A Practical Iterative Algorithm for the Art Gallery Problem using Integer Linear Programming

Davi C. Tozoni · Pedro J. de Rezende · Cid C. de Souza

October 31, 2013

Abstract In the last few decades, the search for exact algorithms for known ℕℙ-hard geometric problems has intensified. Many of these solutions make use of Integer Linear Programming (ILP) modeling and rely on state of the art solvers, to be able to find optimal solutions for large instances in a matter of minutes. In this work, an ILP based algorithm is proposed to optimally solve the Art Gallery Problem (AGP), one of the most intensely studied problems in Computational Geometry. The basic idea of our method is to iteratively generate upper and lower bounds for the problem through the resolution of discretized versions of the AGP, which are reduced to instances of the Set Cover Problem. Our algorithm was implemented and tested on almost three thousand instances and attained optimal solutions for the vast majority of them, greatly increasing the set of instances for which exact solutions are known. To the best of our knowledge, in spite of the extensive study of the AGP in the last four decades, no other algorithm has shown the ability to solve the AGP as effectively and efficiently as the one described here. Evidence of its robustness is presented through tests done on a number of classes of polygons of various sizes with and without holes.

Keywords Art Gallery Problem · Exact Algorithm · Combinatorial Optimization · Computational Geometry

This research was supported by grants from Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) #477692/2012-5, #302804/2010-2, Fundação de Amparo à Pesquisa do Estado de São Paulo (Fapesp) #2007/52015-0, #2012/18384-7, and Faepex/Unicamp.

Institute of Computing, University of Campinas
Campinas, Brazil
www.ic.unicamp.br
davi.tozoni@gmail.com, {rezende | cid}@ic.unicamp.br

1 Introduction

In 1973, Victor Klee proposed a new geometric puzzle, which basically consisted in answering how few guards would be sufficient to ensure that an art gallery (represented by a simple polygon) be fully guarded, assuming that a guard's field of view covers 360 degrees as well as an unbounded distance. This question became known as the *Art Gallery Problem* (AGP) and, in the course of time, it turned into one of the most investigated problems in computational geometry. The breadth of this research can be perceived from the many important works that appeared, including O'Rourke's classical book [14], a recent text by Ghosh on visibility algorithms [10] and surveys by Shermer in 1992 [15] and Urrutia in 2000 [18].

In order to illustrate the problem, consider the Brazilian National Museum of natural history and anthropology, in Rio de Janeiro. Figure 1 (left)[1] shows the floor plan of its second floor. Suppose the Museum's curator intends to place cameras to guard this entire floor while subject to a limited budget. What would be the smallest number of cameras and their positions to completely cover this area? Figure 1 (right) shows an optimal solution that uses 25 cameras.

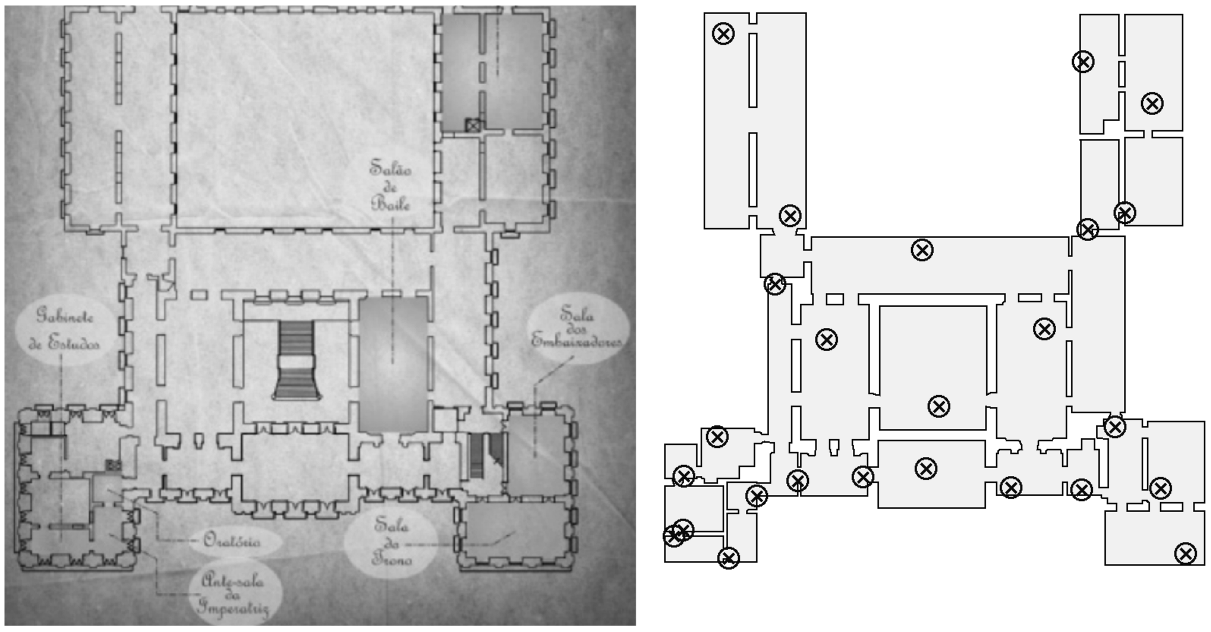

Fig. 1 National Museum floor plan (left); an optimal guard positioning (right).

Besides the natural application just described, the study and resolution of the AGP and its variations may also be of value in other areas. Consider, for instance, an application where one seeks to place the smallest number of nodes forming a sensor network that covers an area under the restriction that the viewing range is limited (in angle or in depth). Similarly, a robot equipped with a 3D scanner (of bounded range) may be used to map out a building floor, a plaza or an entire town. To minimize the number of scan positions is also a straightforward variation of the AGP (see [3]).

From these applications, it is easy to see that by varying the concept of visibility or changing what constitutes coverage or by pre-defining a discrete set of guard candidates, we are led to a number of variations of the AGP. In the *Art Gallery Problem With Fixed Guard Candidates* (AGPFC), in particular, a viable solution consists of a set guards that guarantee surveillance of the entire polygon while having their placement restricted to a finite number of positions in the polygon. In contrast, consider the *Art Gallery Problem With Witnesses* (AGPW) that asks for a set of guards able to observe merely a given finite set of points inside the polygon, while guard placement may be unrestricted. Both of these versions as well as many others are known to be $\mathbb{NP}$-hard [14].

