furthermore, since the interiors of usv and rst are disjoint, this line traverses $\mathcal{P}$ in s. Thus, x and x' are on both sides of $\mathcal{P}$.

(ii) Let Δ be a half straight-line radiating from x and that intersects the open segment rt. We can always choose Δ such that it meets no point of P. By construction, Δ does not intersect the triangle usv. Thus, the intersection points of Δ with $\mathcal{P}$ are the same as the intersection points of Δ with $\mathcal{P}'$, except for the intersection point of Δ with rt. Since rt is an edge of $\mathcal{P}'$ and is not an edge of $\mathcal{P}$, it follows that the number of intersections of Δ with $\mathcal{P}'$ has not the same parity as the number of intersections of Δ with $\mathcal{P}$. Now, the number of intersections of Δ with $\mathcal{P}$ is odd when x is inside $\mathcal{P}$, and is even when x is outside $\mathcal{P}$ (see e.g. [22]). The same holds for $\mathcal{P}'$, implying that x is not on the same side of $\mathcal{P}'$ as of $\mathcal{P}$.

By considering a half straight-line radiating from x' and that intersects the open segment uv, we show in the same way that x' is not on the same side of $\mathcal{P}$ as of $\mathcal{P}'$.

(iii) For every point y in the plane that belongs neither to rst, nor to usv, nor to $\mathcal{P}$, there exits a half straight-line Δ' radiating from y and that intersects neither rst nor usv. It follows that Δ' intersects $\mathcal{P}$ in the same points as $\mathcal{P}'$. Thus, y is on the same side of $\mathcal{P}$ as of $\mathcal{P}'$. □

2.2 Phase 1: Constrained Triangulation

In the first phase of the algorithm, we use the fact that performing node-insertion moves comes to swapping triangles in constrained triangulations.

2.2.1 Swappable Pairs. Given a simple polygonization $\mathcal{P}$ of P, let $\mathcal{T}$ be a triangulation of P constrained by the edges of $\mathcal{P}$, i.e., every edge of $\mathcal{P}$ is an edge of $\mathcal{T}$. Such a triangulation always exists (see e.g. [1]). We call *faces* of $\mathcal{T}$, all the triangles of $\mathcal{T}$ as well as the outer face of $\mathcal{T}$, i.e., the complement of the convex hull conv(P) of P. We say that $\mathcal{T}$ is a *colored triangulation* of P constrained by $\mathcal{P}$, if the triangles of $\mathcal{T}$ inside $\mathcal{P}$ are colored black and the faces outside $\mathcal{P}$ (including the outer face) are colored white (see Figure 5).

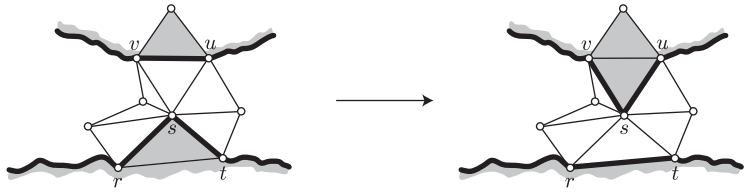


Fig. 6. Performing a node-insertion move of the pair (s, uv) comes to swapping the colors of the triangles rst and usv.

Since every edge of $\mathcal{P}$ is adjacent both to the inside and to the outside of $\mathcal{P}$, it is the common edge of a black triangle and of a white face. Therefore, since every point p of P is a vertex of $\mathcal{P}$, it is the vertex of at least one black triangle and of at least one white face. Furthermore, since p is the vertex of precisely two edges of $\mathcal{P}$, precisely two edges of $\mathcal{T}$ incident in p are shared by a black triangle and by a white face. This implies that all the black triangles incident in p are consecutive around p. In the same way, all white faces incident on p are consecutive around p.

The numbers of black triangles and of white triangles are invariants of P, i.e., they depend neither on the simple polygonization $\mathcal{P}$, nor on the triangulation $\mathcal{T}$. Indeed, it is well known that any triangulation of a simple polygon with n vertices contains $n - 2$ triangles (see e.g. [1]). Thus, $\mathcal{T}$ contains $n - 2$ black triangles. It is also well known that any triangulation of a set of n points contains $2n - n' - 2$ triangles, where n' is the number of vertices of the convex hull of the point set [1]. Thus, $\mathcal{T}$ contains $n - n'$ white triangles, with n' the number of vertices of $\text{conv}(P)$.

Definition 3. Given a colored triangulation $\mathcal{T}$ and two distinct triangles rst and usv of $\mathcal{T}$ incident to the same vertex s, we say that (rst, usv) is a *swappable pair* of triangles in $\mathcal{T}$ if:

— every face of $\mathcal{T}$ incident in s and distinct from rst has a different color than rst,
— the face of $\mathcal{T}$ that shares the edge uv with usv has the same color as rst.

Theorem 4. *Let $\mathcal{P}$ be a simple polygonization of P and let r, s, t, u, v be points of P.*
(i) The pair (s, uv) is a valid pair in $\mathcal{P}$ and r, t are the neighbors of s on $\mathcal{P}$ if and only if there exists a colored triangulation $\mathcal{T}$ of P constrained by the edges of $\mathcal{P}$ in which (rst, usv) is a swappable pair of triangles.
(ii) Furthermore, performing a node-insertion move of (s, uv) in $\mathcal{P}$ comes to swapping the colors of rst and usv in $\mathcal{T}$.

Proof. (i.1) Suppose first that (s, uv) is a valid pair in $\mathcal{P}$ and that r, t are the neighbors of s on $\mathcal{P}$. By Definition 1, rst and usv are non-degenerate triangles whose interiors are disjoint and do not intersect $\mathcal{P}$, and whose open edges either are open edges of $\mathcal{P}$ or do not intersect $\mathcal{P}$. Thus, there exists a colored triangulation $\mathcal{T}$ of P constrained by the edges of $\mathcal{P}$ that admits rst and usv as triangles [1].

Let c be the color of rst. Since rs and st are edges of $\mathcal{P}$, the two faces of $\mathcal{T}$ that share these edges with the triangle rst have not the same color as rst (see Figure 6). Let c' be their color. Now, since all faces of color c' incident in s are consecutive around s, it follows that all faces incident in s other than rst are of color c'. In particular, usv is of color c'. Since uv is an edge of $\mathcal{P}$, the face of $\mathcal{T}$ that shares uv with usv is of color c.

Thus, it follows from Definition 3 that (rst, usv) is a swappable pair of triangles in $\mathcal{T}$.

(i.2) Conversely, suppose that (rst, usv) is a swappable pair of triangles in a colored triangulation $\mathcal{T}$ of P constrained by $\mathcal{P}$. Since rst and usv are distinct triangles of the triangulation $\mathcal{T}$, their interiors are disjoint and do not intersect $\mathcal{P}$, and their open edges either are edges of $\mathcal{P}$ or do not intersect $\mathcal{P}$.

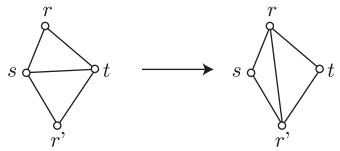


Fig. 7. Flip of the edge st.

Let c be the color of rst. Since, from Definition 3, rst is the unique face of color c incident in s, its two edges rs and st are shared with two faces of $\mathcal{T}$ of color $c' \neq c$. Thus, rs and st are two edges of $\mathcal{P}$, i.e., r and t are the neighbors of s on $\mathcal{P}$. Furthermore, since usv is of color c' and shares its edge uv with a face of color c, uv is an edge of $\mathcal{P}$.

It follows from Definition 1 that (s, uv) is a valid pair in $\mathcal{P}$.

(ii) Let $\mathcal{P}'$ be the polygonization obtained by the node-insertion move of (s, uv) in $\mathcal{P}$. From Lemma 2, the interiors of the triangles rst and usv are not on the same side of $\mathcal{P}'$ as of $\mathcal{P}$. Furthermore, every point of the plane that belongs neither to rst, nor to usv, nor to $\mathcal{P}$ is on the same side of $\mathcal{P}$ as of $\mathcal{P}'$. Thus, if we swap the colors of rst and usv in $\mathcal{T}$, we obtain a colored triangulation of P constrained by $\mathcal{P}'$. □

This theorem shows that if we start with any colored triangulation $\mathcal{T}$ constrained by any simple polygonization of P, we can generate different polygonizations of P by iteratively swapping the colors of pairs of swappable triangles in $\mathcal{T}$. The area of the polygonization induced by $\mathcal{T}$ after a color swap is easily obtained from the area of the previous polygonization, by subtracting the area of the triangle of $\mathcal{T}$ that was black before the swap and by adding the area of the triangle that was white.

