Triangle-Based Heuristics for Area Optimal Polygonizations

NATANAEL RAMOS, RAÍ C. DE JESUS, PEDRO J. DE REZENDE, CID C. DE SOUZA, and FÁBIO L. USBERTI, University of Campinas

In this article, we describe an empirical study conducted on problems of Polygonizations with Optimal Area: given a set S of points in the plane, find a simple polygon with minimum (Min-Area) or maximum (Max-Area) area whose vertices are all the points of S. Both problems arise in the context of pattern recognition, image reconstruction, and clustering. Higher dimensional variants play a role in object modeling, as well as optimal surface design. Both Min-Area and Max-Area are in NP-hard, and heuristics as well as exact methods have already been proposed. Our main contributions include the design and implementation of novel constructive heuristics and local search procedures to improve the solutions obtained by the former methods for the two problems.

CCS Concepts: • **Theory of computation → Discrete optimization**; **Computational geometry;**

Additional Key Words and Phrases: Polygonization, computational heuristics, combinatorial optimization, computational geometry

ACM Reference format:
Natanael Ramos, Raí C. de Jesus, Pedro J. de Rezende, Cid C. de Souza, and Fábio L. Usberti. 2022. Triangle-Based Heuristics for Area Optimal Polygonizations. *J. Exp. Algorithmics* 27, 2, Article 2.1 (May 2022), 25 pages.
https://doi.org/10.1145/3504001

1 INTRODUCTION

Problems regarding the construction of geometric objects from a set of points in the plane are often studied in Computational Geometry, including triangulations of point sets, Voronoi diagrams, and so on. [13]. Naturally, one type of objects that can be constructed from a set of points are polygons. Consider, for instance, *optimal polygonizations*, that is, given a set of points S in the plane, we wish to find a polygon $\mathcal{P}$ whose set of vertices is S so that $\mathcal{P}$ optimizes a given function. In particular, one may be interested in maximizing or minimizing the area of the polygon, which gives rise to the Max-Area and Min-Area problems, respectively. These problems are the object of study of this article. We solved both problems using greedy heuristics and local search procedures, where we

This work was supported in part by grants from: *Coordenação de Aperfeiçoamento de Pessoal de Nível Superior* (CAPES), Brazil – Finance Code 001, #88882.329122/2019-01; *Brazilian National Council for Scientific and Technological Development* (CNPq), Brazil, #311140/2014-9, #309627/2017-6, #304727/2014-8, #314384/2018-9, #140300/2020-1, #313329/2020-6; *São Paulo Research Foundation* (FAPESP), Brazil, #2020/09691-0, #2018/26434-0, #2015/11937-9, #2014/12236-1; *Fund for Support to Teaching, Research and Outreach Activities* (FAEPEX), Brazil.

Authors' address: N. Ramos, R. C. de Jesus, P. J. de Rezende, C. C. de Souza, and F. L. Usberti, University of Campinas, Institute of Computing, Av. Albert Einstein, 1251, Cidade Universitária, Campinas, SP, 13083-852, Brazil; emails: nramos@ic.unicamp.br, raicaetanodejesus@gmail.com, {rezende, cid, fusberti}@ic.unicamp.br.

1084-6654/2022/05-ART2.1 $15.00
https://doi.org/10.1145/3504001

used the concept of *polygon triangulations* as the base idea for the design of our algorithms. Most of this work was carried out during the challenge *Computational Geometry: Solving Hard Optimization Problems*. More details regarding both Max-Area and Min-Area, including complexity and related works, as well as further information about the Challenge itself can be found in Demaine et al. [3]. Additional successful heuristic approaches proposed during that Contest are also discussed in this volume [2, 5, 7, 10]. In what follows, we provide notation and definitions used in this text.

Notation and Definitions. Following [13] (see page 18), we define a *polygon* as a finite set $\mathcal{L}$ of line segments such that every segment extreme point (its *endpoint*) is shared by exactly two segments, and no proper subset of $\mathcal{L}$ has the same property. The segments are the *edges* and their extremes are the *vertices* of the polygon. It follows from the above property that the edges of a polygon constitute a cycle. A polygon is *simple* if there is no pair of nonconsecutive edges of this cycle that share a point. By the Jordan Curve Theorem, a simple polygon partitions the plane into two disjoint regions, the *interior* (bounded) and the *exterior* (unbounded), which are separated by the polygon. For additional geometric concepts, we refer the reader to [13].

We define the *boundary* of a simple polygon $\mathcal{P}$ as the aforementioned cycle of its edges, denoted $\partial\mathcal{P}$. Moreover, the *interior* (*exterior*) of $\mathcal{P}$ is denoted by $Int(\mathcal{P})$ ($Ext(\mathcal{P})$). The sets of vertices and edges of $\mathcal{P}$ are indicated by $V(\mathcal{P})$ and $E(\mathcal{P})$, respectively. Throughout the text, when considering a simple polygon $\mathcal{P}$, we usually refer to the union of its boundary and its interior, unless it is necessary to discriminate between the two. The area of $\mathcal{P}$ is denoted by $\mathcal{A}(\mathcal{P})$, and a polygon is said to be *degenerate* when $\mathcal{A}(\mathcal{P}) = 0$. We also adopt the convention that the boundary of a simple polygon is always given in counterclockwise order with respect to its interior.

A simple polygon $\mathcal{P}$ is *convex* if for all $p, q \in V(\mathcal{P})$, the line segment $\overline{pq}$ is entirely contained in $\mathcal{P}$. The *convex hull* of a point set S, denoted $\mathcal{H}(S)$, is the convex polygon of smallest area such that $S \subseteq \mathcal{P}$.

A *diagonal* of a simple polygon $\mathcal{P}$ is a line segment that connects two nonconsecutive vertices of $\mathcal{P}$ and lies entirely in $\mathcal{P}$. A *triangulation* of $\mathcal{P}$ is a decomposition of $\mathcal{P}$ into non degenerate triangles by a maximal set of non crossing diagonals.

Given a set S of points in the plane and a simple polygon $\mathcal{P}$, we say that $\mathcal{P}$ is *empty* w.r.t. S if there are no points of $S \setminus V(\mathcal{P})$ in $\mathcal{P}$. Whenever the point set S is clear from the context, we simply say that $\mathcal{P}$ is empty.

We can now define a *simple polygonization* of a finite point set S as a simple polygon whose vertex set coincides with S. Notice that when all points of S are collinear and $|S| \geq 4$ no simple polygonization exists.

We will also need some concepts regarding the notion of visibility. Given a simple polygon $\mathcal{P}$, let $p \in \mathcal{P}$. Given a point q, we say that p *sees* q when $\overline{pq} \subset \mathcal{P}$ for $q \in \mathcal{P}$, or when $\overline{pq} \cap Int(\mathcal{P}) = \emptyset$ for $q \in Ext(\mathcal{P})$. Given a point $p \in \mathcal{P}$, the set $\{q : q \in \mathcal{P}$ and p sees $q\}$ is the portion of $\mathcal{P}$ visible from p, called the *visibility polygon* of p, denoted by $\mathcal{V}(\mathcal{P}, p)$ [11]. We say that an edge $\overline{qr} \in E(\mathcal{P})$ is *fully visible* from a point p whenever for any $s \in \overline{qr}$, p sees s. As a caveat, we ask the reader to keep in mind, while reading Section 4, that, solely to streamline our implementation, we opted to regard that a vertex of $\mathcal{P}$ does block the visibility between two other points.

Lastly, when speaking about the solution quality achieved by an algorithm (Section 5.2), we often use, as an evaluation metric, the *score* of a polygonization $\mathcal{P}$ of a point set S, computed as $\frac{\mathcal{A}(\mathcal{P})}{\mathcal{A}(\mathcal{H}(S))}$. Note that this value is in the interval $[0, 1]$, since $\mathcal{A}(\mathcal{H}(S))$ is an upper bound for the area of any polygonization.

Our Contribution. In this article, we present heuristic methods developed for both problems Max-Area and Min-Area. Our main results can be outlined as follows:

- The design and implementation of a constructive heuristic for Min-Area and Max-Area, based on iterative addition of triangles (Section 2.1). This method was found to be effective in practice, generating polygonizations with scores of ∼0.78 and ∼0.25 on average, for the Max-Area and Min-Area, respectively (Section 5.2.1);
- A generalization of the approximation algorithm for Max-Area due to Fekete and Pulleyblank [6] where the starting point of the main procedure can be any point in S (Section 2.1);
- An analysis of the quality of the solutions obtained depending on different pivot selection strategies for the proposed constructive heuristic and the aforementioned generalized approximation algorithm (Section 5.2.1).
- The design and implementation of two local search procedures for Max-Area and Min-Area. Both of these are based on swapping triangles inside and outside of the polygonization, but they differ in asymptotic complexity (Section 2.2). Among the two, the method leading to the best results improved the initial solutions by ∼5.70% and ∼25.63% on average, for Max-Area and Min-Area, respectively (Section 5.2.2);
- A discussion of how to handle degeneracies, i.e., the occurrence of three or more colinear points for all proposed techniques. Furthermore, we show that this can be done with minor or no impact at all on the overall asymptotic complexity of the methods (Section 4).

This article is structured as follows. Section 2 is devoted to the presentation of the constructive heuristics (Section 2.1) and local search procedures (Section 2.2). In these sections, as a matter of clarity of exposition, we assume that the points are in general position. Later on, we show how to handle degenerate cases with three or more collinear points. In Section 3, we introduce techniques to improve the running time of our algorithms both theoretically and in practice. All our methods must deal with degenerate situations where three or more points are collinear, so in Section 4, we show how to adapt the algorithms to handle these cases. Next, in Section 5, we describe the computational environment used to conduct our experiments (Section 5.1), followed by the presentation and discussion of the results in Section 5.2.

2 HEURISTICS

2.1 Constructive Heuristics

The constructive heuristics that we developed at the time of our participation in the contest will be presented in detail in this section. These methods are the approximation algorithm by Fekete and Pulleyblank [6] for the Max-Area problem and a novel constructive heuristic proposed by us.

We start by describing the latter in Algorithm 1, which builds a polygonization by adding triangles iteratively. In the description, we assume that we are solving the Max-Area problem, but we show how it can easily be adapted for the Min-Area problem.

This procedure receives as input a set S of n points in the plane and a *pivot* point $p^* \in S$. In Line 1, the points of S are sorted in non-decreasing order according to their Euclidean distance to the pivot, producing a sequence $\mathcal{S}$. Next, in Line 3, we form a triangle $\Delta_{p_1 p_2 p_3}$ with the first three points in $\mathcal{S}$. Since every triangulation of a simple polygon with n vertices has $n-2$ triangles [13], we produce a complete polygonization by executing Lines 5 to 12 exactly $n-3$ times, which adds, in the ith time around, a new triangle to a partial polygonization $\mathcal{P}_i$. In each iteration, the next point p from $\mathcal{S}$ is fetched and in Line 10 we determine the set $\mathcal{V}_E$ of edges of $\mathcal{P}_i$ that are fully visible from p. In Line 11, we search for a triangle with largest area in the set $\{\Delta_{pqr} : (q,r) \in \mathcal{V}_E\}$. Let p, q^* and r^* be the vertices of such triangle. Next, in Line 12, p is inserted between q^* and r^* in $\mathcal{P}_i$. This process adds a large triangle to the current polygon while maintaining its simplicity. This heuristic can be adapted to the Min-Area by adding, instead, the triangle of smallest area in each iteration. To simplify our presentation, we will henceforth refer to the heuristic in Algorithm 1 as Prox-H.