In the past 10 years, advances have been reached in the development of algorithms for solving the AGP in practice. Among them, we find approximation algorithms [11], heuristics [1,5] and, more recently,

[1] Obtained from `www.museunacional.ufrj.br/visitacao/redescobrindo-a-casa-do-imperador/pavimento-2`.

some advances in the search for exact algorithms [8,12]. However, none of the algorithms proposed to solve the original AGP, where guards are free to be placed anywhere inside the polygon, was able to consistently find optimal solutions for large instances within a reasonable amount of time.

In this paper, we describe a new practical algorithm for solving the original AGP to exactness. It follows an iterative process of obtaining lower and upper bounds that lead to an optimal solution. These bounds are drawn up through the resolution of instances of the aforementioned AGPW and AGPFC problems using Integer Linear Programming (ILP) techniques. As we shall see in Section 5, optimal solutions are reached (in a reasonable amount of time) for almost the totality of the publicly available instances. These results consist in a significant improvement over the previously published techniques. Not only the proposed algorithm solved instances with up to 2500 vertices, something without precedent in the literature, but it also proved to be quite robust, achieving optimality in more than 98% of the 2400+ instances tested.

In the next section, we briefly survey relevant related works. Section 2 is dedicated to presenting a few basic concepts and notations while Section 4 describes the main steps of the proposed algorithm. A discussion of the computational results obtained in our experiments appear in Section 5. Lastly, in Section 6, we draw some conclusions and identify future directions of research.

2 Related Work

As early as 1975, Chvátal proved that $\lfloor n/3 \rfloor$ guards are always sufficient and sometimes necessary for covering a simple polygon of n vertices. Roughly a decade later, Lee and Lin proved the $\mathbb{NP}$-hardness of the AGP for vertex guards [13]. The case of point guards and of edge guards are also known to be equally hard [14].

After progress on theoretical results regarding the complexity of the AGP, researchers turned their attention to the development of algorithms for positioning guards and trying to reach solutions as close as possible to optimality. Ghosh, in 1987, proposed a $\log n$-approximation algorithm for the AGP with vertex or edge guards [9]. The idea, also exploited in more recent for exact algorithms, was to reduce the AGP to the *Set Cover Problem* (SCP) and to apply an already known approximation for the SCP.

In 2007, Amit et al. [1] proposed a set of heuristics based on greedy strategies and polygon partition methods. Four years later, Bottino and Laurentini [5] proposed a new heuristic for the restricted version of the AGP where the objective is to cover only the edges of the polygon. Both algorithms were a step forward and the latter, in particular, was able to find optimal solutions for a considerable number of instances.

Recently, however, the search for algorithms that are able to find provably optimal solutions for specific formulations has intensified. In this line, two results are worth highlighting. In 2011, Couto et al. [8], published the first exact algorithm for the Art Gallery Problem with Vertex-Guards (AGPVG) with verified practical efficiency. They devised an iterative technique that was guaranteed to converge to an optimal solution. While the maximum number of iterations was proven to be polynomial in the number of vertices in the worst case, actual convergence happened after just a few iterations. The first interesting insight was to show that the infinite points of the polygon to be watched could be discretized into a finite set of points, called *witnesses*. The AGPVG, having a finite number of possible guards, coupled with the now finite number of witnesses to be covered, was reduced (in the Karp sense) to a sequence of SCP instances that could be solved to optimality using an Integer Programming solver. These instances started off with a small subset of witnesses and simply included additional witnesses whenever the previous solution was not viable for the original instance. An alternative approach consisted of including all pre-determined witness into a single SCP instance and solving the problem to exactness in a single shot. In fact, in [8], a clever reduction of the witness set is also presented, which makes this alternative approach much more competitive timewise. These algorithms for the AGPVG were tested on more than ten thousand polygons of various classes and, for up to 2500 vertices, optimal solutions were attained in just a few minutes of computer time.

Through a related approach, Kröller et al. [12] tackled the original art gallery problem by seeking to reduce the infinite number of witnesses and guard candidates into finite sets, thus generating instances of the discretized AGP, which can be reduced to SCP instances as mentioned above. By solving the linear relaxation of the primal and dual formulations of the SCP instances created, their method is able to find upper and lower bounds for the original problem. The method is also iterative and adds new witnesses and guards at each new iteration, in hope that the an integer solution for the relaxed AGP is found. A negative result is inferred after a pre-established execution time limit is exceeded. Although their algorithm converged in several instances, giving rise to an increase in the number of instances for which optimal solutions are known, unfortunately, for many polygons the method was unable to converge to integer solutions.

This brief summary of the state of the art on exact solutions for the original AGP sets the stage for the contributions that this paper brings forth.

3 Terminology and Formulation

Recall that the objective of the Art Gallery Problem is to find the minimum number of guards that are sufficient to ensure visual coverage of a given gallery, represented as a simple polygon P (possibly with holes). Consider that the vertex set V of P has cardinality n.

Two points in P are *visible* to each other if the line segment that joins them does not intersect the exterior of P. The *visibility polygon* of a point $p \in P$, denoted by $\text{Vis}(p)$, is the set of all points in P that are visible from p. The edges of $\text{Vis}(p)$ are called *visibility edges* and are said to be *proper* for p if they are not contained in any edges of P.

Given a finite set S of points in P, a *covered* (respectively, *uncovered*) region induced by S in P is a maximal connected region in $\bigcup_{p\in S} \text{Vis}(p)$ (respectively, $P - \bigcup_{p\in S} \text{Vis}(p)$).

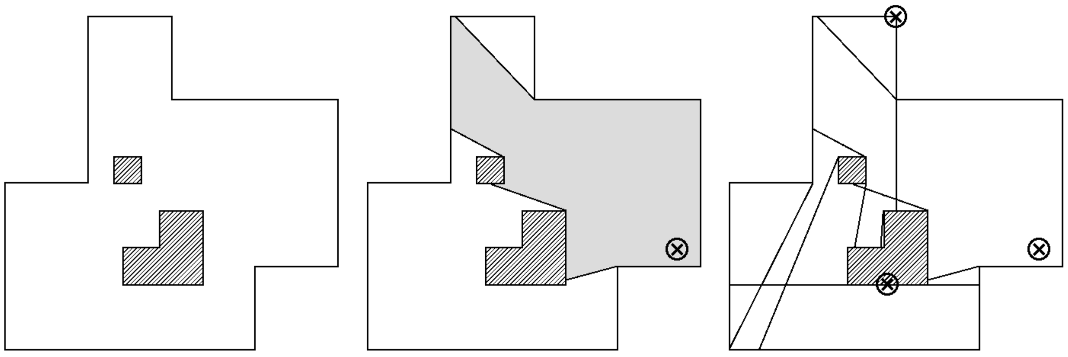

Fig. 2 A polygon with holes (left); the visibility polygon of a point (center); and the arrangement induced by a finite set S of points (right).