2.2.2 Flips. The only polygonizations of P that can be obtained by color swaps are subgraphs of the given triangulation $\mathcal{T}$. If we want to be able to generate other polygonizations, we need to change the triangulation. The *flip* operation is a classical method used to generate different triangulations from a given one.

If we are given two triangles rst and $r'st$ that share an edge st and if $rsr't$ forms a convex quadrilateral, the flip of the edge st consists in replacing st by rr' (see Figure 7). This comes to replace the triangles rst and $r'st$ by the triangles rsr' and rtr'. Edge flips are notably used in optimization algorithms to generate triangulations optimizing various criteria starting from any given triangulation (see [2]).

In particular, it is known that if we start with any triangulation of P constrained by a polygonization $\mathcal{P}$ of P, we can generate all other triangulations of P constrained by $\mathcal{P}$ by iteratively flipping edges that do not belong to $\mathcal{P}$ [20].

This leads to the principle of phase 1:

- generate a colored triangulation of P constrained by an initial polygonization of P;
- perform alternatively sequences of triangle color swaps and sequences of constrained edge flips in the triangulation.

2.2.3 A Simulated Annealing Approach. The principle above is the same whether we search for a polygonization that minimizes its area or for a polygonization that maximizes its area. In the first case, the aim is to converge to a colored triangulation that minimizes the sum of areas of its black triangles. In the second case, the aim is to converge to a colored triangulation that maximizes this sum. In order to ensure the convergence to a good colored triangulation without falling in a local optimum, we must perform, at some times, color swaps that degrade the current colored

triangulation. This is done by a **simulated annealing** metaheuristic **(SA)**. The temperature of the SA determines the probability to accept degrading color swaps. Algorithm 1 gives the main lines of the method.

ALGORITHM 1: phase 1

input : the point set P
output: a simple polygonization $\mathcal{P}$ of P

```
1  construct an initial simple polygonization P of P (see details in Section 3.3)
2  initialize A_init and A_best with the area of P
3  repeat
4      construct a colored triangulation T of P constrained by P (see details in Section 3.3)
5      initialize the temperature t of the SA to 1
6      for i ← 1 to nb_it do
7          repeat nb_cs times
8              perform a sequence of triangle color swaps in the current colored triangulation T
               and maintain continually in T_best the best colored triangulation found so far
9          end
10         perform a sequence of constrained edge flips in the current colored triangulation T
11         reduce the temperature t or reinitialize the SA
12     end
13     update P with the polygonization induced by T_best
14     update A_best with the area of P
15 until A_best is not improved in two consecutive executions of this loop
```

After the construction of an initial polygonization $\mathcal{P}$ of P, the algorithm enters a main loop (Line 3). Each pass in the loop constitutes one step of the algorithm. A step begins by constructing a first colored triangulation $\mathcal{T}$ of the current polygonization $\mathcal{P}$ and by initializing the temperature t of the SA to 1. It then performs nb_{it} iterations composed of nb_{cs} sequences of triangle color swaps and one sequence of edge flips. The number nb_{it} of iterations is proportional to a fractional power of the number n of points in P. The number nb_{cs} of sequences of triangle color swaps per iteration is a constant value. The settings of the parameters of the algorithm will be detailed in Section 3.1.

To perform a sequence of color swaps (Line 8), the points of P are traversed by either increasing or decreasing x-coordinates (the direction alternates at each step of the algorithm). For every traversed point s, we check if there exist swappable pairs of triangles (rst, usv) with vertex s. If so, one of these pairs is randomly chosen and, with probability p_{cs}, the colors of the two triangles are swapped. The probability p_{cs} is determined by adapting a formula classically used in SA [18]. The probability is 1 if the swap improves the current colored triangulation and, otherwise, it equals

$$\frac{1}{1 + e^{\frac{-c_{cs}\ |\delta_A|}{t}}}$$

where:

- t is the current temperature of the SA;
- c_{cs} is a dynamic parameter initialized at the beginning of each step: it depends on the area A_{init} of the initial polygonization and on the current best area A_{best};
- $|\delta_A|$ is the absolute value of the area variation resulting from the ongoing color swap, if it is applied.

To perform one edge flip in Line 10, we randomly select an edge of the current colored triangulation and flip it, if it is flippable. The sequence of edge flips consists in repeating this operation $c_{ef}\ n$ times, where c_{ef} is a constant value.

Once the sequence of edge flips has been performed, the temperature of the SA may be reduced, or the SA may be reinitialized. The temperature is reduced if it has not been changed in the last nb_{tr} iterations of loop 6. Reducing the temperature consists in multiplying it by a real c_{tr}. Both nb_{tr} and c_{tr} are constant parameters.

Now, the lower the temperature is, the lower is the probability p_{cs} to perform degrading color swaps. Thus, at low temperatures, the behavior of the SA is similar to a local search: it is unable to escape a local optimum. This prevents the algorithm from exploring another area of the search space and, possibly, from converging to a better polygonization. To remedy this situation, the SA is regularly reinitialized by reinitializing its temperature and setting the current triangulation to the best colored triangulation $\mathcal{T}_{best}$ found so far. Reinitializing the temperature consists in setting it to $e^{\frac{-i\,c_{it}}{nb_{it}}}$, where i is the iteration number of loop 6 and c_{it} is a constant parameter of the algorithm. The SA is reinitialized if both of the following conditions are satisfied:

- the colored triangulation has not been improved during the last nb_{sc} successive iterations of loop 6 (stagnation criterion), where nb_{sc} is a constant parameter of the algorithm;
- the number of iterations between two consecutive reinitializations is at least equal to $\frac{c_{sar}\ nb_{tr}}{ln(c_{tr})}$, where c_{sar} is another constant parameter (to prevent too frequent reinitializations of the SA).

At the end of each step (Line 13), the polygonization induced by $\mathcal{T}_{best}$ is stored in $\mathcal{P}$ and serves as input for the next step. The algorithm stops if $\mathcal{P}$ has not been improved in two consecutive steps.

To sum up, the phase 1 has 9 parameters that have to be correctly adapted to the instances. Their setting is presented in Section 3.1. A sensitivity analysis of the parameters is presented in Section 3.2 to show which ones are critical and which ones can be left at a default value.

2.3 Phase 2: Visibility Search

In the second phase of the algorithm, we explore all valid pairs and not only those corresponding to swappable pairs in given triangulations. Therefore, this phase is a more exhaustive search than the first one. It is based on visibility.

2.3.1 Valid Pairs and Visibility. Given any two points x, y in the plane and a simple polygonization $\mathcal{P}$, y is said to be *visible from* x (with respect to $\mathcal{P}$) if the open segment xy does not intersect $\mathcal{P}$. A subset E of the plane is said to be (strongly) visible from x (with respect to $\mathcal{P}$) if every point of E is visible from x.

From now on, given a polygonization $\mathcal{P}$ of P and a point s of P, we denote by $\mathcal{P}_s$ the polygonization of $P \setminus \{s\}$ obtained by removing the vertex s from $\mathcal{P}$ and by connecting its two neighbors on $\mathcal{P}$ by an edge. Clearly, if $\mathcal{P}$ is a simple polygonization and if the two neighbors r and t of s on $\mathcal{P}$ are visible from one another with respect to $\mathcal{P}$ then $\mathcal{P}_s$ is also a simple polygonization. It is also not difficult to check that if uv is an edge of $\mathcal{P}_s$ distinct from rt then the pair (s, uv) is a valid pair in $\mathcal{P}$ if and only if r is visible from t with respect to $\mathcal{P}$ and uv is visible from s with respect to $\mathcal{P}_s$. This shows that, if we are given a simple polygonization $\mathcal{P}$ of P and a vertex s of $\mathcal{P}$, and if we want to be able to perform any possible valid node-insertion move with s, we have to:

- check if the neighbors of s on $\mathcal{P}$ are visible from one another,
- find all the edges of $\mathcal{P}_s$ that are visible from s.