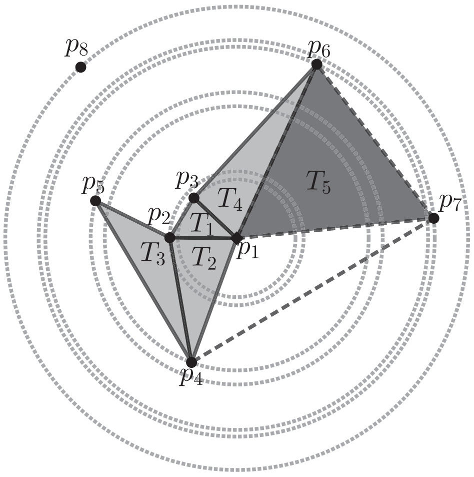


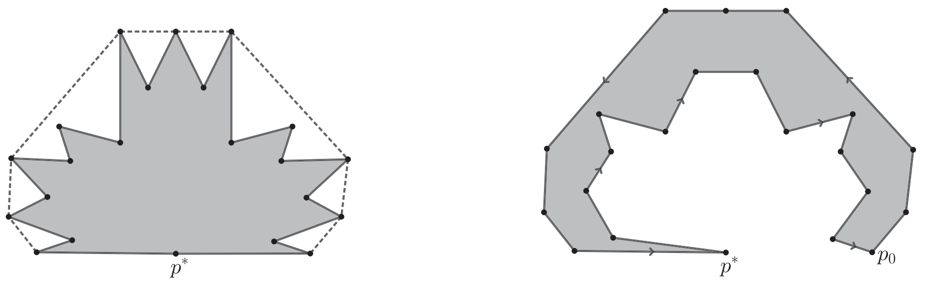
Fig. 1. Example of an execution of the proximity heuristic, where p_1 is the pivot and p_7 is the point being added. The triangles formed with edges from the set $\mathcal{V}_E$ are indicated by dashed segments.

ALGORITHM 1: Proximity Heuristic.

```
    Input   : A set $S$ of $n$ points, $p^* \in S$
    Output  : $\mathcal{P}$
 1  $\mathcal{S} = \langle p_1, p_2, \ldots, p_n \rangle$: sequence of points of $S$ sorted in non-decreasing order by distance to $p^*$
 2  $i \leftarrow 0$
 3  $\mathcal{P}_i \leftarrow \Delta_{\mathcal{S}[1]\mathcal{S}[2]\mathcal{S}[3]}$
 4  $j \leftarrow 4$
 5  while $i \leq n - 3$ do
 6      $i \leftarrow i + 1$
 7      $p \leftarrow \mathcal{S}[j]$
 8      $j \leftarrow j + 1$
 9      $\mathcal{A}^* \leftarrow -\infty$
10      Let $\mathcal{V}_E$ be the set of edges of $\mathcal{P}_i$ fully visible from $p$
11      $(q^*, r^*) \leftarrow$ argmax $\{\mathcal{A}(\Delta_{pqr}) : (q, r) \in \mathcal{V}_E\}$
12      Insert point $p$ between $q^*$ and $r^*$ in $\mathcal{P}_i$
13  return $\mathcal{P}_i$
```

To illustrate one iteration of the heuristic, refer to Figure 1 and consider $p^* = p_1$ and p_7 as the point being added. The circles indicate the distances of the respective points to p^*. The triangles in the current partial polygonization $\mathcal{P}_i$ (in light gray) are labeled T_i, indicating the iteration i in which they were added. There are two edges fully visible from p_7 ($\overline{p_4p_1}$ and $\overline{p_1p_6}$), yielding the triangles $\Delta_{p_7p_4p_1}$ and $\Delta_{p_7p_1p_6}$, indicated with dotted lines. Note that the latter is the one with the largest area. Therefore, it is the one that gets added to the polygonization and labeled as T_5 (in dark gray).

We now argue that this heuristic always yields a feasible solution. From a known result on *segment separation* (see *Lemma* 8.7.1 in [12]), there is always at least one edge in $\mathcal{P}_i$ fully visible from the new point p. Since only fully visible edges are considered, the polygonization remains simple. As a consequence of the initial sorting step, all the triangles added are empty with respect



(a) Star-shaped polygon. (b) Everted polygon.

Fig. 2. Example of solution generated by the approximation algorithm [6].

to S. Hence, at the end of the process, there are no interior points in the resulting (simple) polygonization.

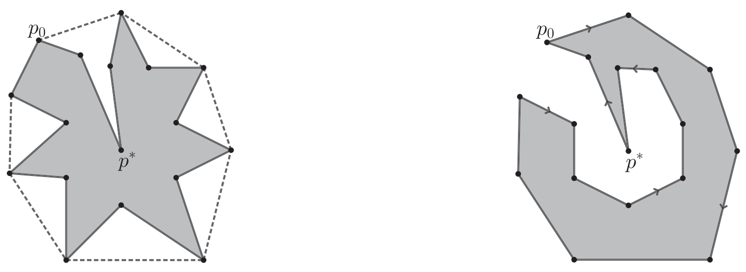
Regarding the complexity of Algorithm 1, the sorting step in Line 1 takes $O(n \lg n)$ time. At each iteration of the loop in Lines 5 to 12, the edges fully visible from p are computed. A naive algorithm to do this could take $\Theta(n^2)$ time per iteration. However, in Section 3, we show how to compute the set $\mathcal{V}_E$ in $O(n \lg n)$ time in the worst case. Moreover, the search performed in Line 11 has $O(n)$ complexity. The main loop is executed $O(n)$ times, so it has complexity $O(n^2 \lg n)$. Lastly, the heuristic has overall complexity $O(n \lg n + n^2 \lg n) = O(n^2 \lg n)$ in the worst case.

Note that, in Algorithm 1, the choice of the pivot point p^* has influence on the magnitude of the area obtained. Different choices for the pivot were considered in our experiments and further details are given in Section 5.2.

Another method used to construct feasible solutions was the approximation to Max-Area proposed by Fekete and Pulleyblank [6]. The algorithm is guaranteed to lead to a simple polygonization with area at least $\frac{1}{2}\mathcal{A}(\mathcal{H}(S))$. Basically, given a set S of points, this algorithm takes a pivot point p^* in $\partial\mathcal{H}(S)$ and sorts the remaining ones in non-increasing counterclockwise angular order with respect to p^*. If the points are connected in this order, a *star-shaped* polygon $\mathcal{P}^s$ is obtained, as illustrated in Figure 2(a), where $p_0 = p^*$. Then, if $\mathcal{A}(\mathcal{P}^s) > \frac{1}{2}\mathcal{A}(\mathcal{H}(S))$, the algorithm stops and returns $\mathcal{P}^s$ as a solution. Otherwise, another simple polygonization is built, referred to as the *everted polygon* $\mathcal{P}^e$: starting in p^*, going through the points in $V(\mathcal{P}^s) \setminus V(\mathcal{H}(S))$ in clockwise order and continuing from the last point visited in the previous clockwise walk, through the points in $V(\mathcal{P}^s) \cap V(\mathcal{H}(S))$ in counterclockwise order. Continuing with our example, with the same set of points, $\mathcal{P}^e$ can be seen in Figure 2(b). Since $\mathcal{A}(\mathcal{P}^s) \leq \frac{1}{2}\mathcal{A}(\mathcal{H}(S))$, it is certainly the case that $\mathcal{A}(\mathcal{P}^e) \geq \frac{1}{2}\mathcal{A}(\mathcal{H}(S))$, so $\mathcal{P}^e$ is returned as a solution. This algorithm can be adapted to the Min-Area problem, by choosing the polygon with the smallest area between $\mathcal{P}^s$ and $\mathcal{P}^e$. Nonetheless, there is no theoretical guarantee on the value of the solution, as in the case of Max-Area. This algorithm has time complexity of $O(n \lg n)$, due to the initial sorting of the points. For simplicity of exposition, in what follows, we will refer to this approximation algorithm as APX-FP.

Note that the choice of the pivot p^* directly influences the magnitude of the area obtained. More details on the impact of this decision in the quality of the final solution are given in Section 5.2.

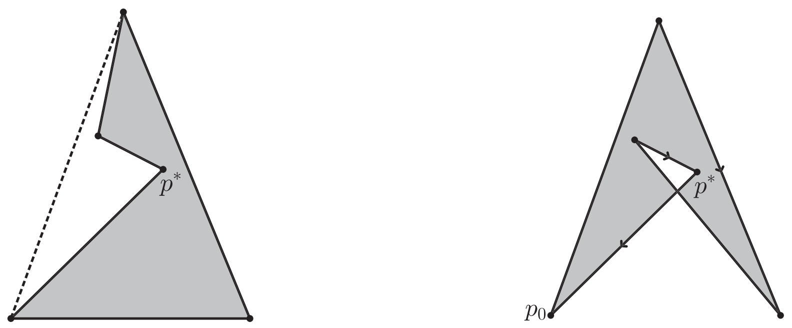
Generalization. As presented in [6], APX-FP starts from a point $p^* \in \partial\mathcal{H}(S)$. We now present a generalization where p^* can be any point of S. First, the star-shaped polygon $\mathcal{P}^s$ is built as in the first phase of the APX-FP algorithm, as depicted in Figure 3(a). Next, the everted polygon $\mathcal{P}^e$ is constructed as follows. Traverse $\mathcal{P}^s$ in counterclockwise order, starting at p^*, until a vertex p_0 of $\mathcal{H}(S)$ is found. Let $V(C_{p^*}^{p_0})$ denote the set of vertices of this polygonal chain from p^* to p_0. Next,



(a) Star-shaped polygon.

(b) Everted polygon.

Fig. 3. Example of solution generated by the generalization of APX-FP.



(a) Star-shaped polygon.

(b) Everted polygon.

Fig. 4. Example of a non simple everted polygon.

walk in clockwise order along $V(\mathcal{P}^s) \cap V(\mathcal{H}(S))$ and then, starting from the last point visited in this path, through $V(\mathcal{P}^s) \setminus (V(\mathcal{H}(S)) \cup V(C_{p^*}^{p_0}))$ up to p^* in counterclockwise order. The everted polygon, for the same set of points, is presented in Figure 3(b). Clearly, the APX-FP algorithm can be seen as a restricted case of this one, for $p^* = p_0$.

Note that the everted polygon $\mathcal{P}^e$ may not be simple. Figure 4 shows an example where this happens. In this case, the conditions that guarantee the simplicity of $\mathcal{P}^e$, established by Fekete and Pulleyblank [6], are violated. In these circumstances, $\mathcal{P}^s$ is returned as a solution. In Section 5.2.1, we show that, in the majority of instances used in our experiments, the generalization—employing all points of S as pivots and retaining the best solution among all at the end—yields better results than APX-FP using all points in $V(\mathcal{H}(S))$ as pivots. We denote this method by GAPX-FP in the rest of the document.

2.2 Local Search Procedures

In this section, we describe the two local search procedures developed for both Max-Area and Min-Area. One of the methods is more far-sighted than the other, albeit with a higher asymptotic complexity. Let us start by presenting the broader one, described in Algorithm 2. Together with it, the reader could benefit from referring to Figure 6 for a better comprehension of the main steps of the algorithm. Along the lines of the previous section, our description focuses on the Max-Area problem and follows by presenting the necessary modifications to adapt it to Min-Area. Given a polygonization $\mathcal{P}$, consider its internal triangles anchored on a reflex vertex. In short, the local

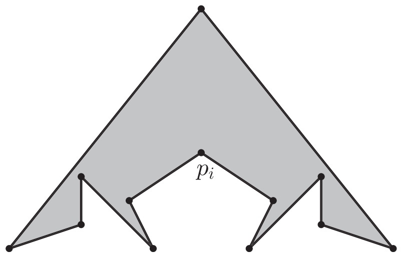


Fig. 5. An example where $\mathcal{V}_E$ is empty in Line 6 of Algorithm 2.