Moreover, the geometric arrangement defined by the visibility edges of the points in S, denoted $\text{Arr}(S)$, partitions P into a collection of closed polygonal faces called *Atomic Visibility Polygons* or simply *AVP*s. Denote by $C_f \subset S$ the set of points in S that cover an AVP f. Define a partial order $\prec$ on the faces of $\text{Arr}(S)$ as follows. Given two AVPs f and f', we say that $f \prec f'$ if and only if $C_{f'} \subset C_f$. We call f a *shadow face* (*light face*) if f is minimal (maximal) with respect to $\prec$. Figures 2 and 3 illustrate these concepts.

One important observation for the complexity analysis of our algorithm is that, for a given set $S \subseteq P$, the time needed to built $\text{Arr}(S)$, as well as the number of AVPs in this arrangement, is $O((|S| + |P|)^3)$ [4].

3.1 ILP Formulation

In a geometric setting, the AGP can be restated as the problem of determining a smallest set of points $G \subset P$ such that $\bigcup_{g\in G} \text{Vis}(g)$ equals P. This leads to a reduction from the AGP to the SCP, in which the points in P are the elements to be covered and the visibility polygons of the points in P are the sets used

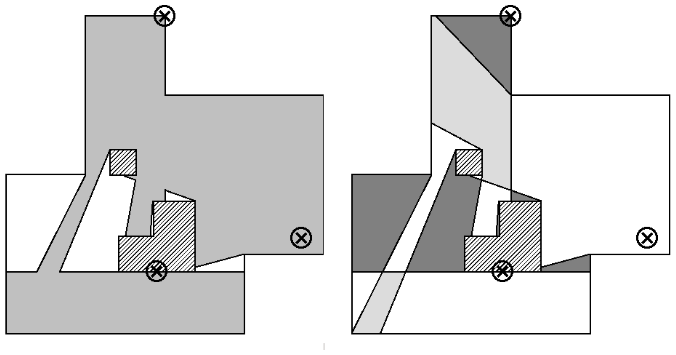

Fig. 3 A set S of three points and its covered (gray) and uncovered (white) regions (left); and the arrangement induced by S with the *light* (gray) and *shadow* (dark gray) AVPs (right).

for covering. Accordingly, we can use this reduction to construct an ILP formulation for the AGP:

$$\begin{aligned} \min \quad & \sum_{c \in P} x_c \\ \text{s.t.} \quad & \sum_{\substack{c \in P \\ w \in \text{Vis}(c)}} x_c \geq 1, \ \forall w \in P \\ & x_c \in \{0,1\}, \ \forall c \in P \end{aligned}$$

However, this formulation has an infinite number of constraints and an infinite number of variables, rendering it of no practical benefit. A natural idea that arises is to make at least one of these quantities finite. By fixing only the guard candidates to be a finite set, we obtain the so-called *Art Gallery Problem With Fixed Guard Candidates* (AGPFC). On the other hand, by restricting solely the witness set to a finite set, we end up with the special AGP variant known as the *Art Gallery Problem With Witnesses* (AGPW). In principle, in the first case, we are still left with an infinite number of constraints while, in the second case, we still have an infinite number of variables. However, in order to have a tractable SCP instance, one should have both the witness and the guard candidate sets of finite size. The AGP variant that fulfills this property is named the *Art Gallery Problem with Witnesses and Fixed Guard Candidates* (AGPWFC). Examples of these three versions of the AGP are shown in Figure 4. Therein, the witnesses and the guard candidates are identified by the symbols "$\times$" and "$\otimes$", respectively. Darker guard candidates refer to guards present in an optimal solution of the corresponding problem and, when appropriate, have their visibility polygons also depicted.

A remarkable theoretical result discussed in the next section is that any instance of the AGPW or of the AGPFC can be reduced in polynomial time to an instance of the AGPWFC. This means that, to solve the AGPW or the AGPFC, one actually has to find an optimal solution of a polynomial-sized SCP instance. Now, notice that the optimum of an AGPW instance always yields a lower bound for the original AGP value, since it is clearly a relaxation of the latter. On the other hand, if the set of guard candidates covers the polygon P, an optimal value for the AGPFC is feasible for the original AGP instance and, therefore, provides an upper bound for it. These are relevant observations since modern ILP solvers are capable to handle very large-sized SCP instances such as AGPWFC instances, for that matter. As we will see later, our algorithm relies on these facts.

To close this section, a couple of additional notations are introduced. Let D and C be finite witness and guard candidate sets, respectively. We denote the AGPW, AGPFC and AGPWFC problems defined for the sets C and D by AGPW(D), AGPFC(C) and AGPWFC(D, C), respectively. The SCP model associated to AGPWFC(D, C) is then

$$\begin{aligned} \min \quad & \sum_{c \in C} x_c, \\ \text{s.t.} \quad & \sum_{\substack{c \in C \\ w \in \text{Vis}(c)}} x_c \geq 1, \ \forall w \in D, \\ & x_c \in \{0,1\}, \ \forall c \in C. \end{aligned}$$

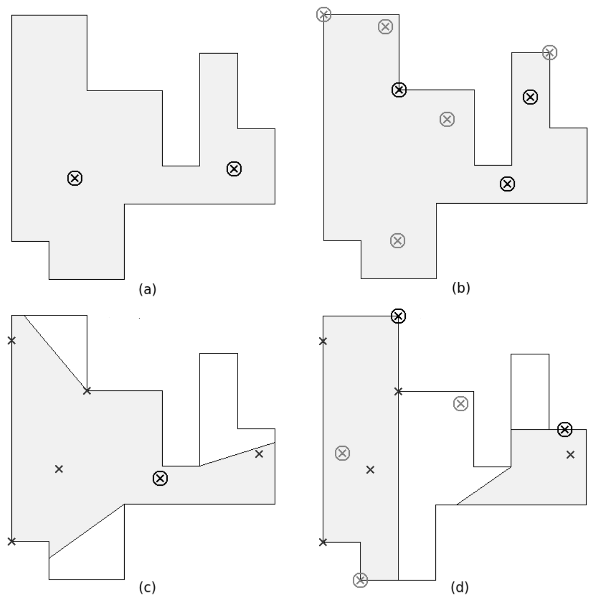

Fig. 4 An illustration of the four different variants of the Art Gallery Problem: (a) AGP; (b) AGPFC; (c) AGPW; (d) AGPWFC.