Many algorithms have been given to solve these two kinds of visibility requests (see [13]). The first request can be addressed using the (interior and exterior) visibility graph of the polygon $\mathcal{P}$. Two vertices of $\mathcal{P}$ are connected by an edge of this graph if they are visible from one another with respect to $\mathcal{P}$. Choudhury and Inkuluan [5] have proposed an algorithm that maintains the visibility graph of a dynamic simple polygon, i.e., a polygon in which insertion and deletion of vertices are allowed.

The second visibility request can be solved by computing the (interior and exterior) visibility polygon of s in $\mathcal{P}_s$, i.e. the set of points of $\mathcal{P}_s$ that are visible from s relatively to $\mathcal{P}_s$. The visibility polygon of a point in a simple polygon can be constructed in linear time using the algorithm of Joe and Simpson [17] (see also [11, 19] for early versions of the algorithm). Recently, Inkulu et al. [16] proposed an output-sensitive algorithm to maintain this visibility polygon when vertices are added to and removed from the simple polygon. However, this algorithm cannot be used here because it only works if the query point from which the visibility is computed is fixed. It is worthwhile mentioning that in our case we do not need to compute the whole visibility polygon since we are only interested in the edges of $\mathcal{P}_s$ that are *strongly* visible from s. It is not difficult to check that an edge of $\mathcal{P}_s$ is strongly visible from s if and only if its endpoints are visible from s. (Notice that this statement only holds when the convex hull of P is not reduced to a triangle, which is the case for all instances of the competition.) Therefore, the visibility graph suffices to answer our two visibility requests.

2.3.2 *Implementation Techniques.*

Meshing the edges: In practice, we have not implemented the computation and maintenance of the visibility graph. Instead, we have decomposed the point set P by a grid with square mesh cells, and we have maintained the intersections of the edges of the polygonization with the cells of the grid. Checking if two points r and t are visible from one another comes to traverse the cells intersected by the segment rt. The method we use to report the set of edges visible from a given vertex is similar to the one developed by Bungiu et al. [3] and implemented in CGAL[1] (this method uses a constrained triangulation of the point set instead of a grid).

Delta evaluation for the area computation: From Lemma 2, if (s, uv) is a valid pair in a simple polygonization $\mathcal{P}$ and if r, t are the neighbors of s on $\mathcal{P}$ then the triangles rst and usv are on both sides of $\mathcal{P}$. Furthermore, if $\mathcal{P}'$ is the polygonization obtained after the node-insertion move of (s, uv), none of rst and usv is on the same side of $\mathcal{P}'$ as of $\mathcal{P}$. All other points of the plane are on the same side of $\mathcal{P}'$ as of $\mathcal{P}$. It follows that the area of $\mathcal{P}'$ is obtained from the area of $\mathcal{P}$ by subtracting the area of that of the triangles rst and usv which is inside $\mathcal{P}$, and by adding the area of the other triangle. Suppose now that r (resp., t) precedes (resp., follows) s in counterclockwise order on $\mathcal{P}$. The triangle rst is inside $\mathcal{P}$ if and only if rst is oriented in counterclockwise direction, i.e., if and only if its signed area is positive (see Figure 8). The same holds for usv in $\mathcal{P}'$ when u (resp., v) precedes (resp., follows) s in counterclockwise order on $\mathcal{P}'$, i.e., when u precedes v in counterclockwise order on $\mathcal{P}$. Thus, in all cases, the area of $\mathcal{P}'$ is obtained from the area of $\mathcal{P}$ by subtracting the signed area of rst and by adding the signed area of usv.

Maximizing vs minimizing the polygon area: In the challenge, two versions of the optimization problem were proposed, namely maximizing or minimizing the polygon area. Here both problems are considered exactly the same, and computed with the same program. For the maximization we maximize the area, and for the minimization we maximize the opposite of the area. Based

[1]Computational Geometry Algorithms Library http://www.cgal.org.

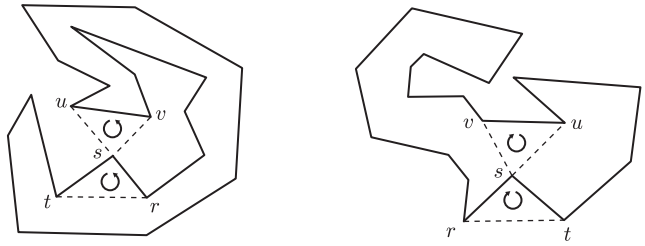


Fig. 8. Delta evaluation of the area.

on the delta evaluation of the area and the fact that a triangle area is considered with its sign, a negative area is obtained by reversing the sequence of vertices of the polygonization. So, for the maximization problem the vertices are taken counterclockwise, and for the minimization the vertices are taken clockwise. That is it.

2.3.3 Algorithm: Again a Simulated Annealing Approach. The starting point of this second phase is a polygonalization $\mathcal{P}$ of the point set P. In practice, we use the result obtained at the end of phase 1, but it could be any polygonalization of P. Based on this starting solution, we will improve (minimize or maximize) the area, by performing node-insertion moves of valid pairs in $\mathcal{P}$.

Given a vertex s of $\mathcal{P}$ and the set of edges visible from s, it is easy to determine with which edge the node-insertion move leads to the best area improvement. The main drawback of always choosing the best local improvement is that it leads to a local optimum, often far from the global optimum. As in phase 1, it appears necessary to add diversity by not always selecting the best possible node-insertion move. To do this, we will again take some principles from the simulated annealing algorithms: accept degrading node-insertion moves at the beginning, but less and less as the algorithm proceeds.

Suppose that s is a vertex of $\mathcal{P}$ whose neighbors r, t are visible from one another, and suppose that the set of edges of $\mathcal{P}_s$ visible from s has been determined. Notice that the edge rt belongs to this set. The edge with which the node-insertion move is performed is selected in respect to a roulette-wheel selection mechanism. The better an edge is, the better chance it has of being selected. For each visible edge uv the signed area of the triangle usv is computed. It will provide the impact of performing the node-insertion move (s, uv). The set of visible edges is then sorted according to these triangle areas. We use the power operator on a random value $r \in [0, 1[$: $a = pow(r, \alpha)$, with α a value that increases along the course of the algorithm. When α becomes big enough, a becomes always 0 implying that the best visible edge is selected. Algorithm 2 gives the main lines of the method.

3 ALGORITHM ENGINEERING

3.1 Parameter Setting of Phase 1

Table 1 summarizes the nine parameters that the user has to set for phase 1. For each parameter, the table gives a relevant interval of values, as well as two empirically fine tuned values: the first was used for the competition and the second, given in the fifth column of the table, was used to obtain the results presented in section 4. The parameter values in the fifth column have been more accurately fine tuned in order to be fitted for the problem instances at hand. The intervals given in the third column are provided for the user wishing to fine tune the parameters of phase 1 for a particular, potentially unusual, instance. Hence, these intervals have been empirically set in order to contain the widest range of relevant values.

ALGORITHM 2: phase 2

input : a starting simple polygonization $\mathcal{P}$ of P
output : an improved simple polygonization of P

reverse the vertex sequence of $\mathcal{P}$ if minimization is required
$\mathcal{P}_{best} \leftarrow \mathcal{P}$
initialize A and A_{best} with the signed area of $\mathcal{P}$
for $\alpha \leftarrow \alpha_0$ **to** α_1 **by** α_{inc} **do**
  repeat nb_{tr} **times**
    select a random vertex s of $\mathcal{P}$
    if *the neighbors* r, t *of* s *are visible from one another* **then**
      generate the set of edges of $\mathcal{P}_s$ visible from s
      select an edge uv in this set depending on α
      if *the selected edge is not* rt **then**
        perform the node-insertion move (s, uv) in $\mathcal{P}$
        update the signed area A of $\mathcal{P}$
        update A_{best} and $\mathcal{P}_{best}$ if A is better than A_{best}
      end
    end
  end
  $\mathcal{P} \leftarrow \mathcal{P}_{best}$
  $A \leftarrow A_{best}$
end

In this table, one can see that the value of nb_{it} is computed according to the number n of points of the instance on which the method is applied. The result of the expression used to compute nb_{it} is rounded to the nearest integer, and limited to stay inside the relevant interval.