ALGORITHM 2: Local Search Algorithm.

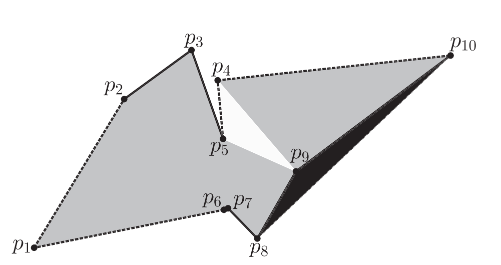
```
Input    : $\mathcal{P}, S$
1  $\mathcal{A}' \leftarrow -\infty$
2  while $\mathcal{A}(\mathcal{P}) > \mathcal{A}'$ do
3      $\mathcal{A}' \leftarrow \mathcal{A}(\mathcal{P})$
4      for each reflex vertex $p_i$ of $\mathcal{P}$ do
5          if $\Delta_{p_{i-1}p_ip_{i+1}}$ is empty with respect to $S$ then
6              Let $\mathcal{V}_E$ be the set of edges of $\mathcal{P}$ fully visible from $p_i$
7              if $\mathcal{V}_E \neq \emptyset$ then
8                  $(q^*, r^*) \leftarrow \text{argmin}\{\mathcal{A}(\Delta_{qp_ir}) : (q,r) \in \mathcal{V}_E, \mathcal{A}(\Delta_{qp_ir}) > 0\}$
9                  if $\mathcal{A}(\Delta_{p_{i-1}p_ip_{i+1}}) > \mathcal{A}(\Delta_{q^*p_ir^*})$ then
10                     Remove $p_i$ from $\mathcal{P}$
11                     Insert $p_i$ between $q^*$ and $r^*$ in $\mathcal{P}$
```

search successively inspects such triangles, seeking to exchange them for triangles of larger area external to $\mathcal{P}$, while preserving simplicity. This iterative process comprises Lines 4 to 11. There, the vertices of $\mathcal{P}$ are traversed in counterclockwise order, starting from the first vertex given by the input's order. First, in Line 5, we check whether the triangle $\Delta_{p_{i-1}p_ip_{i+1}}$ is empty with respect to S and, if that is the case, we start to search for an internal triangle to be removed. To do so, in Line 6, the set $\mathcal{V}_E$ of edges of $\mathcal{P}$ that are fully visible from p_i, except from those incidents on it, is computed. Note that, unlike in Algorithm 1, $\mathcal{V}_E$ may be empty here, as illustrated in Figure 5. Hence, we need an additional test that is performed in Line 7.

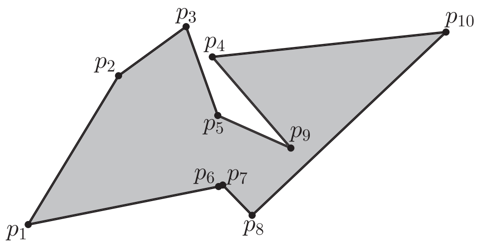
In Line 8, we search for a triangle with smallest area in $\{\Delta_{p_iqr} : (q,r) \in \mathcal{V}_E, \mathcal{A}(\Delta_{p_iqr}) > 0\}$ to be considered for removal from $\mathcal{P}$. Note that the condition $\mathcal{A}(\Delta_{p_iqr}) > 0$ ensures that we do not add degenerate triangles. Let this triangle be formed by points p_i, q^* and r^*. Lastly, in Line 9, if the triangle to be inserted has a larger area than the one to be removed, the two are exchanged, moving p_i to the position between q^* and r^* in $\mathcal{P}$. This process, described in Lines 2 to 11, is repeated until no further improvements in area can be achieved by these local moves.

Note that this local search can be adapted to Min-Area in the following way. When looking for a triangle to be inserted (removed), consider the one with smallest (largest) area among all candidates. The exchange takes place whenever there is gain according to the Min-Area objective.

Figure 6 shows an example of a triangle exchange as performed by Algorithm 2. The polygonization from Figure 6(a) was generated by the proximity heuristic (Algorithm 1) and has a score of 0.77. Considering the reflex vertex p_9, the triangle $\Delta_{p_8p_9p_{10}}$ (in dark gray) is empty, so it is a candidate for insertion. The edges of the set $\mathcal{V}_E = \{\overline{p_1p_2}, \overline{p_4p_5}, \overline{p_4p_{10}}, \overline{p_1p_6}\}$ are the dashed ones. The triangle with smallest area among those formed by p_9 and edges in $\mathcal{V}_E$ is $\Delta_{p_9p_5p_4}$ (in white). Since



(a) Solution generated by Algorithm 1. The triangle $\Delta_{p_8p_9p_{10}}(\Delta_{p_5p_9p_4})$ will be inserted (removed).



(b) Solution generated by the execution of one local move.

Fig. 6. Example of an exchange performed by the local search.

the area of $\Delta_{p_8p_9p_{10}}$ is larger than that of $\Delta_{p_9p_5p_4}$, the exchange is performed, as shown in Figure 6(b). The score of the new polygonization is 0.79.

Regarding the complexity of Algorithm 2, the loop from Lines 4 to 11 is executed $O(n)$ times, since the number of reflex vertices in $\mathcal{P}$ is also in $O(n)$. The verification in Line 5 can be done in $O(n)$ time. The computation of the set $\mathcal{V}_E$ can be done in $O(n \lg n)$ in the worst case, using the same idea as in the proximity heuristic (Section 2.1). The search performed in Line 8 takes $O(n)$ time. The operation to change the position of p_i can be done in constant time, by maintaining pointers to the positions of $p_{i-1}, p_i, p_{i+1}, q^*$ and r^*. Therefore, the overall complexity of the loop from Lines 4 to 11 is $O(n^2 \lg n)$ in the worst case. Considering the main loop from Lines 2 to 11, we only perform a triangle swap when such operation yields an improvement for the objective function. There are $O(n^6)$ triangle pairs and each one can contribute in at most one iteration, since the reverse operation would result in worsening the total area. Moreover, because we only consider those triangles that are empty and that do not have points in their boundaries other than their vertices, a triangle pair would not appear more than once as a candidate. Therefore, the complexity of Algorithm 2 is $O(n^8 \lg n)$ in the worst case. In the rest of the article, we refer to this local search procedure as G-LS.

There are at least two interesting questions to be posed regarding G-LS. First, is there an upper bound on the number of iterations of the loop from Lines 2 to 11 asymptotically tighter than $O(n^6)$? Second, is it possible to transform any polygonization to any other using the local moves described in Algorithm 2? If so, we can reach an optimal solution by applying these local transformations, given enough time.

For the latter, we have a partial answer for points not in general position. Consider the case depicted in Figure 7, where the polygon contains the reflex vertices p_4, p_6, p_7, and p_{11}.

Vertices p_4, p_6, and p_{11} are not candidates for a local search move since triangles $\Delta_{p_3p_4p_5}$, $\Delta_{p_5p_6p_7}$ and $\Delta_{p_{10}p_{11}p_{12}}$ are not empty. As for vertex p_7, although the triangle $\Delta_{p_6p_7p_8}$ is empty, the triangles induced by p_7 and its fully visible edges are either degenerate ($\Delta_{p_7p_8p_9}$ and $\Delta_{p_7p_9p_{10}}$) or non empty ($\Delta_{p_7p_{10}p_{11}}$), since we are assuming that vertices do not block visibility. As a consequence, no further local movements are possible. Hence, there is no way to move from this solution to another one using the aforementioned local moves. Nevertheless, the question remains open for points in general position. Also, this example can be used to show that the neighborhood of a solution can be extended to include further movements of convex vertices. To see this, notice that the internal triangle $\Delta_{p_4p_5p_6}$ can be replaced by the external triangle $\Delta_{p_{11}p_5p_{12}}$, which results in a simple polygon with smaller area. This more general idea is explored in [10]. However, this broader notion of neighborhood was not used by us and is worth future investigation.

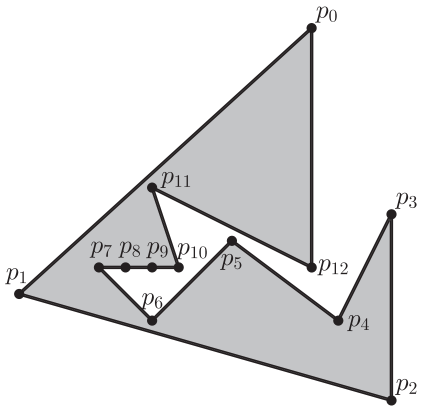


Fig. 7. A polygonization where no local movement can be applied.

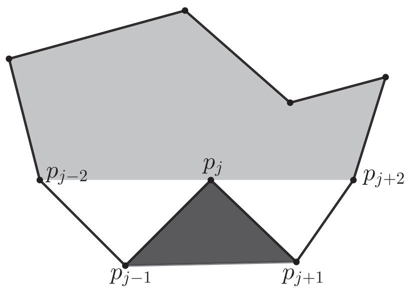
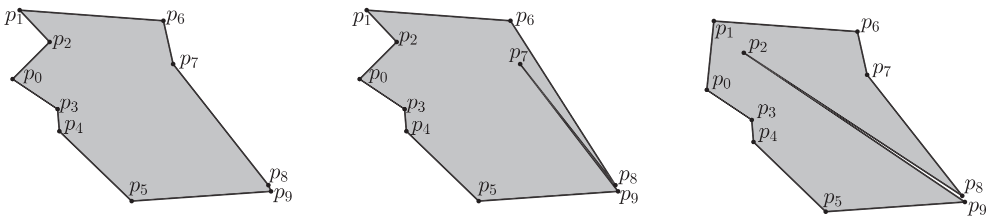


Fig. 8. Example of situation in which the simple local search can perform an exchange.

In practice, the time required to execute one iteration of the loop from Lines 2 to 11 in Algorithm 2 can be excessively large, specially in instances with a large number of points. For that, a simpler notion of local move was developed that reduces the complexity of an iteration to $o(n^2 \lg n)$, in the worst case. In this case, reflex vertices are still considered for finding candidate triangles for insertion, but only in the following, more restricted, situation. First, a reflex vertex (refer to p_j in Figure 8) must be a neighbor of two convex vertices and the triangle formed by it and its neighbors must be empty (as is the case for triangle $\Delta_{p_{j-1}p_jp_{j+1}}$ in dark gray). Under these two conditions, this triangle is a candidate for insertion. Next, we inspect triangles $\Delta_{p_{j-2}p_{j-1}p_j}$ and $\Delta_{p_jp_{j+1}p_{j+2}}$(in white) as candidates to be removed. Note that, to maintain simplicity of the polygonization, the triangle to be removed must be empty. Since we want to improve the area of the current polygon, the triangle must also have area smaller than $\mathcal{A}(\Delta_{p_{j-1}p_jp_{j+1}})$. Therefore, if both triangles are empty, the one with the smallest area is picked. Following our example, consider that $\Delta_{p_{j-2}p_{j-1}p_j}$ is chosen, so the exchange with $\Delta_{p_{j-1}p_jp_{j+1}}$ is performed. This corresponds to replacing the subsequence $p_{j-2}p_{j-1}p_jp_{j+1}$ of the current polygon by the subsequence $p_{j-2}p_jp_{j-1}p_{j+1}$.

As for G-LS, this procedure can be adapted to Min-Area. During the inspection of a reflex vertex, when looking for the triangle to be removed among the ones formed with its convex neighbors, we now have to select the one with the largest area—assuming that both are empty—and it must also have area larger than $\mathcal{A}(\Delta_{p_{j-1}p_jp_{j+1}})$.