4 A Practical Iterative Algorithm for AGP

The purpose of this section is to describe our algorithm, which basically consists of iteratively obtaining increasing lower (dual) and decreasing upper (primal) bounds for the AGP. These steps are repeated until either the gap between the bounds reaches zero or the time limit is exceeded. New bounds are obtained by solving discretized versions of the original AGP. To find a new lower bound, an AGPW instance is solved, while in the upper bound case, an AGPFC instance is worked out in which the guard candidate set covers the polygon. One important fact to remember is that an optimal solution for such an instance of AGPFC is also viable for AGP, since the AGPFC asks for a solution that guards the entire polygon. Consequently, reducing the gap to zero means reaching an optimal solution for the original AGP.

In Algorithm 1, we present a simplification of how our technique works. After initializing the witness and guard candidate sets, the algorithm enters its main loop (lines 4 to 10). At each iteration new lower and upper bounds are computed and, if optimality has not been proved, the witness and guard candidate sets are updated in line 8. Later we will see how this can be done in a way that the duality gap decreases monotonically through the iterations. It is also worth noticing that, as we still do not have a proof of convergence for the algorithm, the parameter `MAXTIME` is used to limit its running time.

Algorithm 1 AGP Algorithm

```
1: Set UB $\leftarrow |V|$ and LB $\leftarrow 0$
2: Select the initial witness set $D$
3: Select the initial guard candidate set $C \supseteq V$
4: **while** (UB $\neq$ LB) and (MAXTIME not reached) **do**
5:    Solve AGPW($D$), set $G_w \leftarrow$ optimal solution of AGPW($D$) and LB $\leftarrow \max\{\text{LB}, |G_w|\}$
6:    Solve AGPFC($C$), set $G_f \leftarrow$ optimal solution of AGPFC($C$ and UB $\leftarrow \min\{\text{UB}, |G_f|\}$
7:    **if** (UB $\neq$ LB) **then**
8:       Update $D$ and $C$
9:    **end if**
10: **end while**
11: **return** $G_f$
```

In the following two subsections, the procedures for solving AGPW (line 5) and AGPFC (6) instances are described in detail. Subsequently, Section 4.3 describes the resolution method for AGPWFC instances through ILP techniques. As said in the previous section, both the AGPW and the AGPFC can be cast as an AGPWFC, justifying why we focus on the latter. In Section 4.4, we present how the management of witness set and, as a consequence, of the guard candidate set, is done. Lastly, Section 4.5 gathers all the algorithmic information presented previously and describes the complete algorithm for the AGP.

4.1 Solving AGPW

Consider AGPW(D) the Art Gallery Problem instance where the objective is to cover all points in a finite set D. As said before, the resolution of an AGPW on D allows for the discovery of a new lower bound for the AGP since fully covering P requires at least as many guards as the minimum sufficient to cover the points in D.

However, despite being a simplification of the original AGP problem, we are still dealing with an infinite number of potential guard positions, which does not allow for a direct reduction to the set cover problem. Thus, our approach is based on discretizing the guard candidate set, creating an AGPWFC instance from our original AGPW. To do this, we apply Theorem 1.

Theorem 1 *Let D be a finite subset of points in P. Then, there exists an optimal solution for AGPW(D) in which every guard belongs to a light AVP of* Arr(D).

Proof Let G be an optimal (cardinality-wise) set of guards that covers all points in D. Suppose there is a guard g in G whose containing face F in Arr(D) is not a light AVP. This means that F is not maximal with respect to $\prec$ (see Section 3). In other words, there exists a face F' of Arr(D) that shares an edge with F such that $F \prec F'$, i.e., F' sees more points of D than F does. An inductive argument suffices to show that this process eventually reaches a light AVP (maximal w.r.t. $\prec$) wherein lies a point that sees at least as much of D as g does, i.e., g may be replaced by a guard that lies on a light AVP. The Theorem then follows, by induction, on the number of guards of G. □

From Theorem 1, one concludes that there exists an optimal solution for AGPW(D) in which all the guards are in Light AVPs of the arrangement induced by D. Besides, as every point (whether inside or on the boundary) of an AVP can see the same set of witnesses, we can state that there is an optimal solution G where each point in G belongs to the set $V_{\mathcal{L}}(D)$ of all vertices from the light AVPs of Arr(D). Therefore, an optimal solution for AGPW(D) can be obtained simply by solving AGPWFC($D, V_{\mathcal{L}}(D)$). As seen before, the latter problem can be modeled as an ILP, where the numbers of constraints and of variables are polynomial in $|D|$. This follows, as mentioned in Section 3, from the fact that the number of AVPs (and vertices) in Arr(D) is bounded by a polynomial in $|D|$ and $|P|$. Algorithm 2 shows a pseudo-code of the AGPW resolution method.

Algorithm 2 AGPW(D) Algorithm

```
1: Arr($D$) ← construct the arrangement from the visibility polygons of the points in $D$
2: $V_{\mathcal{L}}(D)$ ← identify the vertices of the light AVPs of Arr($D$)
3: $C \leftarrow V_{\mathcal{L}}(D)$
4: Solve AGPWFC($D$, $C$): set $G_w$ ← optimal solution of AGPWFC($D$, $C$)
5: return $G_w$
```

4.2 Solving AGPFC

As we now know how to generate dual bounds for the AGP, the next task is to compute good upper bounds for the problem. A possible way to achieve this is through the resolution of an AGPFC instance in which the guard candidate set is known to cover the polygon. As said earlier, under this condition, an AGPFC solution is always viable for the original problem.

In this context, consider AGPFC(C) the instance of AGPFC where the finite guard candidate set is C. In contrast to what was discussed regarding the AGPW, we now have a finite number of guard candidates and an infinite number of points to be covered. Therefore, a straightforward resolution method using a reduction to an SCP is not possible. To circumvent this, our algorithm discretizes the original AGPFC instance, relying on what Theorem 2 establishes.