The expression used to compute c_{cs} at the beginning of each step depends on the best area A_{best} found in the previous steps and on the area A_{init} of the first polygonization (in the first step, $A_{best} = A_{init}$). The fitted value of c_{cs} is also computed from the standard deviation $\mathcal{T}_{sd}$ of the areas of the triangles in the first colored triangulation constructed by the algorithm.

Due to a lack of time, the parameter settings for the competition have been performed mainly using the "euro-night-0001000" instance, by averaging the results only over five runs per set of parameter values (see [7] for a description of the instances). It led to suboptimal parameter values, that were used to obtain the results sent for the competition. Furthermore, the expression for nb_{it} given in the fourth column is the result of a compromise to accelerate the computation of the results for the competition. It has been voluntarily modified, as well as its interval of relevant values which has been reduced to $(5000, 28000)$, in order to decrease the computation time required by phase 1 to terminate. It is also the case for the parameters nb_{cs} and c_{ef}, which have been reduced in order to accelerate phase 1, at the expense of its effectiveness. However, the purpose of the second optimization process, performed by phase 2 of the proposed method, is to further improve the polygonization found by phase 1 and, thus, to compensate these compromises and suboptimal settings.

After the competition, these compromises have been discarded, and a grid search for the fitted parameter values has been performed, by averaging the results over 20 runs per set of parameter values, using the instances "euro-night-0001000", "skylake-0007000", and "uniform-0030000-2". The resulting fitted values are presented in the fifth column of Table 1.

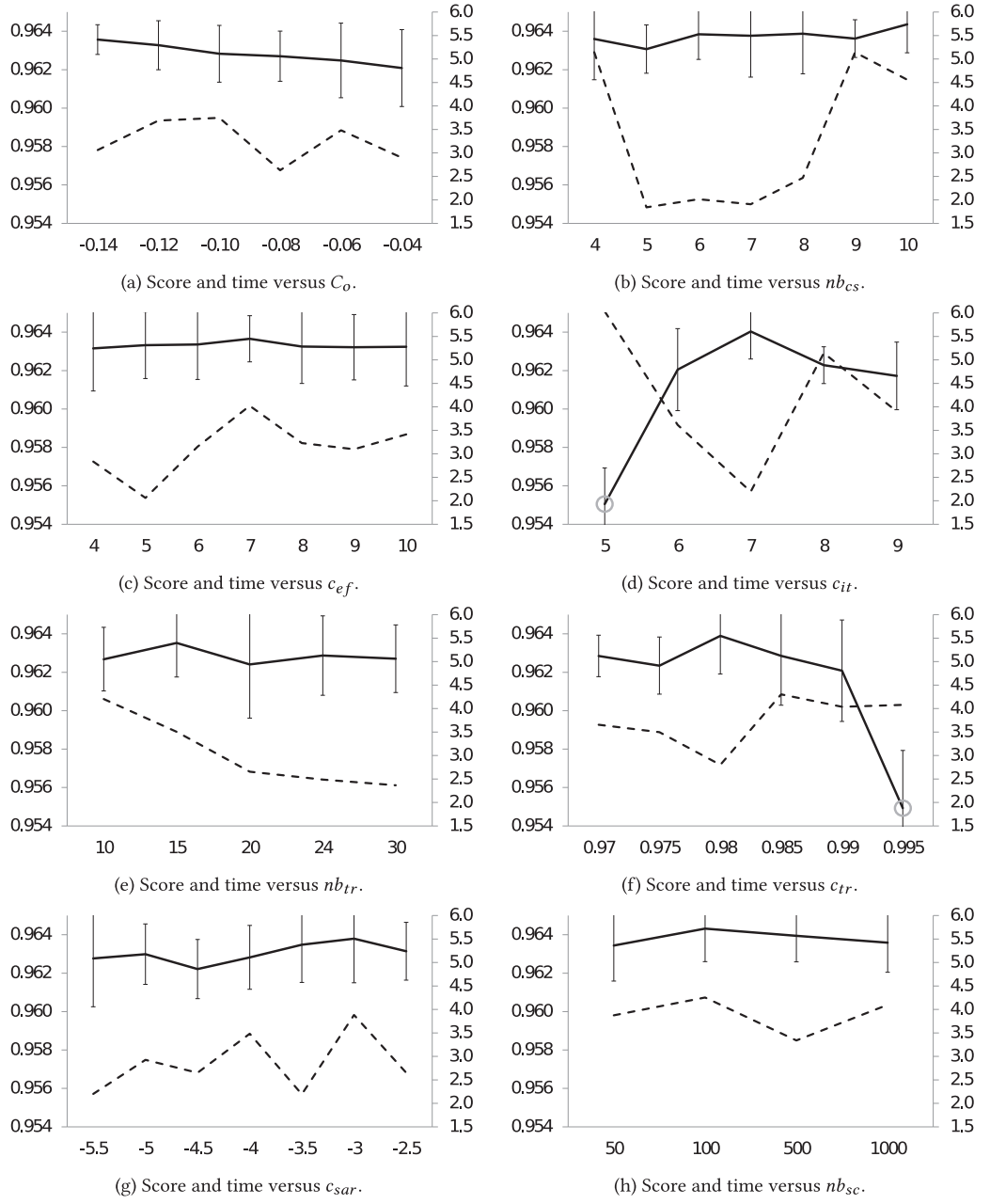
Table 1. Parameter Setting of Phase 1

Name	Type	Interval	Value used for the competition	Fitted value	Short description
c_{cs}	real	$C_c \in$ (-1000, -1) $C_o \in$ (-0.5, -0.002)	$\frac{C_c}{A_{best}}, C_c = -20$	$\frac{C_o\, G}{\mathcal{T}_{sd}}, C_o = -0.08,$ $G = \begin{cases} \frac{A_{best}}{A_{init}} & \text{if the polygonization area is maximized} \\ \frac{A_{init}}{A_{best}} & \text{otherwise} \end{cases}$	coefficient that controls the probability of accepting a degrading color swap
nb_{it}	integer	(5000, 200000)	$764\ n^{0.34}$	$152.6\ \sqrt{n + 10550}$	number of iterations per step
nb_{cs}	integer	(1, 25)	4	8	number of color swap sequences applied per iteration
c_{ef}	real	(1, 50)	3.5	7	coefficient that controls the number of edge flips applied per iteration
c_{it}	real	(1, 15)	6	7	coefficient that controls the initial temperature of the SA when it is reinitialized
nb_{tr}	integer	(1, 30)	10	24	number of iterations between two consecutive temperature reductions
c_{tr}	real	(0.8, 0.995)	0.98	0.99	temperature reduction coefficient
c_{sar}	real	$(-1, -15)$	-7.6	-3	coefficient that controls the minimum number of iterations between two consecutive reinitializations of the SA
nb_{sc}	integer	(10, 1000)	100	100	number of iterations that defines the stagnation criterion which has to be satisfied in order to reinitialize the SA

For the competition, the computation time of an execution of phase 1 was limited to 10 hours in order to get all the results in time. This limitation has also been removed for the new results.

3.2 Sensitivity Analysis of the Parameters of Phase 1

The sensitivity of phase 1 is studied by varying the value of one of its parameters, while the others are left fixed to a default value. Each parameter is studied this way, by applying phase 1 to the Max-Area problem on the instance "us-night-0001000" (a structured instance), and to the Min-Area problem on "uniform-0010000-2" (a larger uniform instance). These instances have been chosen in order to compare the effect of the parameter setting not only between maximization and minimization, but concomitantly on completely different instances (different by nature and by size). The default values used for the fixed parameters are the ones defined in the fifth column of Table 1. For both problems, Figures 9 and 10 show how the *score* of the solutions found by phase 1, averaged over 10 runs, evolves when varying the value of a parameter. As for the evaluation at



(a) Score and time versus C_o.

(b) Score and time versus nb_{cs}.

(c) Score and time versus c_{ef}.

(d) Score and time versus c_{it}.

(e) Score and time versus nb_{tr}.