For each reflex vertex in the aforementioned situation, $O(n)$ operations suffice to check whether the three triangles are empty, and it is possible to perform the swap, when it is beneficial, in



(a) Initial solution. (b) Solution reached by S-LS. (c) Solution reached by G-LS.

Fig. 9. Max-Area: example where both local search methods start at the same solution and S-LS achieves a better solution than G-LS. The instance is `uniform-000010-2`, the initial solution has area 140 296, the one by S-LS has area 147 176 and G-LS has area 143 676.

constant time. Therefore, this set of operations has complexity $O(n^2)$. In the rest of the article, we refer to this local search procedure as S-LS.

Note that the possible local movements by S-LS are a subset of the ones realizable by G-LS since, for a given reflex vertex p_j, both edges $\overline{p_{j-1}p_{j-2}}$ and $\overline{p_{j+2}p_{j+1}}$ are in $\mathcal{V}_E$. However, in G-LS we always choose the movement that yields the best improvement for each reflex vertex, hence it may prevent further swaps that S-LS would do, as shown in Figure 9. Figure 9(a) shows the initial polygon, with an area 140 296, given to both local searches, for the instance `uniform-000010-2` from the contest's benchmark. In both methods, we start at vertex p_6 and traverse the polygon in counterclockwise order, considering Max-Area. S-LS inserts the triangle $\Delta_{p_8p_7p_6}$ with area 7 570 and remove $\Delta_{p_9p_8p_7}$ with area 690. No further exchanges can improve the solution and the final one has area 147 176. G-LS inserts triangle $\Delta_{p_1p_2p_0}$ with area 5 880 and removes $\Delta_{p_9p_2p_8}$ with area 2 500. Notice that the swap applied by S-LS is no longer viable. G-LS stops at a local optimum, with area 143 676.

3 ALGORITHM ENGINEERING

The problem of computing the set $\mathcal{V}_E$ of the fully visible edges of a polygonal chain from a given *query point* occurs in both Algorithms 1 and 2, as previously mentioned. We now show how to do such computation in $o(n^2)$ time, using visibility polygons. Basically, given a simple polygon $\mathcal{P}$ and one of its vertices p, we compute the visibility polygon $\mathcal{V}(\mathcal{P}, p)$ and then, the set $\mathcal{V}_E$ comprised of the edges that are in both polygons. This idea can immediately be applied to Algorithm 2 where $\mathcal{P}$ is readily available and corresponds to the current polygonization. However, the situation is different in Algorithm 1 because the polygon required to use the previous technique is not the current polygonization and, therefore, has to be built at each iteration. This can be done as follows. Let p be the point to be inserted in the current polygonization $\mathcal{P}_i$ (Line 7). Assume that there is a vertical line r such that p lies in the closed half space on the right of r and $\mathcal{P}_i$ lies in the closed half space on the left of r. Let l^* (u^*) be the lowest (highest) vertex of $\mathcal{H}(V(\mathcal{P}_i))$ as seen from p. There are two polygonal chains in $\mathcal{P}_i$ going from l^* to u^*. Let $C_{l^*}^{u^*}$ be the one that contains the closest segment to p. Define $\mathcal{P}_i'$ as the simple polygon obtained by adding the segments $\overline{l^*p}$ and $\overline{u^*p}$ to $C_{l^*}^{u^*}$. Now, let $\mathcal{P} = \mathcal{P}_i$, we have the aforementioned setup needed to compute the set $\mathcal{V}_E$ using visibility polygons. Note that, in this case, we must filter out the edges $\overline{l^*p}$ and $\overline{pu^*}$ from $\mathcal{V}_E$ afterward. An illustration of the procedure to find $\mathcal{V}_E$, for the case of Algorithm 1, can be seen in Figure 10. The visibility polygon corresponds to the gray region and the fully visible edges of $\mathcal{P}_i$ from p are indicated by the dashed segments.

With respect to asymptotic complexity, the additional steps of Algorithm 1 required to compute $\mathcal{V}_E$ can be performed in $O(n)$ as follows. We maintain the convex hull of $V(\mathcal{P}_i)$ at each iteration

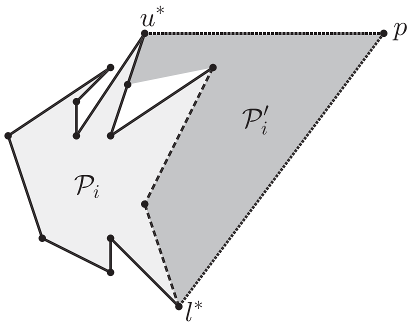


Fig. 10. Example of the setup in Algorithm 1 to compute $\mathcal{V}_E$ using a visibility polygon. Fully visible edges are indicated by the dashed segments.

Table 1. Statistics on the Speedup Obtained with the Visibility Polygon Based Algorithm to Compute $\mathcal{V}_E$

	Min	**Median**	**Mean**	**Std. Deviation**	**Max**
Overall	0.0563	1.5571	7.5571	11.9270	41.7027
At least 5 min.	9.4030	25.4230	25.5190	10.5534	41.7030

of Lines 5 to 12, updating it with each new point p. To do so, we find the above mentioned vertices l^* and u^*, remove the chain $C_{l^*}^{u^*}$ and add the segments $\overline{l^*p}$ and $\overline{u^*p}$. These operations and the construction of polygon $\mathcal{P}_i'$ can be done in $O(n)$ time. Moreover, the visibility polygon $\mathcal{V}(\mathcal{P}_i', p)$ can also be found in $O(n)$ time, using Joe and Simpson's algorithm [9].

Note that, using a cross-reference mechanism, one can identify the edges that are both in $\mathcal{P}$ and $\mathcal{V}(\mathcal{P}, p)$ in $O(n)$ time. However, since we opted to use CGAL's [15] implementation of the linear-time polygon visibility algorithm as a "black box", a postprocessing step to find such edges is needed. Such task can be done by storing the edges of $\mathcal{P}$ in a balanced search tree, attaining a complexity of $O(n \lg n)$ in the worst case [1]. Nevertheless, in our implementation, we used hash tables since they often offer better performance in practice.Moreover, we should emphasize that CGAL offers mechanisms to extend the DCEL data structure and allows for cross-referencing.

3.1 Computing Fully Visible Edges in Practice

In order to assess how the theoretical improvements in complexity discussed before (Section 3) affect the efficiency of our algorithms in practice, we consider the Max-Area problem, solved by the Prox-H-LL heuristic applied to 186 instances with $n \leq 9000$ points. See Section 5.2 for further details on the computational environment and instance set used. Here, we compare two approaches to compute $\mathcal{V}_E$ (Line 10, Algorithm 1): the naive $O(n^2)$ one and the one that uses visibility polygons. With respect to the latter, in order to find the edges that are in both $\mathcal{P}$ and $\mathcal{V}(\mathcal{P}, p)$, we use a hash table, as mentioned earlier.

Table 1 shows statistics about the speedup achieved with the visibility polygon based method. The first row corresponds to all 186 instances. In 103 instances, a speedup (greater than one) was achieved. The second row indicates statistics for the 48 instances, from these 103, that required at least five minutes of computing time, using the Prox-H-LL heuristic with the naive approach. Choosing to analyze such instances separately aims at reducing noise in running time measurements that may have been caused by operating system overhead. Note that higher speedups occurred for this subset of the benchmark with the largest one attaining a ~42-fold improvement. As

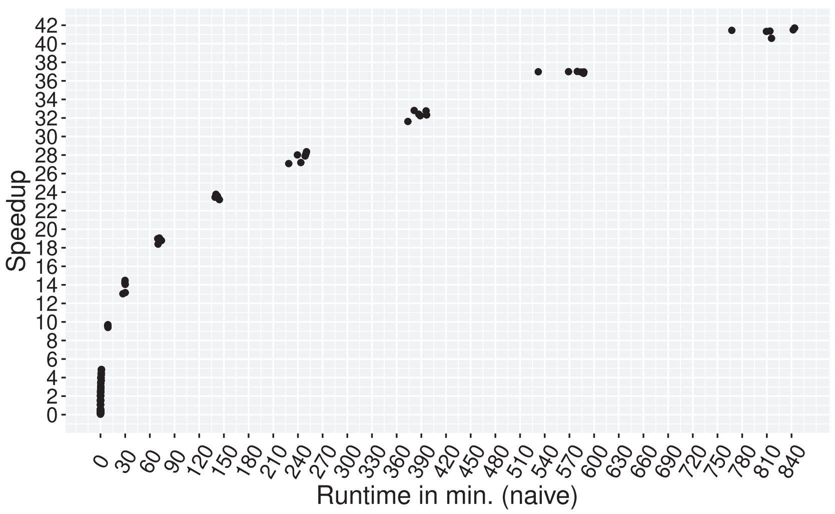


Fig. 11. Speedup obtained with the $O(n)$ algorithm to compute $\mathcal{V}_E$.

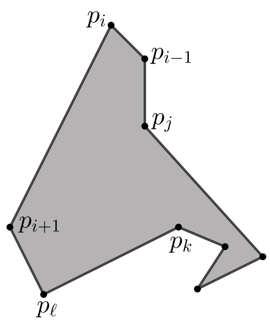
seen in the first row, a much larger variance in the speedup values was observed when the entire test set was considered, with the standard deviation exceeding the mean value.

Figure 11 depicts the speedup obtained (y-axis) as a function of the runtime of Prox-H-LL using the $O(n^2)$ approach (x-axis). One sees that the speedup grows as the runtime increases. This, together with the results of Table 1, is an indication of the improvement, both in theory and in practice, provided by the $O(n)$ approach for the computation of $\mathcal{V}_E$. Therefore, we adopted this algorithm in both Prox-H and G-LS. It is worth noticing that the linear time algorithm was only developed after the end of the contest, so the naive approach was used there.

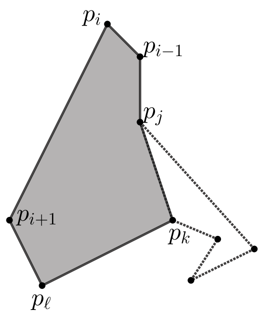
4 DEGENERACIES

In many of the algorithms described in Sections 2 and 3, we must deal with degeneracies that may occur in an instance, such as the presence of three or more collinear points. Here, we describe modifications to our methods that are intended to achieve robustness and comment on whether these changes affect their overall asymptotic complexity. In Line 3 of Algorithm 1, the three initial points — $\mathcal{S}[1], \mathcal{S}[2]$ and $\mathcal{S}[3]$ —can be collinear, implying that the triangle $\Delta_{\mathcal{S}[1]\mathcal{S}[2]\mathcal{S}[3]}$ is degenerate. When this situation arises, care must be taken to determine which are the segments fully visible from the next point in the sequence. The following strategy was adopted to cope with this issue. Let $\mathcal{S}_c$ be the longest prefix of the sequence $\mathcal{S}$ formed by collinear points and let $\overline{pq}$ be the shortest segment containing all the points of $\mathcal{S}_c$. Assume without loss of generality that p is lexicographically smaller than q and note also that $p, q \in \mathcal{S}_c$. Furthermore, let r be the next point in $\mathcal{S}$ after $\mathcal{S}_c$ and $\mathcal{P}$ be the polygon formed by the triangle Δ_{pqr}, which is non degenerate. Then, insert the points of $\mathcal{S}_c \setminus \{p, q\}$ in $\mathcal{P}$ in lexicographic order, between p and q. Let $i \leftarrow |\mathcal{P}| - 2, \mathcal{P}_i \leftarrow \mathcal{P}, j \leftarrow |\mathcal{P}| + 1$ and this replaces Lines 2 to 4. Since $\mathcal{P}_i$ is a simple polygon, Algorithm 1 can proceed normally. Note that this adaptation does not change the asymptotic complexity of the heuristic.