Theorem 2 *Let D and C be two finite subsets of P, so that C fully covers P. Assume that $G(D, C)$ is an optimal solution for AGPWFC(D, C). If $G(D, C)$ also covers P, then $G(D, C)$ is an optimal solution for AGPFC(C).*

Proof Assume that $G(D, C)$ covers P, but it is not an optimal solution for AGPFC(C). Then, there exists $G' \subseteq C$ with $|G'| < |G(D, C)|$ such that G' is a feasible solution for AGPFC(C), i.e., G' covers P. This implies that G' is also a feasible solution for AGPWFC(D, C), contradicting the fact that $G(D, C)$ is optimal for this problem. □

Theorem 2 shows that an optimal solution for AGPFC may be obtained through the resolution of an AGPWFC instance, provided it fully covers P. Whenever an optimal solution for the simplified version (AGPWFC) leaves uncovered regions in P, additional work is required. To guarantee that we attain an optimal solution for AGPFC, we will employ here a technique designed by Couto et al. [8] to solve the AGPVG, a special case of AGPFC where $C = V$, but which may be used to solve the general case without significant changes.

Initially, consider that we have a finite witness set D. Using the guard candidate set C, we can create and solve the AGPWFC(D,C) instance. If the solution fully covers polygon, we have satisfied the hypothesis of Theorem 2 and, consequently, we have an optimal solution for AGPFC. Otherwise, there are regions of the polygon that remain uncovered. We now update the witness set, adding new points within the uncovered regions, denoted $C_{\mathcal{U}}(C)$, and repeat the process.

As shown in [8], the iterative method for solving AGPFC converges in polynomial time. To clarify this point, consider the arrangement induced by all visibility polygons of guard candidates in C. This arrangement partitions the polygon into $O((|C| \cdot n)^2)$ faces (AVPs). These faces have the property that if any of its point is covered, the entire face is covered. Therefore, our aim is to iteratively construct D by choosing witnesses in the interior of AVPs of Arr(C), leading to an AGPWFC instance whose optimal solution will also be viable and optimal for the AGPFC. While the number of iterations is clearly in $O((|C| \cdot n)^2)$, in [8] it is shown that, in practice, this number is much lower. Moreover, it can even be argued that it suffices to select one point from each shadow AVP of the arrangement induced by C. See [8], for details.

A pseudo-code for the algorithm used to solve the AGPFC is shown in Algorithm 3.

Algorithm 3 AGPFC(C) Algorithm

```
1: $D_f \leftarrow$ initial witness set
2: **repeat**
3:     Solve AGPWFC$(D_f, C)$: set $G_f \leftarrow$ optimal solution
4:     **if** $G_f$ does not fully cover $P$ **then**
5:         $D_f \leftarrow D_f \cup C_{\mathcal{U}}(G_f)$
6:     **end if**
7: **until** $G_f$ fully covers $P$
8: **return** $G_f$
```

4.3 Solving AGPWFC (SCP)

Having reduced both AGPW and AGPFC problems into AGPWFC instances in order to obtain the desired bounds, the objective becomes solving the latter efficiently. Since an AGPWFC(D,C) instance can easily be reduced to an SCP, where the witnesses in D are the elements to be covered and the visibility polygons of the guard candidates in C are the subsets considered, we will make use of the ILP formulation for SCP presented in Section 3.

A simple and straightforward approach would be to directly use an ILP solver, such as XPRESS or CPLEX, since even large instances of the ($\mathbb{NP}$-hard) SCP can be solved quite efficiently by many modern integer programming solvers. However, as observed in our experiments, some AGP instances can generate significantly complex and very large SCP instances, rendering the solvers less efficient. For this reason, some known techniques were implemented to improve the solvers' running time. Among these, the most effective consisted in the reduction on the number of constraints and variables in the initial model. A simple primal heuristic based on Lagrangian relaxation was also implemented to obtain viable solutions. Despite the fact that these are standard techniques to ILP practitioners, some of them proved to be instrumental in the context of the AGP. So, for completeness, we briefly describe them below.

The method used for reducing constraints and variables is based on containment relationships between columns (guard candidates) and between rows (witnesses) of the Boolean constraint matrix corresponding to the ILP model of the SCP instance.

Firstly, we search for and discard *redundant guard candidates (columns)*. A guard candidate g_r is *redundant* if there is another candidate that covers the same witness set as g_r. For this step, we divide all guard candidates into groups, according to the light AVP they belong to. Guard candidates from the same AVP are then tested pairwise for redundancy, and, when one is found, it is removed from the original matrix. Here a slight improvement applies to SCP instances arising from the AGPW. Remember that when lower bounds are computed, the witness set D remains fixed while the guard candidates are the vertices of the light AVPs in $\text{Arr}(D)$. In this case, by definition, all vertices of a given light AVP cover the same subset of witnesses. Hence, all but one of the corresponding variables is redundant in the SCP instance, i.e., we are left with a single vertex from each light AVP. This reduces the size of the guard candidate set by a factor of at least three, although the remaining subset still needs to be checked pairwise for redundancy.

After the removal of columns, we also test each pair of rows in search for removable ones. We say that a row represents an *easy witness* w whenever the set of guard candidates that see w properly contains the (whole) set of candidates that see some other witness.

It is important to notice that the test for redundant candidates (columns) and easy witnesses (rows) can be performed simply by using very fast bit string operations.