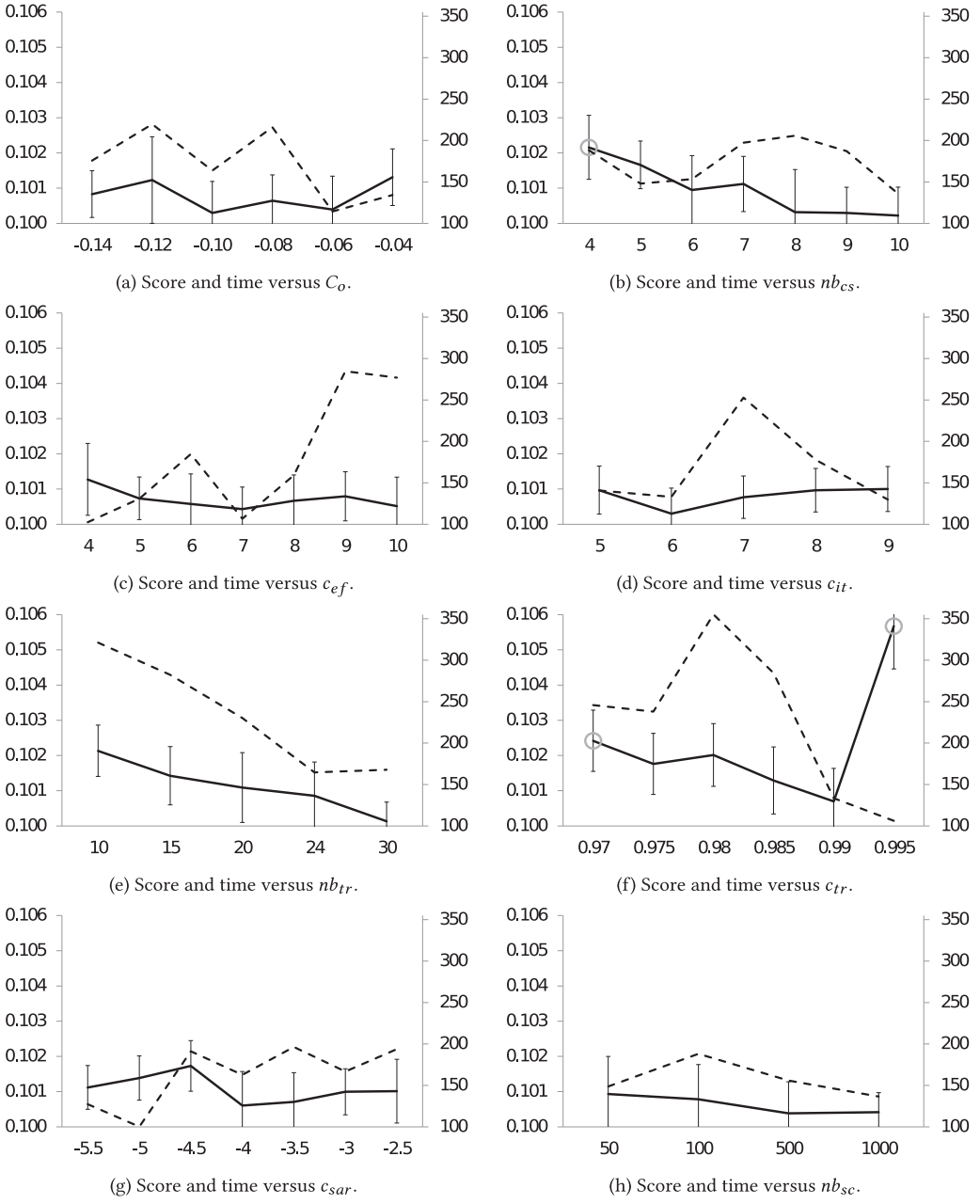
(f) Score and time versus c_{tr}.

(g) Score and time versus c_{sar}.

(h) Score and time versus nb_{sc}.

Fig. 9. Sensitivity for maximization on "us-night-0001000". Each graph represents, using a solid line, the evolution of the score of the solutions found (left ordinate axis), averaged over 10 runs, for several values of a given tested parameter (in abscissa). The vertical bars show the standard deviation. The evolution of the average running time (in minutes on the right ordinate axis) is also depicted by a dashed line.

the competition, the score is obtained by dividing the area of the polygonization by the area of the convex hull of the point set (see [7]). The area of a polygonization being bounded by the area of the convex hull, the score ranges between 0 and 1. When the points are in convex position, the



(a) Score and time versus C_o.

(b) Score and time versus nb_{cs}.

(c) Score and time versus c_{ef}.

(d) Score and time versus c_{it}.

(e) Score and time versus nb_{tr}.

(f) Score and time versus c_{tr}.

(g) Score and time versus c_{sar}.

(h) Score and time versus nb_{sc}.

Fig. 10. Sensitivity for minimization on "uniform-0010000-2". The tested parameters are the same as in Figure 9. The evolution of the average score of the solutions found (left ordinate axis) is depicted by a solid line and the evolution of the average running time (in minutes on the right ordinate axis) is depicted by a dashed line.

convex hull is the polygonization with maximal area. Even for point sets whose convex hull is as large as we like, the minimal area may tend toward 0. The score averaged over 10 runs reflects the performance of phase 1.

For each problem and for each parameter, the Kruskal-Wallis statistical test has been applied. This test determines, at 95% confidence level, if at least one of the parameter values leads to a significantly different performance of phase 1 compared to the other parameter values. Wherever it is the case, the Tukey-Kramer multiple comparisons procedure has been used to determine which parameter values lead to a significantly different performance, in comparison with the default value (given in the fifth column of Table 1). In Figures 9 and 10, the scores that are significantly different from the one obtained using the default parameter value are displayed in a gray circle. The average running time of phase 1 is depicted by a dashed line, showing how it evolves when varying the value of a parameter. It can indeed be taken into account to find a compromise between solution quality and computation time.

In Figure 9, for parameter c_{it}, the parameter value 5 leads to a significantly lower performance than the default value. Also, for the parameter c_{tr}, using a value equal to 0.995 leads to a significantly lower performance. In Figure 10, the parameter values $nb_{cs} = 4$, $c_{tr} = 0.97$, and $c_{tr} = 0.995$ also lead to a significantly lower performance. Hence, even if such values are allowed according to the third column of Table 1, it would be a good choice to select $c_{it} > 5$, $nb_{cs} > 4$, and $0.97 < c_{tr} < 0.995$, in order to get good performance for a wide range of instances.

More generally, we can see from Figures 9 and 10 that the parameters of phase 1 do not need to be precisely fitted. Indeed, using a parameter value in the neighborhood of the default one does not lead to a significant change in the performance of phase 1. From this sensitivity analysis, phase 1 seems to be tolerant to a significantly imprecise setting of the parameters.

3.3 Complexity of Phase 1

The initial polygonization $\mathcal{P}$ constructed in Line 1 of Algorithm 1 is obtained in the following way. Consider the sequence $p_0, p_1, \ldots, p_n$ of vertices of P lexicographically sorted by increasing x and y coordinates. Form a "lower half-polygonization" of P by connecting, in sorted order, the points of P that are strictly below the straight line (p_0, p_n). Form an "upper half-polygonization" of P by connecting the other points of P, also in sorted order. The initial polygonization $\mathcal{P}$ is then obtained by connecting together, on the one hand, the leftmost point of the lower half-polygonization with the leftmost point of the upper half-polygonization and, on the other hand, the rightmost points of the two half-polygonizations.

The colored triangulation $\mathcal{T}$ of Line 4 is constructed by adapting the sweep-line algorithm described in [8]. The plane is swept from left to right by a vertical straight-line Δ. When Δ encounters a vertex p of $\mathcal{P}$, p is connected to the already swept vertices of $\mathcal{P}$ that are visible from p. In order to find the visible vertices efficiently, we maintain a partition of Δ, induced by the intersections of Δ with the edges of $\mathcal{P}$. We construct in this way a triangulation of P constrained by $\mathcal{P}$ in $O(n \log n)$ time.

We note that $\mathcal{T}$ could be constructed in linear time, as follows. We can first construct the convex hull $\text{conv}(\mathcal{P})$ of $\mathcal{P}$ in linear time with the algorithm of Melkman [21]. The boundary of every connected component of $\text{conv}(\mathcal{P}) \setminus \mathcal{P}$ is a simple polygon. Thus, using the algorithm of Chazelle [4], each polygon can be triangulated in linear time. Since the total number of triangles of the triangulations of all these polygons is linear in n, the total number of their vertices is also linear in n (even if each vertex of $\mathcal{P}$ can be a vertex of several polygons). Thus, $\mathcal{T}$ can be constructed in $O(n)$ time. However, the algorithm in $O(n \log n)$ is easier to implement and its complexity is not determinant for the overall complexity of Algorithm 1. Let us also mention two practical polygon triangulation algorithms: The algorithm of Seidel [25] whose expected running time is $O(n \log^* n)$, and the algorithm of Held [15] whose worst case running time is $O(n^2)$ but is fast and robust in practice.