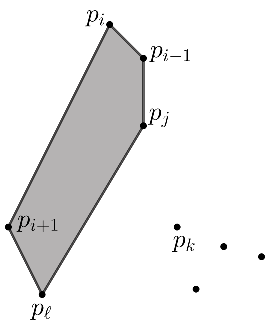
The approximation algorithm by Fekete and Pulleyblank [6] depends on the computation of the convex hull $\mathcal{H}(S)$ of the input point set S to obtain the everted polygon $\mathcal{P}^e$. Note that when the description of $\mathcal{H}(S)$ is minimal, a point $p \in S$ may lie on $\partial\mathcal{H}(S)$ and not be a vertex of $\mathcal{H}(S)$, when degeneracies are present. In this case, $\mathcal{P}^e$ will not be a simple polygon, since $p \in V(\mathcal{P}^s) \setminus V(\mathcal{H}(S))$. To handle this issue, we *expand* the description of $\mathcal{H}(S)$. Basically, for any point $p \in S$ contained



(a) Simple polygon $\mathcal{P}$. Consider computing the visibility polygon from p_i.



(b) The visibility polygon $\mathcal{V}(\mathcal{P}, p_i)$ is shown in gray. Points p_i, p_j and p_k are collinear. In this case, p_j blocks the vision from p_i to p_k



(c) The polygon obtained after the removal of the edges incident on p_k, since the vision from p_i to p_k is blocked by p_j.

Fig. 12. Example of visibility polygon with three non adjacent collinear points.

in the strict interior of an edge $\overline{qr}$ of $\mathcal{H}(S)$, p is inserted between q and r in $\mathcal{H}(S)$. This can be done during the construction of the convex hull, in the sweep line based algorithm, for instance, without affecting the optimal $O(n \lg n)$ complexity. Moreover, in the initial sorting of the points of S to obtain $\mathcal{P}^s$, if two points have the same angular coefficient with respect to p^*, ties are broken by increasing distance to it. However, if the respective angular coefficient is the largest one, ties are broken by decreasing distance [6]. Note that these adaptations do not change the asymptotic complexity of the algorithm.

Recall that, in Section 3, we show how to compute the set of fully visible edges of a simple polygon $\mathcal{P}$ from one of its vertices p, using visibility polygons. There, in the presence of degeneracies, situations that need special treatment may appear, as depicted in Figure 12. Shown in Figure 12(a) is a simple polygon $\mathcal{P}$ and a query point p_i. Note that the three points p_i, p_j and p_k are collinear. Figure 12(b) shows the visibility polygon $\mathcal{V}(\mathcal{P}, p_i)$. Since $\overline{p_ip_j}, \overline{p_jp_k} \subset \mathcal{P}$, by definition of a visibility polygon, p_j and p_k are visible from p_i. However, for both Min-Area and Max-Area, this situation is not desirable since the addition of either one of the triangles $\Delta_{p_ip_jp_k}$ or $\Delta_{p_ip_kp_\ell}$ would lead to a non simple polygon. To cope with this, the following strategy was adopted. First, sort the vertices of the chain $C_{p_{i+1}}^{p_{i-1}}$ in non-increasing angular order around p_i, with ties broken by increasing distance from p_i. Denote this sequence of points by $\mathcal{S}$. Let $\mathcal{S}_a$ be any consecutive subsequence of $\mathcal{S}$ formed by points that have the same angular coefficient with respect to p_i. Let

q be the first point in $\mathcal{S}_a$. Note that, for the purpose of this operation, we will consider that q blocks the vision from p_i to all other points in $\mathcal{S}_a$. Therefore, any segment of $\mathcal{V}(\mathcal{P}, p_i)$ that has as endpoint some point in $\{\mathcal{S}_a[m] : m = 2, \ldots, n'\}$, where $n' = |\mathcal{S}_a|$, is not fully visible from p_i and must be discarded from $\mathcal{V}(\mathcal{P}, p_i)$. In our example, the resulting polygon is the one depicted in Figure 12(c), where it can be seen that the edges $\overline{p_j p_k}$ and $\overline{p_k p_\ell}$ were removed from the polygon. This adaptation does not change the worst case asymptotic complexity of both Algorithms 1 and 2.

5 COMPUTATIONAL EXPERIMENTS

This section is dedicated to reporting and analyzing the results obtained by applying the algorithmic ideas previously discussed.

5.1 Computational Environment

The computational environment used in our experiments was the following: a machine with an Intel(R) Xeon(R) CPU E5-2603 v3 @ 1.60 GHz processor, 32 GB of RAM, and operating system Ubuntu 16.04.6 LTS. The implementation was done in C++, compiled with gcc version 5.4.0 20160609. The CGAL [15] library version 4.13 was used for geometric computations. In all test instances, the coordinates of the points are even integers, so that the area of any polygonization is integral [12]. Since the main operations of our geometric algorithms deal only with integer numbers, we were able to employ the *Cartesian Kernel* from CGAL and a floating point number representation with finite precision. However, for visibility computations, since we could have to handle points with rational coordinates, we once again used the *Cartesian Kernel*, but here with rational number representation and arbitrary precision. Each execution was performed using a single thread, i.e., no parallelism was employed. The instances used as benchmark were the ones made available on the official web page of the contest.[1]

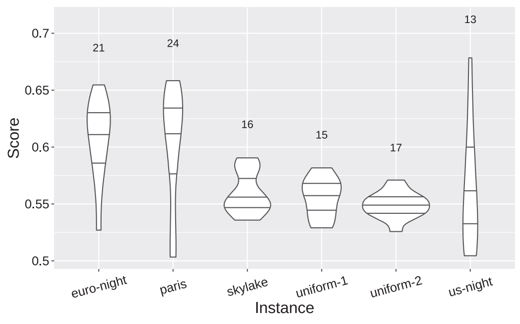
5.2 Results

In this section, we present and discuss the results obtained by our algorithms starting with the ones from the constructive heuristics (Section 5.2.1), followed by those from the local search procedures (Section 5.2.2). Next, in Section 3.1, we show how the improvements on the complexity of the algorithm to compute the set of fully visible edges (see Section 2) perform in practice. Lastly, in Section 5.2.3, we discuss the score-based results attained up to, as well as after, the closing of the contest. We employ non parametric statistical tests to compare different algorithms, with the confidence level parameter set to 95% [14].

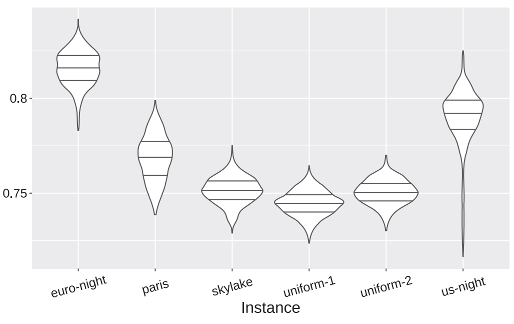
5.2.1 Constructive Heuristics. We first show how the choice of the *pivot* in Prox-H, APX-FP and GAPX-FP (see Section 2.1) can affect the quality of the solution produced. In this preliminary experiment, the six instances with $n = 1{,}000$ were used. In the case of Prox-H and GAPX-FP, all the points of the input set S were considered once as the pivot and for APX-FP, all the vertices of $\mathcal{H}(S)$ played that role.

The results are presented in Figure 13 (Max-Area) and Figure 14 (Min-Area), using *violin plots*. The y-axis corresponds to the score values and the x-axis to the name of the corresponding instance. Each plot extends from the minimum to the maximum values. The three horizontal segments strictly inside the plot, from bottom to top, indicate the first, second and third quartile, respectively. A rotated density plot is displayed symmetrically to the left and to the right of the vertical line at the respective value on the x-axis, giving information about the distribution of the data [8]. In the

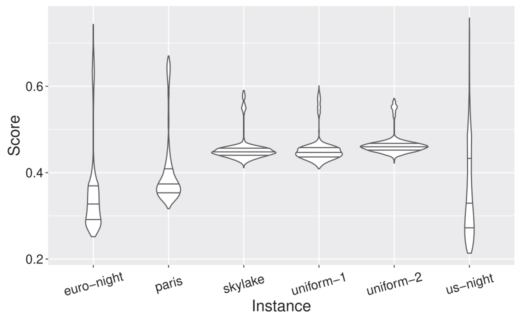
[1] https://cgshop.ibr.cs.tu-bs.de/competition/cg-shop-2019/.



(a) APX-FP



(b) Prox-H



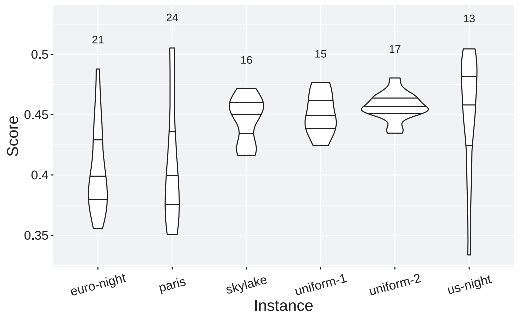
(c) GAPX-FP

Fig. 13. Results with multiple pivots, considering the Max-Area.

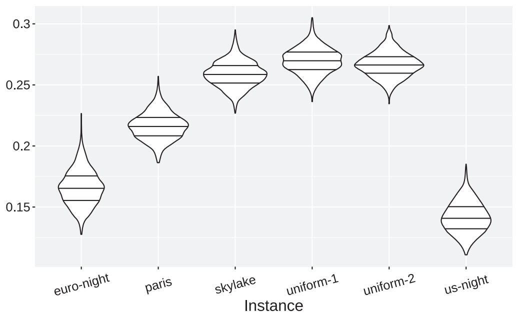
case of APX-FP (Figures 13(a) and 14(a)), the number of observations (e.g., $|V(\mathcal{H}(S))|$) per instance is shown above each plot.

Generally, the results obtained with the APX-FP method are distributed along with a wider range than with Prox-H and GAPX-FP. However, note that there is a smaller number of observations in this case, since only the vertices of $\mathcal{H}(S)$ are considered. With respect to GAPX-FP (Figures 13(c) and 14(c)) and Prox-H (Figures 13(b) and 14(b)), most of the samples are around the second quartile (median) for instances `skylake`, `uniform-1` and `uniform-2`, resulting in a higher density in that region. For the three remaining instances, the values are more spread out, specially for GAPX-FP. Note that in Figure 13(c), scores smaller than 0.5 are the cases where the everted polygon was not simple and the star-shaped one did not have area greater than or equal to half of the area of the convex hull. For all methods, it is possible to see that different choices of pivot can lead to solutions with rather different values. This can be observed, for example, from the distance between the three quartiles. Furthermore, in general, the plots have thin tails towards the best score values, meaning that just a few points lead to them. We should add that, from these tests, we were not able to identify structural features of the instances that could help to determine efficiently which point of S would lead to the best solution.