Besides conducting matrix reduction, our algorithm also uses an initial *Lagrangian Heuristic* (LH) with the goal of finding good (or optimal) viable solutions for the AGPWFC. The heuristic implemented is based on the work by Beasley [2], which, among other results, presents a *Lagrangian Relaxation* for SCP. The idea of the relaxation is to move all coverage constraints into the objective function and use penalties u_w (for each constraint), the Lagrangian Multipliers (LM), resulting in the following *Lagrangian Primal Problem* (LPP):

$$\begin{aligned} z(u) = \min & \sum_{c \in C} x_c + \sum_{w \in D} u_w \left(1 - \sum_{\substack{c \in C \\ w \in \text{Vis}(c)}} x_c \right) \\ & x_c \in \{0,1\}, \forall c \in C \end{aligned}$$

After this step, the LPP (with no actual constraints) can be solved by inspection. It is well-known that the optimum of this problem gives a lower bound for the original SCP instance. The new quest is to find the values for LM (u_w variables) that maximize this lower bound. The optimization problem of finding the best Lagrangian Multipliers is called *Lagrangian Dual Problem* (LDP):

$$\begin{aligned} Z &= \max\ z(u) \\ u_w &\geq 0, \quad \forall w \in W. \end{aligned}$$

A classical approach to tackle the LDP is to apply the *subgradient method* (c.f., [2]), which is an algorithm in which the LM values are updated iteratively using a subgradient of the objective function. In this way, at each iteration, a new LPP instance is solved and, based on this result, we apply a primal heuristic, which attempts to find a new viable solution for the original SCP instance. The primal heuristic consists in a greedy algorithm that uses the Lagrangian costs obtained during the last LPP resolution to define which candidates will be selected to join the new viable solution (see [2] for more details). In summary, at each iteration, a new lower bound for the SCP is obtained from the LPP resolution and a new upper bound is obtained by the primal heuristic. The subgradient method stops when either an optimal solution is found or a maximum number of iterations has been reached.

After running the subgradient method for solving LDP, we have a reduced ILP matrix and an upper bound for AGPWFC. This information can be used in the ILP Solver to improve its performance.

Algorithm 4 summarizes the steps for solving the AGPWFC instance, using matrix reduction and a Lagrangian Heuristic.

Algorithm 4 AGPWFC(D,C) Algorithm

```
 1: $M \leftarrow$ Boolean matrix of the SCP model constructed using $D$ and $C$
 2: $R \leftarrow$ matrix obtained from $M$ after applying the reduction procedure
 3: Set $I \leftarrow$ SCP instance associated to $R$
 4: Solve $I$ using LH
 5: Set $\mathrm{LB}_{\mathrm{SCP}} \leftarrow$ best lower bound found for $I$ by the subgradient method
 6: Set $G_v \leftarrow$ best viable solution found for $I$ by the primal heuristic
 7: $G_s \leftarrow G_v$
 8: if $|G_s| \neq \mathrm{LB}_{\mathrm{SCP}}$ then    {LH was not able to prove optimality}
 9:     Solve $I$ using an ILP Solver with the primal bound $G_v$
10:     Set $G_s \leftarrow$ optimal solution found for the ILP model
11: end if
12: return $G_s$
```

4.4 Witness Management

The witnesses selected during the execution of our algorithm play an important role in the search for an optimal solution for the AGP. This occurs because the choice of the witness set D directly affects the quality of the lower and upper bounds and, consequently, the resolution of the AGPW (as well as the AGPFC) instances, as we will see later.

Recall that, before the first iteration of Algorithm 1, an initial witness set is chosen and in the following ones, it gets suitably updated (line 5). Several initialization alternatives were considered, tested and analyzed. We present below the three most effective of them.

The first initialization choice, called *All-Vertices* (AV), consists in using all vertices of the polygon as witnesses, i.e., $D = V$. Secondly, in an attempt to start off with a small number of witnesses, we also considered initializing D with only the convex vertices of the polygon, which is referred to as the *Convex-Vertices* (CV) initialization. The reason for trying to reduce the initial witness set lies on the fact that a smaller such set is likely to lead to a lower number of visibility polygon calculations. Moreover, the visibility arrangement used becomes less complex and so does the SCP model.

The third and last alternative is based on a work by Chwa et al. [7], where the authors define a polygon P to be *witnessable* when there exists a finite witness set $W \subset P$ with the property that any set of guards that covers W must also cover the entire polygon. That paper also presents an algorithm that computes a minimum witness set for a polygon whenever it is witnessable. The method for constructing this minimum witness set consists of placing a witness in the interior of every *reflex-reflex* edge of P and on the convex vertices of every *convex-reflex* edge. The terms *convex* and *reflex* here refer to the interior angles at a vertex or at the endpoints of an edge. Based on these selection criteria, we devised our third discretization method, called *Chwa-Points* (CP), which assembles the initial witness set for our algorithm from the midpoints of all reflex-reflex edges and all convex vertices from convex-reflex edges.

It follows from the results in [7] that, when the Chwa-Points discretization is used for a witnessable input polygon, our AGP algorithm will find an optimal solution in a single iteration. However, as observed in our experiments, non-witnessable polygons are the norm. In fact, among our random benchmark instances, they constitute the vast majority.

Let us now focus on the updating process of the witness set throughout the algorithm iterations. Of course, the better the choice a new set of points for inclusion into the witness set, the better the lower or the upper bound obtained would be. In essence, the process consists simply of adding points from the uncovered regions arising from the solution of the previous AGPW instance.

Our experiments showed that the inclusion of an interior point from each uncovered region was not sufficient to lead the algorithm to good performance and convergence. The shortfall was traced to the absence of new points on the boundary of the polygon, which proved to be useful to the evolution of the bounds obtained. Therefore, whenever an edge of an uncovered region is found to be contained on the boundary of the polygon, its vertices and its midpoint were also selected to increment the witness set throughout the iterations. These points along with an interior point of each uncovered region formed the whole set M of new witness.

4.5 Resulting Algorithm

Once each of the main steps of the algorithm is understood, we are able to describe how these parts fit together to comprise the complete algorithm. Algorithm 5 sums up how the process as a whole works.

Algorithm 5 AGP Algorithm

```
1:  $D \leftarrow$ initial witness set   {see Section 4.4}
2:  Set LB $\leftarrow 0$, UB $\leftarrow n$ and $G^* \leftarrow V$
3:  loop
4:    Solve AGPW($D$): set $G_w \leftarrow$ optimal solution and $z_w \leftarrow |G_w|$   {see Section 4.1}
5:    $C \leftarrow V_{\mathcal{L}}(D) \cup V$
6:    if $G_w$ is a coverage of $P$ then
7:      return $G_w$
8:    end if
9:    $U \leftarrow C_{\mathcal{U}}(G_w)$   {one additional point per uncovered region}
10:   LB $\leftarrow \max\{\text{LB}, z_w\}$
11:   if LB = UB then
12:     return $G^*$
13:   end if
14:   $D_f \leftarrow D \cup U$
15:   Solve AGPFC($C$), using $D_f$: set $G_f \leftarrow$ optimal solution and $z_f \leftarrow |G_f|$   {see Section 4.2}
16:   UB $\leftarrow \min\{\text{UB}, z_f\}$ and, if UB $= z_f$ then set $G^* \leftarrow G_f$
17:   if LB = UB then
18:     return $G_f$
19:   end if
20:   $D \leftarrow D \cup U \cup M$   {see Section 4.4}
21: end loop
```

It is important to notice that the set of guard candidates used in the AGPW resolution, which consists of V plus all vertices from the light AVPs induced by D, is actually the set C from the AGPFC(C) instance solved on Line 14. This means that the AGPW resolution is actually the first iteration of the AGPFC algorithm (Algorithm 3). Thus, all information obtained during the solution of AGPW(D) can be reused for AGPFC(C), which improves the performance of the implementation.