Instead of constructing a new triangulation $\mathcal{T}$ constrained by $\mathcal{P}$ in each step, we could construct an initial triangulation before the first step. Each other step could then reuse the already

Table 2. Number of Improvements and Number of Steps for Minimization in Phase 1

	Number of improvements of $\mathcal{T}_{best}$ per point			Improvements of $\mathcal{T}_{best}$ made in the first step			Number of steps		
Instance	min	max	average	min	max	average	min	max	average
euro-night-0001000	2.48	2.93	2.71	72%	89%	80%	8	28	17.30
euro-night-0004000	2.39	2.89	2.61	75%	94%	85%	4	24	10.50
euro-night-0007000	2.38	2.61	2.47	81%	91%	87%	5	19	10.60
skylake-0001000	2.02	2.36	2.20	85%	99%	90%	4	18	10.70
skylake-0004000	2.11	2.73	2.44	73%	95%	82%	5	33	17.80
skylake-0007000	2.24	2.76	2.35	72%	88%	84%	9	26	14.00
uniform-0001000-2	2.01	2.39	2.24	85%	99%	90%	4	15	8.30
uniform-0004000-2	2.15	2.50	2.32	79%	95%	86%	7	31	16.40
uniform-0007000-2	2.05	2.55	2.31	76%	96%	85%	5	25	14.90
uniform-0030000-2	2.22	2.46	2.33	77%	86%	82%	26	52	35.20

Table 3. Number of Improvements and Number of Steps for Maximization in Phase 1

	Number of improvements of $\mathcal{T}_{best}$ per point			Improvements of $\mathcal{T}_{best}$ made in the first step			Number of steps		
Instance	min	max	average	min	max	average	min	max	average
euro-night-0001000	2.45	3.03	2.75	77%	96%	86%	4	16	9.10
euro-night-0004000	2.34	2.45	2.41	93%	98%	95%	4	9	6.40
euro-night-0007000	2.41	2.59	2.48	90%	97%	95%	4	11	7.50
skylake-0001000	2.21	2.61	2.43	78%	94%	85%	4	25	15.90
skylake-0004000	2.00	2.52	2.24	83%	100%	91%	3	19	8.60
skylake-0007000	2.07	2.24	2.15	90%	98%	93%	4	9	6.40
uniform-0001000-2	2.08	2.62	2.34	81%	98%	87%	6	29	12.70
uniform-0004000-2	2.08	2.29	2.17	86%	95%	91%	4	12	8.00
uniform-0007000-2	2.06	2.31	2.17	86%	95%	90%	5	14	8.50
uniform-0030000-2	1.98	2.11	2.05	91%	96%	93%	14	38	22.80

constructed triangulation $\mathcal{T}_{best}$; recall that $\mathcal{P}$ is precisely the polygonization induced by $\mathcal{T}_{best}$. However, the experiments have shown that better results are obtained when $\mathcal{T}$ is reinitialized at each step.

Each sequence of triangle color swaps, in Line 8, traverses the n vertices of $\mathcal{T}$. For each vertex, the triangles incident in the vertex are traversed to find all pairs of swappable triangles incident in that vertex. Thus, every triangle of $\mathcal{T}$ is traversed three times and at most n color swaps are performed. Each color swap may improve the best triangulation $\mathcal{T}_{best}$, and each update of $\mathcal{T}_{best}$ requires an $O(n)$ copying time. This implies that each sequence of color swaps may need $O(n^2)$ time to update $\mathcal{T}_{best}$.

However, this bound is much lower in practice. Tables 2 and 3 give results obtained by Algorithm 1 for different instances of "euro-night", "skylake", and "uniform-2". The results have been obtained over 10 runs for each instance, both for minimization and for maximization. We observe that the total number of improvements of $\mathcal{T}_{best}$ over all steps of the algorithm varies between $1.98n$ and $3.03n$. Thus, in practice, the total number of improvements of $\mathcal{T}_{best}$ over all steps is in $O(n)$, implying that the overall update time of $\mathcal{T}_{best}$ is $O(n^2)$.

Table 4. Parameter Setting of Phase 2

Name	Type	Value used	Short description
α_0	int	5	Initial temperature
α_1	int	25	Final temperature, for which the algorithm is supposed to choose always the best visible edge
α_{inc}	int	0.01	Temperature increment between two consecutive steps
nb_{tr}	int	$10 * n$	Number of iterations between two consecutive temperature reductions (n is the number of points in the instance)
$nb_{pointsPerCell}$	int	1	Average number of points per grid cell

We notice also that between 72 and 100 percent of the improvements are performed in the first step of the algorithm. Thus, $O(n)$ improvements are performed in the first step. Notice that, from the stopping criterion of the algorithm, no improvement is made in the two last steps.

Since nb_{cs} is a constant, the complexity of the other instructions performed in loop 7 is in $O(n)$ per iteration in loop 6. In the same way, since every sequence of edge flips in Line 10 traverses c_{ef} n edges and since c_{ef} is a constant, the complexity of a sequence of edge flips is in $O(n)$.

The number nb_{it} of iterations was fixed to $O(n^{0.34})$ for the competition, but the experimentally fitted value is $O(n^{0.5})$; see Table 1. Thus, in practice, the complexity of one step of Algorithm 1 varies between:

— $O(n^2)$ for the first step due to the $O(n)$ improvements of $\mathcal{T}_{best}$,
— and $O(n^\alpha)$ for the last step where no improvement is performed, with $\alpha = 1.34$ for the results obtained for the competition, and $\alpha = 1.5$ for the results of Section 4.

With the fitted values of the parameters, the number of steps per run varies between 3 and 52 for the 10 runs per instance mentioned above. Notice that it is mostly greater for minimization than for maximization. We shall return to this difference in Section 5.

3.4 Parameter Setting of Phase 2

Only very few parameters are needed for the phase 2. Table 4 summarizes the parameters used for our experimentation. These parameters were chosen experimentally and were also improved after the competition.

Notice that what we call the temperature here is the α coefficient used in Section 2.3.3. That means that we have $\alpha \in [\alpha_0, \alpha_1]$. When $\alpha = 1$ the selection of a visible edge has a linear certainty related to the improvement. At the opposite, when α increases the certainty of selecting the best visible edge increases too. In other words, the bigger this coefficient is, the most certainty each vertex has to be at its locally best position.

The size of the grid cells is computed in such a way that each cell contains a given average number of points. Suppose that $xMin$, $xMax$, $yMin$, and $yMax$ are the extreme coordinates over all points of P. The cell side c_s is computed as follows:

$$c_s = \sqrt{\frac{(xMax - xMin) * (yMax - yMin)}{n}.nb_{pointsPerCell}}$$

with n the total number of points and $nb_{pointsPerCell}$ the average desired number of points in each cell. Notice that the cell size influences only the execution time but not the quality of the solution. It turns out that minimal execution time is achieved when $nb_{pointsPerCell}$ is set to 1 (at the competition it was set to 10).

Table 5. Key Instructions for the Complexity of Algorithm 2

	Number of improvements of $\mathcal{P}_{best}$		Node-insertion moves performed per point		Number of times test 7 is true	
Instance	Min-Area	Max-Area	Min-Area	Max-Area	Min-Area	Max-Area
us-night-0001000	90.8	96.0	249.3	230.3	79%	78%
us-night-0004000	339.8	527.6	272.8	262.8	80%	78%
euro-night-0004000	284.0	294.8	271.5	271.5	79%	78%
euro-night-0007000	260.2	457.2	302.6	286.7	79%	79%
skylake-0004000	149.4	245.6	277.1	267.9	76%	76%
skylake-0007000	146.8	244.0	309.2	267.4	77%	76%
uniform-0030000-2	331.0	535.4	299.3	324.8	76%	77%

3.5 Complexity of Phase 2

If we do not use any particular data structure, phase 2 runs in $O(n^2)$ worst case time. Indeed, loop 4 of Algorithm 2 is executed a constant number of times and loop 5 is executed $O(n)$ times (see Table 4). Checking whether r is visible from t (Line 7) can obviously be done in $O(n)$ time. Using the algorithm of [17] mentioned in Section 2.3.1, all edges of $\mathcal{P}_s$ visible from s can be reported in $O(n)$ time (Line 8). The node-insertion move (Line 11) and the update of the signed area (Line 12) are performed in constant time. When needed, $\mathcal{P}_{best}$ is updated in $O(n)$ time (Line 13).