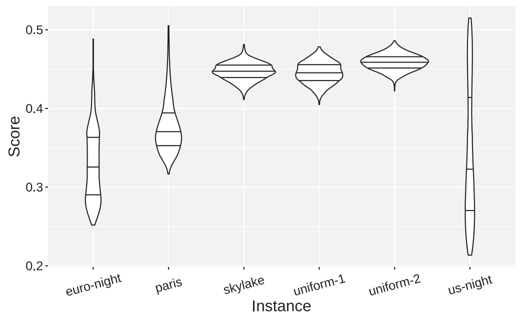
We now discuss and compare the constructive methods with different pivot selection strategies. In this experiment, we consider the 186 instances with $n \leq 9{,}000$ points ($\sim$75% of the total) as benchmark. As a matter of convenience, since this set is used in other experiments, we will refer to it as BASE-I. In Prox-H, two choices for pivots were employed: the lowest leftmost point, denoted by Prox-H-LL; the center (within $Int(\mathcal{H}(S))$) of the largest (smallest non degenerate) of the circles that circumscribe the triangles of the Delaunay Triangulation for the Max-Area (Min-Area),



(a) APX-FP



(b) Prox-H



(c) GAPX-FP

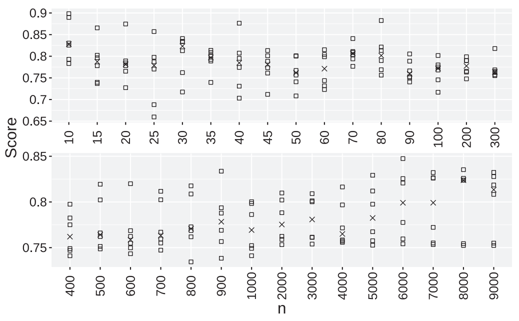
Fig. 14. Results with multiple pivots, considering the Min-Area.

denoted by Prox-H-BC. With respect to APX-FP, due to its low asymptotic complexity, all convex hull vertices were tested as pivots and the best solution among them was kept (denoted as APX-FP-A). Lastly, each point of the input set S was used as a starting point in GAPX-FP and the best solution among all was retained (denoted as GAPX-FP-A). To compare two different strategies, the Wilcoxon Signed-Ranks Test was employed in order to determine whether the difference between the results of the algorithms is statistically significant. That occurs when the result of the test, denoted as w, is smaller than -1.96, meaning that the null-hypothesis that both algorithms perform equally well is rejected [4]. If the methods differ by this test, we count the number of wins of each one of them, i.e., how many times each method was strictly better than the other. If the number of wins is at least $N/2 + 1.96\sqrt{N}/2$, then the algorithm is *significantly better* according to the z-test, where N is the number of observations (186 in this case). For more details about these tests, see Demsar [4].

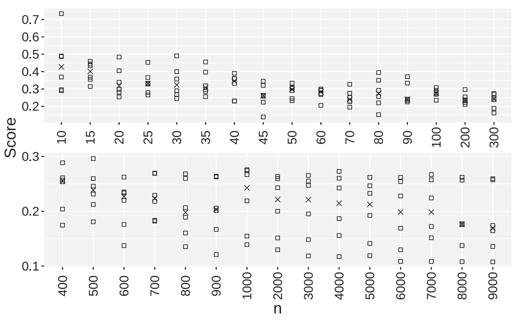
Table 2 compares the results obtained by APX-FP-A, GAPX-FP-A, and the two different pivot selection strategies for Prox-H. The first column indicates which problem is under consideration, followed by the pair of methods being compared and the value w of the Wilcoxon Test. Lastly, the **LHS** (**RHS**) column shows how many times (out of 186) the method on the left (right) of the corresponding comparison was strictly better than the other. The null-hypothesis was rejected in all comparisons and the algorithm with more wins was significantly better in all cases. Note that for both Min-Area and Max-Area, GAPX-FP-A outperformed APX-FP-A. Taking into account all executions of GAPX-FP-A, considering Max-Area, the star-shaped polygon had better area in $\sim 0.15\%$ and the everted one in $\sim 2.28\%$ of the cases. Moreover, the everted polygon was not simple in $\sim 97.55\%$ of the executions. However, note that, when it was simple, it had better area

Table 2. Comparison between the Constructive Heuristics

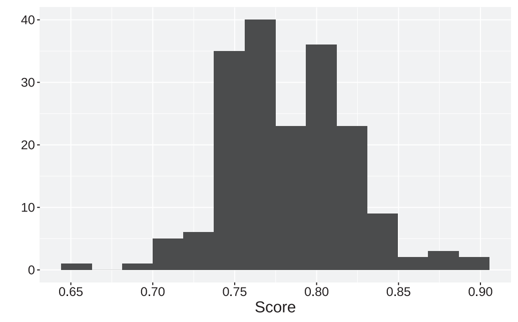
Problem	Comparison	w	LHS	RHS
Max-Area	Prox-H-BC × Prox-H-LL	−04.4067	124	60
	Prox-H-BC × APX-FP-A	−11.7357	182	4
	GAPX-FP-A × APX-FP-A	−11.8227	183	0
	Prox-H-BC × GAPX-FP-A	−10.5469	164	22
Min-Area	Prox-H-BC × Prox-H-LL	−03.0534	115	70
	Prox-H-BC × APX-FP-A	−11.5915	176	10
	GAPX-FP-A × APX-FP-A	−11.0597	159	15
	Prox-H-BC × GAPX-FP-A	−10.8021	166	20



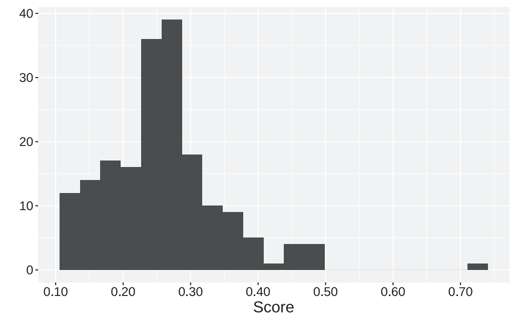
(a) Max-Area



(b) Min-Area



(c) Max-Area



(d) Min-Area

Fig. 15. Scores achieved with Prox-H-BC heuristic.

~93% of the time. So, for the purpose of providing good solutions on an instance basis, it is still an interesting approach. Note that, the results for Min-Area are complementary: APX-FP-A will have better solutions when they are worse for Max-Area. Analogously for GAPX-FP-A. With respect to the comparison Prox-H-BC × Prox-H-LL, in both problems, Prox-H-BC was the winner in more instances than Prox-H-LL. The same method was also superior to APX-FP-A and GAPX-FP-A in both. Therefore, we can conclude that Prox-H-BC was the best constructive heuristic for both Max-Area and Min-Area and we use it in the subsequent experiments. Note that this pivot selection strategy was developed after the closing of the contest, while Prox-H-LL had been used during the contest.

Figure 15 shows the final scores yielded by Prox-H-BC, where Figure 15(a) refers to Max-Area and Figure 15(b) to Min-Area. The x-axis corresponds to the instance sizes and the y-axis to the

Table 3. Max-Area: Performance Comparison between G-LS and S-LS

Instance	Score (G-LS)	Score Diff.	Area Diff.	Runtime (G-LS)	Speedup
world-7000	0.86	−0.02	−257950576980.00	305.04	221.43
uniform-8000-2	0.79	−0.03	−6250441578.00	372.60	203.71
jupiter-8000	0.86	−0.03	−229031453972.00	392.67	221.61
uniform-8000-1	0.79	−0.03	−6428842332.00	368.37	189.14
euro-night-8000	0.87	−0.03	−10039270.00	386.51	216.12
world-8000	0.86	−0.02	−273711406884.00	380.97	211.17
us-night-8000	0.88	−0.03	−17729100.00	388.03	203.86
world-9000	0.86	−0.02	−267861465558.00	487.13	213.00
jupiter-9000	0.84	−0.03	−227035430784.00	482.32	230.33
uniform-9000-2	0.79	−0.03	−7763360986.00	471.60	188.89
us-night-9000	0.88	−0.03	−18606614.00	487.96	201.53
uniform-9000-1	0.79	−0.03	−7801852278.00	453.66	183.03
euro-night-9000	0.85	−0.02	−8755088.00	466.37	215.60

score values. There are six observations per instance size (squares) and the median is indicated by an X mark. For each problem, the plots are separated by instances with $n \leq 300$ (top) and $n > 300$ (bottom). There are also two histograms, Figure 15(c) (Max-Area) and Figure 15(d) (Min-Area), indicating the overall distribution of the scores obtained. It can be observed that the heuristic was effective in both problems. For Max-Area, most of the scores are between 0.75 to 0.85 and, for Min-Area, between 0.2 to 0.35. In the case of Min-Area, instance `uniform-0000010` had an exceptionally high score of 0.7337. This happened because the initial greedy choices forced the heuristic to pick a triangle with an exceedingly large area at the end.

5.2.2 Local Search Procedures. We now present and discuss the results achieved with the local search procedures described in Section 2.2. First, we compare the two methods, G-LS and S-LS, by means of the Wilcoxon and z Tests. The set of instances BASE-I was used in this experiment and the initial solutions are those produced by the Prox-H-BC heuristic, as it was concluded to be the best one in Section 5.2.1. Both procedures were executed in an iterative fashion, that is, the solution obtained after one round of the local search is passed as input to it for a new round. This process is repeated until a local optimum is reached. The outcome of the Wilcoxon Test was $w = -11.8179$, meaning that there is a statistically significant difference between the techniques. G-LS won in 183 instances, while S-LS won just once (out of 186) and by the z-test, G-LS was significantly better than S-LS.

In what follows, we analyze how much faster S-LS was than G-LS, in contrast with the worsening of solution quality. This comparison relies on the data displayed in Tables 3 and 4. Here, we consider only instances where G-LS took at least 5 minutes to reach a local optimum, in order to avoid runtime discrepancies caused by measurement inaccuracies. The columns of each table show the instance name, the score obtained by G-LS, the absolute difference in score and area, runtime of the G-LS and the speedup achieved by S-LS; respectively. The score and area differences are read as "how much worse was S-LS than G-LS". We see high speedup values, specially for Min-Area, such as ~412 for the instance jupiter-8000 in Table 4. However, for both problems, whilst S-LS was faster, it produced solutions of poorer quality, as shown by the loss in score ranging from −0.2 to −0.5, which in some cases translates to billions of units in absolute area. Hence, as expected, although G-LS consumes more CPU time, it generally produces better results than S-LS.

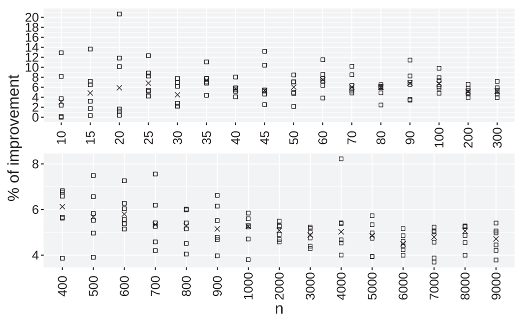
Next, we analyze the gain in solution quality when G-LS is applied in the initial solution provided by Prox-H-BC. Again, we used the BASE-I instance set. Figure 16 shows the percentage