Another relevant aspect of this algorithm is that information on bounds may be used throughout the iterations in order to skip unnecessary steps. For instance, if an upper bound UB was found in a previous iteration and a new AGPFC instance is being solved, whose current solution is not lower than UB, then we may stop the AGPFC resolution before obtaining an optimal solution since the upper bound cannot be improved.

Figures 5, 6 and 7 illustrate the execution of the AGP algorithm on an orthogonal polygon.

5 Computational Results

To report how the experiments, conducted to validate our algorithm, were performed, we describe the computing environment, the instances used, the parameters employed and how they affect the overall performance of the resulting implementation. Lastly, a comparison with other existing techniques is presented, followed by a summary of the results obtained with all tested instances.

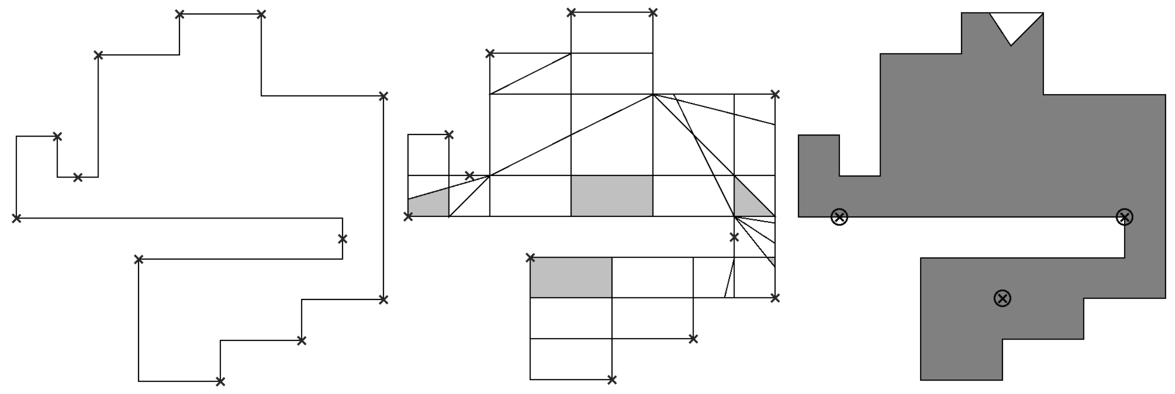

Fig. 5 Solving the AGPW (lower bound): The initial witness set D (Chwa-Points) (left); the arrangement $\mathrm{Arr}(D)$ and the light AVPs (center); the solution to $\mathrm{AGPW}(D)$ (right).

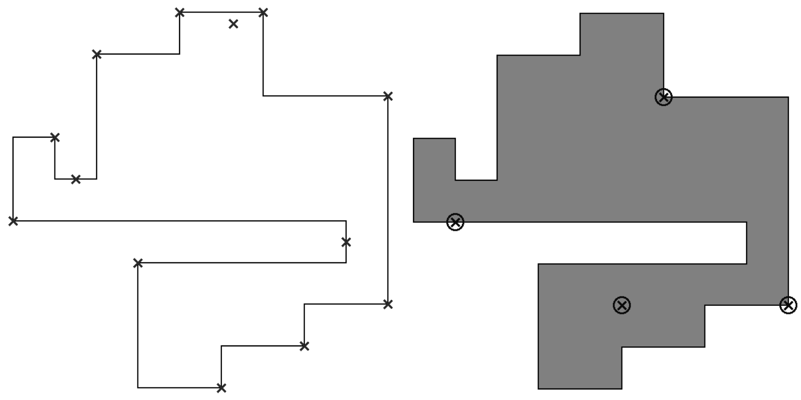

Fig. 6 Solving AGPFC (upper bound): Updated witness set D_f (left); the solution to $\mathrm{AGPWFC}(D_f, V_{\mathcal{L}}(D) \cup V)$ and to $\mathrm{AGPFC}(V_{\mathcal{L}}(D) \cup V)$ (right).

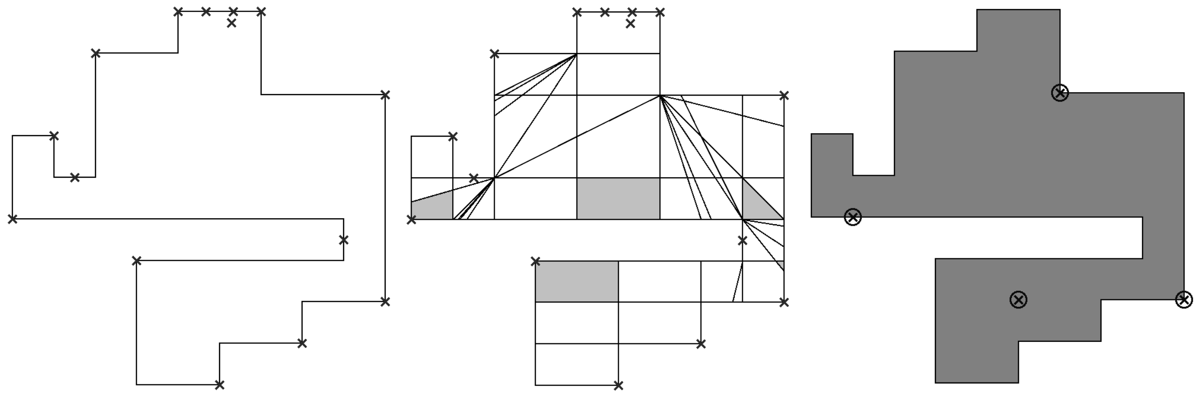

Fig. 7 Solving the AGPW (lower bound): The new witness set D (left); the arrangement $\mathrm{Arr}(D)$ and the light AVPs (center); the solution to $\mathrm{AGPW}(D)$ and to the AGP (right).