If we use the dynamic visibility graph of [5] also mentioned in Section 2.3.1, we need $O(n + e \log e)$ preprocessing time to construct this graph, with e the size of the graph. Notice that e can be as large as $O(n^2)$. In the rest of the algorithm, the overall $O(n^2)$ time complexity of the visibility tests is replaced by an overall $O(k(\log n)^2)$ time complexity for the maintenance of the visibility graph, where k is the total number of changes caused in the graph by the vertex insertions and deletions during the whole algorithm. The overall update time of $\mathcal{P}_{best}$ remains in $O(n^2)$ worst case time.

In practice the total number of improvements of $\mathcal{P}_{best}$ is very low, as shown in the two first columns of Table 5. Recall that, from Table 4, the total number of iterations in loop 5 equals $(\frac{25-5}{0.01} + 1) \times 10 \times n = 20{,}010 \times n$. This low number of improvements is because phase 2 starts with an already good polygonization generated by phase 1. Columns 3 and 4 show that even the effective number of node-insertion moves performed per point is low. For the tested instances, the mean value of this number over 10 runs varies between 230.3 and 309.2, whereas 20,010 attempts where made per point. This comes from the fact that we do not want to degrade too much the already good initial polygonization. Even in the first steps of the simulated annealing process, the roulette-wheel selection mechanism concludes in most cases that the current position of the point under consideration is the best one. In the two last columns of Table 5, we can see that the neighbors of the point under consideration are visible from one another in about 78% of cases. In [10], Eder et al. observed that, over a large family of simple polygons, about 98% of the convex vertices form the bases of ears, i.e., their neighbors are visible from one another. The difference in our case is that we are dealing with particular polygonizations that have small or large areas. (Notice that we are considering also reflex vertices).

It follows from the measures given in Table 5 that the visibility tests made both in lines 7 and 8 are by far the most performed instructions in Algorithm 2. Therefore, it is useful to have a data structure in which these tests can be performed efficiently. The update time of the structure is not crucial, since the polygonization is not often modified. The practical execution time of our algorithm is given in the next section.

Table 6. Best Scores Achieved at the Competition by Each Phase of the Algorithm, and Percentage of Improvement Achieved by Phase 2 Over Phase 1

	Minimization results			Maximization results		
Instance	Phase 1	Phase 2	Improvement	Phase 1	Phase 2	Improvement
us-night-0001000	0.03425	0.03300	3.67%	0.96186	0.96562	0.39%
us-night-0004000	0.02744	0.02662	2.97%	0.96665	0.96983	0.33%
euro-night-0004000	0.04526	0.04362	3.62%	0.94909	0.95282	0.39%
euro-night-0007000	0.03957	0.03886	1.78%	0.95085	0.95452	0.39%
skylake-0004000	0.08768	0.08488	3.19%	0.90775	0.91162	0.43%
skylake-0007000	0.08138	0.07971	2.05%	0.90831	0.91246	0.46%
uniform-0030000-2	0.11475	0.11446	0.24%	0.85119	0.85616	0.58%

Table 7. Best Scores Achieved Over 10 Runs with Fitted Parameters

	Minimization results			Maximization results		
Instance	Phase 1	Phase 2	Impr.	Phase 1	Phase 2	Impr.
us-night-0001000	0.03220 (6.00%)	0.03199 (3.03%)	0.635%	0.96732 (0.57%)	0.96837 (0.28%)	0.108%
us-night-0004000	0.02548 (7.12%)	0.02523 (5.25%)	1.006%	0.97301 (0.66%)	0.97456 (0.49%)	0.159%
euro-night-0004000	0.04082 (9.81%)	0.04075 (6.58%)	0.173%	0.95903 (1.05%)	0.95920 (0.67%)	0.018%
euro-night-0007000	0.03557 (10.11%)	0.03555 (8.53%)	0.054%	0.96426 (1.41%)	0.96442 (1.04%)	0.017%
skylake-0004000	0.07786 (11.20%)	0.07779 (8.36%)	0.096%	0.92090 (1.45%)	0.92091 (1.02%)	0.001%
skylake-0007000	0.07259 (10.81%)	0.07256 (8.97%)	0.036%	0.92708 (2.07%)	0.92719 (1.61%)	0.012%
uniform-0030000-2	0.093693 (18.35%)	0.093691 (18.15%)	0.001%	0.906361 (6.48%)	0.906363 (5.86%)	0.0002%

For each phase, the percentage of improvement over the best score achieved by that phase at the competition is written in parentheses.

4 RESULTS

The best results obtained by the algorithm at the competition for several instances of both Min-Area and Max-Area problems are presented in Table 6. Neither the number of runs nor the execution time was registered at that time. The percentage of improvement achieved by phase 2 over phase 1 is computed as follows:

$$\begin{aligned} 100\ \frac{S_{ph1}-S_{ph2}}{S_{ph1}} &\quad \text{for minimization} \\ 100\ \frac{S_{ph2}-S_{ph1}}{S_{ph1}} &\quad \text{for maximization} \end{aligned}$$

where S_{ph1} and S_{ph2} are the best scores achieved by phase 1 and phase 2 on the same instance, respectively. One can observe that the percentage of improvement is greater for the Min-Area problem than for the Max-Area problem.

Table 7 presents the best scores obtained over 10 runs after the competition with fitted parameters. The percentage of improvement achieved by each phase over the score obtained by the same phase at the competition is computed as follows:

$$\begin{aligned} 100\ \frac{S_{comp}-S_{fit}}{S_{comp}} &\quad \text{for minimization} \\ 100\ \frac{S_{fit}-S_{comp}}{S_{comp}} &\quad \text{for maximization} \end{aligned}$$

where S_{comp} (resp. S_{fit}) is the best score achieved by the considered phase at the competition (resp. with fitted parameters).

First of all, one can notice that the results obtained by phase 1 using fitted parameters are even better than the ones obtained by phase 2 at the competition. It shows the significance of having fitted the phase 1 parameters. Furthermore, the improvement gained using fitted parameters is more and more significant as the number of points of the instance increases. It is also more

Table 8. Running Time of the Algorithm Averaged Over 10 Runs with Fitted Parameters

	Minimization time (min)			Maximization time (min)		
Instance	Phase 1	Phase 2	Total	Phase 1	Phase 2	Total
us-night-0001000	3.0	4.6	7.6	3.3	4.9	8.2
us-night-0004000	24.1	22.9	47.0	16.6	22.9	39.5
euro-night-0004000	31.4	20.0	51.5	28.8	19.6	48.4
euro-night-0007000	55.4	40.0	95.4	37.7	39.5	77.1
skylake-0004000	31.8	15.5	47.4	22.4	16.0	38.4
skylake-0007000	76.8	29.0	105.7	50.3	32.1	82.4
uniform-0030000-2	1912.9	185.9	2098.9	1790.6	184.5	1975.1

Table 9. Best Scores Achieved after 1 hour by Algorithm 2 Compared to the Best Scores Found by Algorithm 1 when both Algorithms are Launched 10 Times on a Same, not Already Good, Initial Polygonization

	Minimization results			Maximization results		
	Algorithm 2	Algorithm 1		Algorithm 2	Algorithm 1	
Instance	Score	Score	Time (min)	Score	Score	Time (min)
us-night-0001000	0.03379	0.03220	5.8	0.96716	0.96732	3.0
us-night-0004000	0.03024	0.02548	19.4	0.97180	0.97301	15.9
euro-night-0004000	0.04871	0.04086	60.0	0.95372	0.95903	16.6
euro-night-0007000	0.04616	0.03557	58.0	0.95525	0.96426	56.9
skylake-0004000	0.09176	0.07786	52.4	0.90764	0.92090	15.9
skylake-0007000	0.08742	0.07259	55.5	0.91231	0.92708	41.4
uniform-0030000-2	0.12956	0.09619	60.0	0.87225	0.90372	60.0

significant for minimization than for maximization, which shows that fitting the parameters of phase 1 had a greater effect on the minimization than on the maximization.