Table 4. Min-Area: Performance Comparison between G-LS and S-LS

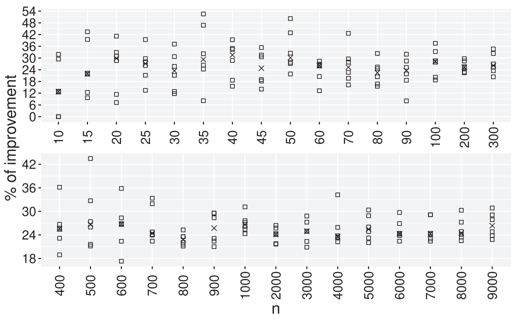
Instance	Score (G-LS)	Score Diff.	Area Diff.	Runtime (G-LS)	Speedup
skylake-6000	0.17	−0.04	−11943358.00	332.41	398.34
uniform-6000-1	0.20	−0.05	−6816040612.00	335.11	373.05
world-6000	0.13	−0.03	−382280186772.00	346.40	402.21
euro-night-6000	0.09	−0.03	−9487956.00	347.19	367.59
us-night-6000	0.08	−0.02	−13449460.00	343.42	370.57
uniform-6000-2	0.20	−0.05	−5977174776.00	338.33	374.02
skylake-7000	0.17	−0.04	−12245526.00	483.38	401.90
euro-night-7000	0.11	−0.04	−12802364.00	462.33	372.06
uniform-7000-1	0.21	−0.05	−8853304518.00	461.30	380.71
us-night-7000	0.08	−0.02	−12517814.00	464.96	372.37
uniform-7000-2	0.20	−0.05	−8388433840.00	485.92	400.33
world-7000	0.13	−0.03	−373154851344.00	474.01	404.23
uniform-8000-2	0.20	−0.05	−11264666058.00	622.09	369.42
world-8000	0.13	−0.03	−384577912036.00	632.17	362.06
uniform-8000-1	0.20	−0.05	−10777226798.00	629.98	404.13
euro-night-8000	0.10	−0.03	−9229178.00	609.90	353.93
jupiter-8000	0.13	−0.04	−315024333256.00	633.95	412.29
us-night-8000	0.08	−0.03	−15275214.00	618.50	370.13
world-9000	0.13	−0.03	−403597431366.00	787.48	387.97
uniform-9000-2	0.20	−0.05	−13655573390.00	795.00	399.10
jupiter-9000	0.11	−0.04	−308338096782.00	766.97	400.02
us-night-9000	0.08	−0.02	−13370972.00	756.83	383.40
uniform-9000-1	0.20	−0.05	−14486837320.00	737.11	381.66
euro-night-9000	0.10	−0.03	−11089480.00	723.68	359.40

of improvement attained applying G-LS, in both Max-Area (Figure 16(a)) and Min-Area (Figure 16(b)). The percentage of improvement is computed by the formula: $100 \times \frac{|\mathcal{P}^0 - \mathcal{P}^*|}{\mathcal{P}^0}$, where $\mathcal{P}^0$ is the solution cost obtained by the Prox-H-BC heuristic and $\mathcal{P}^*$ is the one produced by G-LS applied iteratively on $\mathcal{P}^0$. For each problem, the plots are separated by instances with $n \leq 300$ (top) and $n > 300$ (bottom). The x-axis corresponds to the size of the instances, while the y-axis is the relative improvement. There are six observations per size of instances (squares) and the median is indicated by an X mark. There are also two histograms, Figure 16(c) (Max-Area) and Figure 16(d) (Min-Area), indicating the overall distribution of the improvement achieved. It can be seen that, in most cases, the local search is more effective in Min-Area than in Max-Area. In one particular instance, namely `us-night-0000035`, there was an improvement of more than 50%, going from a score of 0.4556 down to 0.2162, after the local search. There are at least two possible explanations for this phenomenon. First, it could be that solutions returned by Prox-H-BC for Max-Area are generally better than for Min-Area and, consequently, there is not much room for improvement using G-LS. The second possible reason is that the number of reflex vertices in good solutions for Max-Area tends to be smaller than in the ones for the Min-Area, so the number of possible exchanges during G-LS is also smaller.

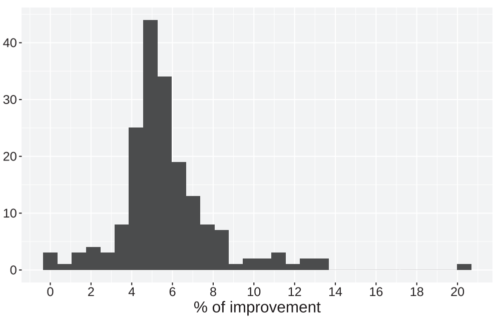
Furthermore, in the same experiment, we also measured how much time was spent in each of the two phases of our approach: the constructive heuristic (Prox-H-BC) and the local search procedure (G-LS). Here, we consider only instances where the two phases together took at least 5 minutes. This analysis is presented in Figure 17, for Max-Area (Figure 17(a)) and Min-Area



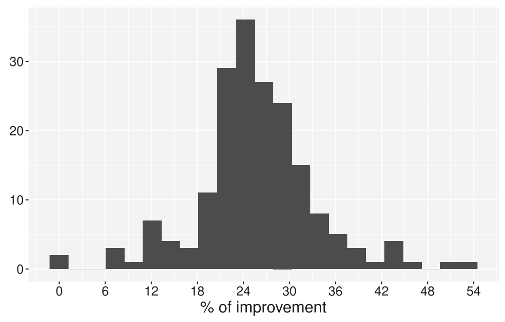
(a) Max-Area



(b) Min-Area

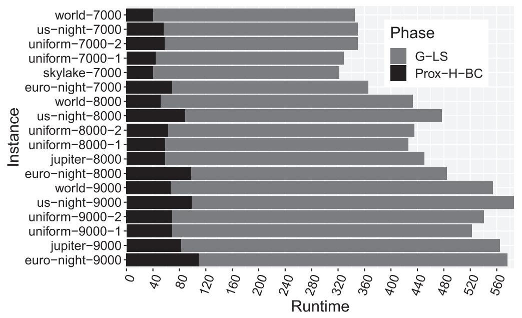


(c) Max-Area

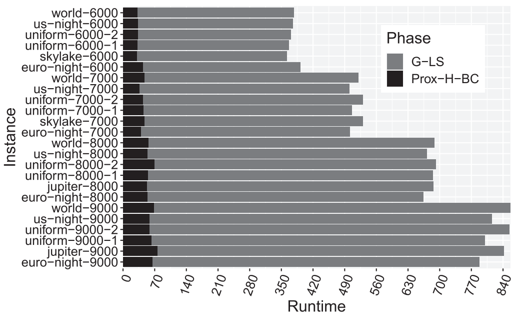


(d) Min-Area

Fig. 16. Percentage of improvement obtained applying G-LS.



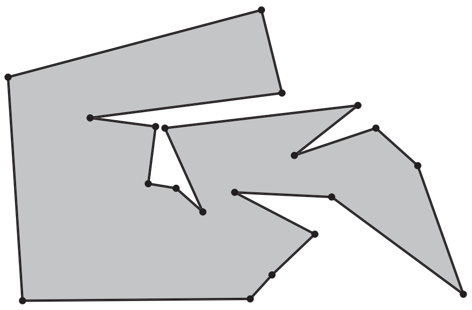
(a) Max-Area



(b) Min-Area

Fig. 17. Time spent (in seconds) by the constructive heuristic and the local search, considering only instances where it took at least 5 minutes to run both methods.

(Figure 17(b)). The y-axis contains the instances' names, while the x-axis indicates the runtime, in seconds. The dark bar represents the computing time used by Prox-H-BC and the gray one, G-LS. Generally, Prox-H-BC consumes a small portion of the time, when compared to G-LS. The maximum runtime observed was ∼2 minutes, for an instance with 9 000 points for Max-Area. Moreover, Prox-H-BC tends to be slower for Max-Area than for Min-Area. A possible reason for this is that, when solving Min-Area, it is preferable to add "thin" triangles with long edges at each iteration instead of "fat" ones, which blocks the visibility of many other edges in the next


(a) Max-Area

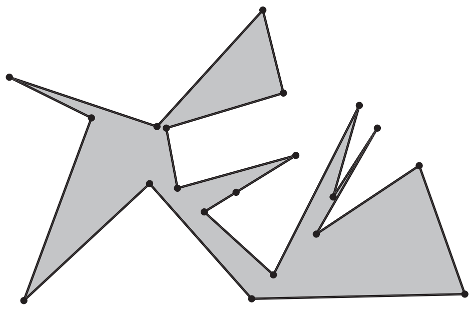
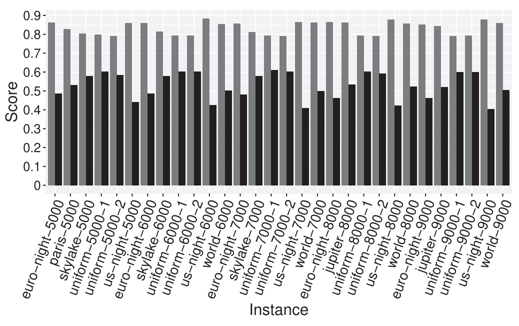
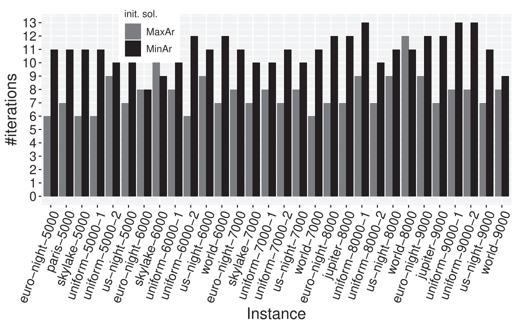

(b) Min-Area

Fig. 18. Solutions generated by Prox-H-BC for the instance `euro-night-0000020`.



(a) Comparison of scores.

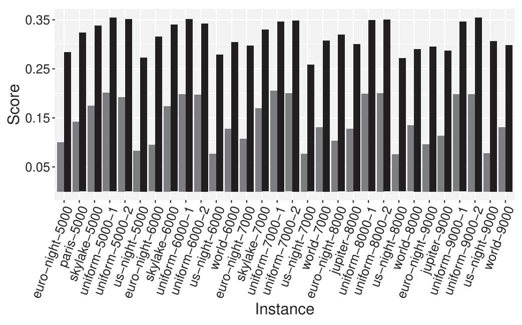


(b) Comparison of number of iterations.

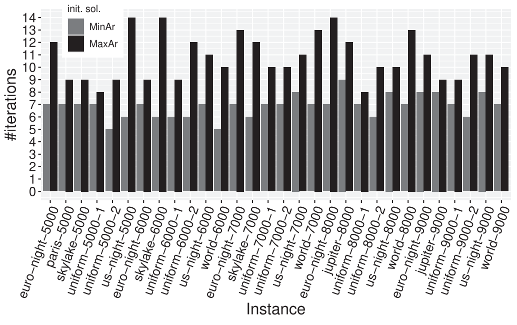
Fig. 19. G-LS – Max-Area: Results starting from poor (Min-Area) and good (Max-Area) solutions.

iteration. As a consequence, the size of the set $\mathcal{V}_E$ tends to be smaller at each iteration for Min-Area than for Max-Area. Lastly, G-LS usually consumes more time for Min-Area than for Max-Area. We see it as a consequence of a larger number of reflex vertices in solutions generated by the Prox-H-BC for Min-Area than for Max-Area, which leads to more opportunities to apply local moves in G-LS. In Figure 18, we can see the aforementioned differences between solutions generated for Max-Area and Min-Area by Prox-H-BC, in terms of presence of "thin" triangles and number of reflex vertices.

Additionally, an experiment was conducted to verify whether G-LS reaches equally good solutions, with no significant difference in running time, when starting from very different ones, especially, from a good and from a bad solution. This being true, we could conclude that, considering the respective set of instances and the proposed constructive heuristics, there is no need to invest time looking for a solution with good quality to feed the local search. In this case, the 30 largest instances of the set BASE-I were employed as benchmark. Considering Max-Area (Min-Area), G-LS was executed on the solutions computed by Prox-H-BC for Min-Area (Max-Area). Clearly, this is intended to be the case when the local search initiates at a poor solution. The results were then compared with the ones generated starting from good solutions, i.e., solutions to the same problem by Prox-H-BC. Figure 19 shows the results considering Max-Area, whereas Figure 19(a) refers to the score values and Figure 19(b) the number of iterations executed by G-LS, indicated on the y-axis. Here, *iteration* refers to a new execution of the local search, starting from



(a) Comparison of scores.



(b) Comparison of number of iterations.

Fig. 20. G-LS – Min-Area: Results starting from poor (Max-Area) and good (Min-Area) solutions.

the previous produced solution. Moreover, in both charts, for each instance in the x-axis, the gray (left) bar corresponds to the results obtained beginning on a good (Max-Area) solution, while the dark (right) one indicates the same commencing from a poor (Min-Area) solution. Analogously, Figure 20 depicts the results for Min-Area.

With respect to Max-Area, the Wilcoxon Test with the outcomes of G-LS starting at good and poor solutions showed that there is statistically significant difference between the two, with $w = -4.7821$. As can be seen in Figure 19(a), in all cases, the final score was higher when initiating at Max-Area solutions than at Min-Area ones and, by the z-test, the former was significantly better. Furthermore, a larger number of iterations was executed when G-LS commenced from poor solutions in all instances, except `skylake-0006000` and `world-0008000`. The results are analogous for Min-Area, except that, in this case, the number of iterations performed by the local search starting from good solutions was always strictly smaller than from poor ones. Hence, by these observations, we can conclude that the quality of the initial solution passed to G-LS has a direct impact on its outcome and effectiveness. Generally, starting from good solutions led to better results in fewer iterations than from poor quality solutions. Therefore, it is worth to invest effort in seeking high quality solutions to be passed to the local search, using Prox-H-BC, for example.

5.2.3 Results Obtained during and after the Contest. We now present the score-based results achieved during and after the contest, starting with a rough description of how the methods of Section 2 were used, chronologically. The first solutions were obtained with the constructive heuristics described in Section 2.1. The Prox-H and APX-FP were applied to instances with up to 100 000 points, while only APX-FP was used for all the larger ones. In case of the `mona-lisa-1000000` instance, in particular, all the convex hull vertices were tested as pivots in APX-FP and the best solution found was reported. The G-LS (see Section 2.2) local search procedure, was applied, in an iterative fashion, on the solutions of the instances with up to 10 000 points, while a single round was executed for the ones with 20,000 and 30,000 points. The simpler local search technique (S-LS) was applied iteratively to the remaining instances due to its lower asymptotic complexity, except for the `mona-lisa-1000000` instance, where just one round was executed. Recall that the improvements in complexity of the methods mentioned in Section 3.1 were not used during the contest. The process was analogous for both Min-Area and Max-Area.

The overall scores in the contest were measured by the sum of the individual scores across all instances. Our values were 46.432 (Min-Area) and 201.839 (Max-Area). Statistics about the scores achieved during the contest are shown in Tables 5 and 7, grouped by instance size. Analogously, Tables 6 and 8 display the respective scores obtained after the contest. These new results were

Table 5. Max-Area: Statistics about Scores Achieved during the Contest

n	min.	avg.	max.	n	min.	avg.	max.
10	0.8123	0.8788	0.9425	500	0.7527	0.7899	0.8386
15	0.8059	0.8431	0.9155	600	0.7818	0.8023	0.8584
20	0.8012	0.8685	0.9105	700	0.7773	0.8049	0.8501
25	0.7438	0.8219	0.8780	800	0.7876	0.8052	0.8269
30	0.8289	0.8616	0.9155	900	0.7792	0.8050	0.8440
35	0.7832	0.8370	0.8905	1000	0.7670	0.8047	0.8437
40	0.8078	0.8346	0.8943	2000	0.7891	0.8142	0.8496
45	0.7428	0.8117	0.9200	3000	0.7839	0.8133	0.8521
50	0.7440	0.8162	0.8797	4000	0.7906	0.8177	0.8638
60	0.7576	0.8122	0.8751	5000	0.7920	0.8234	0.8666
70	0.7978	0.8388	0.8866	6000	0.7967	0.8290	0.8615
80	0.7579	0.8081	0.8831	60000	0.7626	0.7950	0.8490
90	0.7763	0.8091	0.8326	70000	0.7658	0.8046	0.8585
100	0.7573	0.8116	0.8803	80000	0.7656	0.7996	0.8491
200	0.7914	0.8118	0.8237	90000	0.7647	0.8074	0.8606
300	0.7654	0.8047	0.8636	100000	0.7659	0.8013	0.8538
400	0.7819	0.7953	0.8171	1000000	0.5875	0.5875	0.5875

Table 6. Max-Area: Statistics about Scores Achieved during (d) and after (a) the Contest

n	min. (d)	avg. (d)	max. (d)	min. (a)	avg. (a)	max. (a)
7000	0.7909	0.8281	0.8719	0.7909	**0.8298**	0.8719
8000	0.7764	0.8250	0.8611	**0.7884**	**0.8324**	**0.8648**
9000	0.7790	0.8184	0.8420	**0.7907**	**0.8291**	**0.8525**
10000	0.7853	0.8181	0.8354	**0.7927**	**0.8259**	**0.8428**
20000	0.7872	0.8277	0.8683	**0.7957**	**0.8357**	**0.8768**
30000	0.7625	0.8234	0.8793	**0.7935**	**0.8402**	**0.8913**
40000	0.7654	0.8113	0.8563	**0.7961**	**0.8393**	**0.8829**
50000	0.7644	0.8050	0.8570	**0.7963**	**0.8339**	**0.8823**

achieved with G-LS being applied to instances with 20 000 to 50 000 points. Whenever the scores improved, they are highlighted in bold face. The new overall scores obtained after the contest were 46.411 (−0.021) and 202.482 (+0.643) for Min-Area and Max-Area, respectively.

6 CONCLUSIONS

In this article, we presented heuristic methods to solve both problems Max-Area and Min-Area. First, we provided two novel constructive heuristics: one based on iterative addition of triangles and proximity information about the input point set; and one that generalizes the idea of the approximation algorithm by Fekete and Pulleyblank [6]. We performed several experiments to assess the efficacy of these techniques and to verify how sensitive they are to the choice of an initial pivot point, a decision that must be made in both of them.

Next, we introduced two local search procedures intended to improve the solutions attained by the aforementioned constructive techniques. Both methods are based on the concept of exchanging triangles that are inside and outside of the current polygonization. Although the ideas behind these

Table 7. Min-Area: Statistics about Scores Achieved during the Contest

n	**min.**	**avg.**	**max.**	**n**	**min.**	**avg.**	**max.**
10	0.2511	0.3262	0.3850	600	0.1053	0.1551	0.1945
15	0.1511	0.3023	0.4572	700	0.1138	0.1608	0.2012
20	0.1861	0.2340	0.2901	800	0.1025	0.1628	0.2158
25	0.1535	0.2434	0.3211	900	0.0995	0.1619	0.2122
30	0.1634	0.2315	0.3714	1000	0.0935	0.1635	0.2182
35	0.2019	0.2398	0.3309	2000	0.0945	0.1601	0.2087
40	0.1906	0.2421	0.2845	3000	0.0887	0.1574	0.2096
45	0.2070	0.2447	0.3197	4000	0.0858	0.1561	0.2024
50	0.1687	0.2028	0.2448	5000	0.0811	0.1514	0.1972
60	0.1503	0.2072	0.2826	6000	0.0749	0.1483	0.2017
70	0.1758	0.2104	0.2428	40000	0.0883	0.1667	0.2443
80	0.1362	0.2019	0.2873	50000	0.0807	0.1761	0.2462
90	0.1739	0.2308	0.2850	60000	0.0809	0.1822	0.2467
100	0.1648	0.1956	0.2267	70000	0.0805	0.1735	0.2457
200	0.1469	0.1848	0.2023	80000	0.0785	0.1803	0.2459
300	0.1201	0.1871	0.2278	90000	0.0765	0.1706	0.2477
400	0.1429	0.1717	0.2051	100000	0.0772	0.1794	0.2471
500	0.1112	0.1733	0.2173	1000000	0.3723	0.3723	0.3723

Table 8. Min-Area: Statistics about Scores Achieved during (d) and after (a) the Contest

n	**min. (d)**	**avg. (d)**	**max. (d)**	**min. (a)**	**avg. (a)**	**max. (a)**
7000	0.0774	0.1491	0.2014	**0.0741**	**0.1477**	0.2014
8000	0.0826	0.1464	0.2105	**0.0772**	**0.1407**	**0.2026**
9000	0.0764	0.1426	0.2082	**0.0734**	**0.1380**	**0.2029**
10000	0.0794	0.1415	0.2050	**0.0741**	**0.1360**	**0.1980**
20000	0.0722	0.1449	0.2080	**0.0673**	**0.1392**	**0.2007**
30000	0.0812	0.1543	0.2329	**0.0655**	**0.1361**	**0.1996**

procedures are shared by other works in this volume [2, 5, 7, 10], our approach differs in the movements that are chosen to be applied and in what order. For instance, some works, namely [7, 10], select a random swap to be performed, while we take the one that gives the best improvement in total area for the resulting polygon. Moreover, we employ visibility polygons to speed up the computation of the fully visible edges of a polygon by one of its vertices. Lastly, we conducted numerous experiments to analyze the increase in solution quality achieved by these approaches and how their performance and efficacy are affected by the structure of the initial solution. Regarding the improvement in solution quality, the initial solutions were enhanced by ~5.70% and ~25.63% on average, for Max-Area and Min-Area, respectively. With respect to the second point, we observed that it is indeed worth to invest effort into finding a good initial solution since a poorer starting one leads to worst solutions and more iterations of the local search.

REFERENCES

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. *Introduction to Algorithms* (3rd ed.). The MIT Press.

[2] Loïc Crombez, Guilherme D. da Fonseca, and Yan Gerard. 2021. Greedy and local search solutions to the minimum and maximum area. *Journal of Experimental Algorithmics* (2021).

[3] Erik D. Demaine, Sándor P. Fekete, Phillip Keldenich, Domink Krupke, and Joseph S. B. Mitchell. 2021. Area-optimal simple polygonalizations: The CG challenge 2019. *Journal of Experimental Algorithmics* (2021).

[4] Janez Demsar. 2006. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research* 7, 1 (2006), 1–30. http://jmlr.org/papers/v7/demsar06a.html.

[5] Günther Eder, Martin Held, Steinþór Jasonarson, Philipp Mayer, and Peter Palfrader. 2021. 2-Opt moves and flips for area-optimal polygonalizations. *Journal of Experimental Algorithmics* (2021).

[6] Sándor P. Fekete and William R. Pulleyblank. 1993. Area optimization of simple polygons. In *Proceedings of the 9th Annual Symposium on Computational Geometry*. Chee Yap (Ed.). ACM, 173–182.

[7] Nir Goren, Efi Fogel, and Dan Halperin. 2021. Area-optimal polygonization using simulated annealing. *Journal of Experimental Algorithmics* (2021).

[8] Jerry L. Hintze and Ray D. Nelson. 1998. Violin plots: A box plot-density trace synergism. *The American Statistician* 52, 2 (1998), 181–184.

[9] Barry Joe and Richard B. Simpson. 1987. Corrections to Lee's visibility polygon algorithm. *BIT Numerical Mathematics* 27, 4 (1987), 458–473.

[10] Julien Lepagnot, Laurent Moalic, and Dominique Schmitt. 2021. Optimal area polygonization by triangulation and ray-tracing. *Journal of Experimental Algorithmics* (2021).

[11] Joseph O'Rourke. 1987. *Art gallery theorems and algorithms*, Vol. 57. Oxford University Press Oxford.

[12] Joseph O'Rourke. 1998. *Computational Geometry in C* (2nd ed.). Cambridge University Press.

[13] Franco P. Preparata and Michael I. Shamos. 1985. *Computational Geometry*. Springer.

[14] David J. Sheskin. 2007. *Handbook of Parametric and Nonparametric Statistical Procedures* (4th ed.). Chapman & Hall/CRC.

[15] The CGAL Project. 2018. *CGAL User and Reference Manual (v. 4.13)*. CGAL Editorial Board. Retrieved from https://doc.cgal.org/4.13/Manual/packages.html.

Received January 2021; revised July 2021; accepted November 2021