As at the competition, the improvement brought by phase 2 is greater for minimization than for maximization. It suggests a disparity in the behavior of at least one phase of the algorithm between its application for minimization and for maximization.

Table 8 gives the mean time over 10 runs of the algorithm with fitted parameters on an Intel Xeon computer with 2.60GHz clock.

In order to illustrate the interest of having combined the two phases of the algorithm, Table 9 shows the results obtained by Algorithm 2 when it is launched on a not already good polygonization (in fact, the initial polygonization on which Algorithm 1 is launched). For this purpose, the parameters of Algorithm 2 have been tuned so that it explores a wider search space and converges in about one hour. The first and fourth columns of Table 9 report the best scores achieved over 10 runs per instance after 60 minutes per run. The other columns give the best scores found over 10 runs by Algorithm 1 and the times needed to achieve these scores. For the runs whose execution takes more than one hour, the scores are the ones achieved after 60 minutes.

5 OUTLOOK

Our results have been obtained by applying the same algorithm to all instances, for both minimization and maximization. However, it seems that two different methods may be necessary in order to improve the results. As pointed out in sections 3.3 and 4, the phase 1 of the proposed method appears to behave indeed differently depending on whether it is applied to the Min-Area problem or to the Max-Area problem. In particular, the average number of steps per run is mostly greater for minimization than for maximization.

Among the reasons that can explain it, we can point out the expression used to compute the parameter c_{cs} of phase 1, given in Table 1. As a reminder, c_{cs} is the coefficient that controls the probability of accepting a degrading color swap. The expression used to compute c_{cs} for the competition, given in the fourth column of Table 1, depends on A_{best} which is the area of $\mathcal{T}_{best}$, the best colored triangulation found so far. Hence, as $\mathcal{T}_{best}$ is improved by phase 1, c_{cs} either increases if A_{best} is being minimized, or decreases if A_{best} is being maximized.

In order to get better values of c_{cs}, we use the expression given in the fifth column of Table 1. It produces values of c_{cs} that increase as $\mathcal{T}_{best}$ is improved by phase 1, whether A_{best} is being minimized or maximized. However, this expression is different for minimization than for maximization. It produces values of c_{cs} that do not increase in the same way for minimization than for maximization, leading to a difference of behavior of phase 1.

To further improve the proposed method, future works can focus on studying how this difference of behavior can affect the effectiveness of the method. An improved variant of the method can result from this study, showing even more differences in its components, whether it is applied to the Min-Area problem or to the Max-Area problem, than just using a different expression to compute c_{cs}. Two different methods, one for minimization and one for maximization, can even be required.

In [7], a comparison between the algorithms that competed in the challenge is presented. The authors notice that the results obtained by the competing algorithms are more disparate for Min-Area than for Max-Area. This observation reinforces the idea of a possible need to solve the Min-Area problem differently than the Max-Area problem.

As mentioned in the introduction, the node-insertion move technique seems to be particularly well adapted to the generation of families of simple polygonizations. It would thus be useful to prove that any two simple polygonizations can be obtained from one another by a sequence of valid node-insertion moves. One way to prove it would be to show that any polygonization can be transformed that way in a given particular polygonization.

ACKNOWLEDGMENTS

The authors wish to thank Bruno Adam, Mathieu Brévilliers, Lhassane Idoumghar, Julien Kritter, Yvan Maillot, and Hojjat Rakhshani for many helpful discussions during the competition period.

REFERENCES

[1] Jean-Daniel Boissonnat and Mariette Yvinec. 1998. *Algorithmic Geometry*. Cambridge University Press, USA.
[2] Prosenjit Bose and Ferran Hurtado. 2009. Flips in planar graphs. *Computational Geometry* 42, 1 (2009), 60–80.
[3] Francisc Bungiu, Michael Hemmerand, John Hershberger, Kan Huang, and Alexander Kröller. 2014. Efficient computation of visibility polygons. In *30th European Workshop on Computational Geometry*. Ein-Gedi, Israel.
[4] Bernard Chazelle. 1991. Triangulating a simple polygon in linear time. *Discrete Comput. Geom.* 6, 5 (1991), 485–524.
[5] Tameem Choudhury and R. Inkulu. 2019. Maintaining the visibility graph of a dynamic simple polygon. In *Algorithms and Discrete Applied Mathematics*, Sudebkumar Prasant Pal and Ambat Vijayakumar (Eds.). Springer International Publishing, Cham, 42–52.
[6] Loïc Crombez, Guilherme D. da Fonseca, and Yan Gerard. 2020. Greedy and local search solutions to the minimum and maximum area polygons. *Journal of Experimental Algorithmics* (2020), Submitted.
[7] Erik D. Demaine, Sándor P. Fekete, Phillip Keldenich, Dominik Krupke, and Joseph S. B. Mitchell. 2020. Area-Optimal simple polygonalizations: The CG challenge 2019. *Journal of Experimental Algorithmics* (2020), Submitted.
[8] Herbert Edelsbrunner. 2000. Triangulations and meshes in computational geometry. *Acta Numerica* (2000), 133–213.
[9] Günther Eder, Martin Held, Steinthor Jasonarson, Philipp Mayer, and Peter Palfrader. 2020. 2-Opt moves and flips for area-optimal polygonalizations. *Journal of Experimental Algorithmics* (2020), Submitted.
[10] Günther Eder, Martin Held, and Peter Palfrader. 2018. Parallelized ear clipping for the triangulation and constrained Delaunay triangulation of polygons. *Computational Geometry* 73 (2018), 15–23. Secial Issue on EuroCG2015.
[11] Hossam El Gindy and David Avis. 1981. A linear algorithm for computing the visibility polygon from a point. *Journal of Algorithms* 2, 2 (1981), 186–197.

[12] Sándor P. Fekete, Andreas Haas, Phillip Keldenich, Michael Perk, and Arne Schmidt. 2020. Computing area-optimal simple polygonalizations. *Journal of Experimental Algorithmics* (2020), Submitted.
[13] Subir Kumar Ghosh. 2007. *Visibility Algorithms in the Plane*. Cambridge University Press. https://doi.org/10.1017/CBO9780511543340
[14] Nir Goren, Efi Fogel, and Dan Halperin. 2020. Area-optimal polygonization using simulated annealing. *Journal of Experimental Algorithmics* (2020), Submitted.
[15] Martin Held. 2001. FIST: Fast industrial-strength triangulation of polygons. *Algorithmica* 30, 4 (2001), 563–596.
[16] R. Inkulu, K. Sowmya, and Nitish P. Thakur. 2020. Dynamic algorithms for visibility polygons in simple polygons. *International Journal of Computational Geometry & Applications* 30, 1 (2020), 51–78.
[17] Barry Joe and Richard B. Simpson. 1987. Corrections to Lee's visibility polygon algorithm. *BIT Numerical Mathematics* 27 (1987), 458–473.
[18] Scott Kirkpatrick, C. Daniel Gelatt, and Mario P. Vecchi. 1983. Optimization by simulated annealing. *Science* 220, 4598 (1983), 671–680.
[19] Der-Tsai Lee. 1983. Visibility of a simple polygon. *Computer Vision, Graphics, and Image Processing* 22, 2 (1983), 207–221.
[20] Der-Tsai Lee and Arthur K. Lin. 1986. Generalized Delaunay triangulation for planar graphs. *Discrete Comput. Geom.* 1 (1986), 201–217.
[21] Avraham A. Melkman. 1987. On-line construction of the convex hull of a simple polyline. *Inform. Process. Lett.* 25 (1987), 11–12.
[22] Joseph O'Rourke. 1998. *Computational Geometry in C* (2nd ed.). Cambridge University Press, USA.
[23] Natanael Ramos, Rai Caetan de Jesus, Pedro de Rezende, Cid de Souza, and Fabio Luiz Usberti. 2020. Heuristics for area optimal polygonizations. *Journal of Experimental Algorithmics* (2020), Submitted.
[24] Gerhard Reinelt. 1994. *The Traveling Salesman: Computational Solutions for TSP Applications*. Springer-Verlag, Berlin.
[25] Raimund Seidel. 1991. A simple and fast incremental randomized algorithm for computing trapezoidal decompositions and for triangulating polygons. *Computational Geometry* 1, 1 (1991), 51–64.

Received 29 January 2021; revised 30 September 2021; accepted 12 November 2021