5.1 Computing Environment

All the experiments were performed on standard PCs featuring an Intel® Core™ i7-2600 at 3.40 GHz, 8 GB of RAM and running `GNU/Linux 3.2.0`.

Our algorithm (Algorithm 5) was coded in the `C++` programming language. The program uses the Computational Geometry Algorithms Library (CGAL) [6], version 3.9, to benefit from visibility operations, arrangement constructions and other geometric tasks. To solve the integer programs that model the SCP instances, we used the XPRESS Optimization Suite [20], version 7.0.

To achieve accurate time measurements, all tests were run in isolation, i.e., no other process was being executed concomitantly. Moreover, the experiments were run on a single core of the processor, hence, not profiting from parallelism. Each process was given a time limit of 60 minutes, after which, the instance was considered unsolved and the program was terminated.

5.2 Instances

In total, our benchmark included more than 2400 publicly available test polygons with and without holes. By collecting such an extended experimentation testbed, from various sources, comprised of polygons of multiple classes and sizes, we were able to stress the algorithm robustness.

For polygons without holes, three classes of polygons were used: simple, orthogonal and Von Koch polygons. The sets of simple polygons were obtained from the work by Bottino et al. [5] (250 polygons) and by Couto et al. [8] (600 polygons).

Random orthogonal polygons included 80 instances from [5] and another 600 instances from [8], while a third set of hole-free polygons contained 480 Von Koch polygons also from [8].

The instances were grouped according to their source and sizes: those from Bottino et al. contain between 30 and 60 vertices and are subdivided in groups of 20 polygons each. Those from Couto et al. range from 20 to 2500 vertices and each group has 30 instances of equal size.

In the case of polygons with holes, 120 instances called *spike polygons* ranging with 60, 100, 200 and 500 vertices were obtained from the work by Kröller et al. [12]. As we point out later, the availability of these instances afforded us the possibility of comparing our method with the technique presented in [12], which currently is among the most promising methods for solving the AGP.

On the other hand, to make for the lack of diversity of instances with holes in the available benchmarks, we generated another 300 random polygons with holes. These additional instances were made available in our web page [16] so that they can be used for further comparisons. To construct these new polygons with holes, two different generation techniques were used, each responsible for producing half of these new instances.

To understand the process of generating random simple polygons, picture a randomly generated set of points uniformly distributed inside a given rectangle. CGAL's `random_polygon_2` procedure generates a (not necessarily simple) polygon whose vertices are given by a random permutation of those points and applies a method of elimination of edge intersections based on *2-opt moves* (see [19]).

Now, for the first technique, denoted here *G1*, the process works as follows: after creating the outer boundary for a random simple polygon with holes, consider that we are left with v vertices to be distributed among h holes. After generating a random uniform partition of v into h parts, we iteratively generate the h holes in the following way. At each iteration, we randomly select a point in the interior of the polygon around which we center an isothetic square entirely contained inside the polygon. This square is then stretched in each of the four orthogonal directions, chosen in random order, by λD where λ is a stretch factor randomly picked from the interval $[0.25, 0.75]$ and D is the maximum elongation within the polygon in that direction. A hole, as a simple polygon, is then created, within this placeholder rectangle, having its number of vertices chosen from one of the unused parts of the previously mentioned random partition of v. Here, for an instance with a total of n vertices, $n/4$ of them were assigned to the outer boundary and $3n/4$ of them distributed among $n/10$ holes.

In contrast, in the second technique, denoted *G2*, the outer boundary of the polygon as well as all the holes are random orthogonal polygons. The maximal placeholder rectangle is constructed so that the chosen random interior point c is one of its vertices. Then, at each iteration, besides randomly selecting a new hole size, two holes are allowed to intersect, in which case, they merge into a single hole. Similarly to the previous generator, for an instance with a total of n vertices, the outer boundary has roughly $n/4$ of them, while the remainder are distributed among the resulting $n/10$ holes.

The range $[100, 500]$, with step size 100, was used for the number of vertices of the polygons with holes. A total of 30 polygons of each size per generator were produced using these methods, resulting in *G1* creating 150 instances (the *simple-simple* class) and an equal number produced by *G2* (the *ortho-ortho* class).

5.3 Development History

Since the present work expands and improves our own previous work [17], a comparison of results is due. Let us take, as an example, the polygon in Figure 8, which is a real instance representing a square in Bremen, Germany, with 14 holes and 311 vertices taken from [3]. More than 20000 seconds were required to obtain an optimal solution using version V1 of our implementation reported in [17].

Subsequent improved versions of the algorithm and its implementation, leading to version V2, included:

- upgrading the visibility polygon calculation by avoiding computation related to holes that would not benefit the result;
- cutting short the AGPFC procedure whenever the general upper bound found for the AGP had the same cardinality as the current AGPWFC solution, allowing it to skip to the next iteration of Algorithm 5.

As an illustration, V2 took less than 10000 seconds to obtain an optimal solution for the aforementioned Bremen instance.

Next, by reversing the point of view of visibility testing from the perspective of the guards to that of the witnesses (fewer in number, in our approach) we avoided computing many visibility polygons and reused those already required for setting up the arrangement. Moreover, by caching visibility information based on pairs of points already tested, the performance was further improved. Notice that the ILP matrix construction remained the same and changed little on how the algorithm works. This lead to version V3 which solved the Bremen instance five times faster than V2.

Recall from Section 4.3 that the ultimate version, V4, of the method described in this work includes the use of the Lagrangian heuristic for solving the SCP model, besides the removal of lines and columns from the original SCP matrix.

Moreover, V4 takes advantage of reusable information from previous iterations. For instance, if a lower bound LB for the AGP instance has been established, Algorithm 2 can be halted when solving a new AGPW(D) instance as soon as a primal solution with cardinality LB is found for the corresponding SCP instance in line 4 (for instance, using the Lagrangian Heuristic). This follows from the fact that LB was obtained by solving an AGPW(S) instance, for some $S \subset D$. Another interesting situation happens when we are solving an AGPWFC instance inside the iterative algorithm for AGPFC (Algorithm 3). Since, in this procedure, witnesses are added and never removed, we can guarantee that the solution of the previous AGPWFC instance is a lower bound for the next instance. Using V4, it took less than 600 seconds to reach an optimal solution for the Bremen instance!

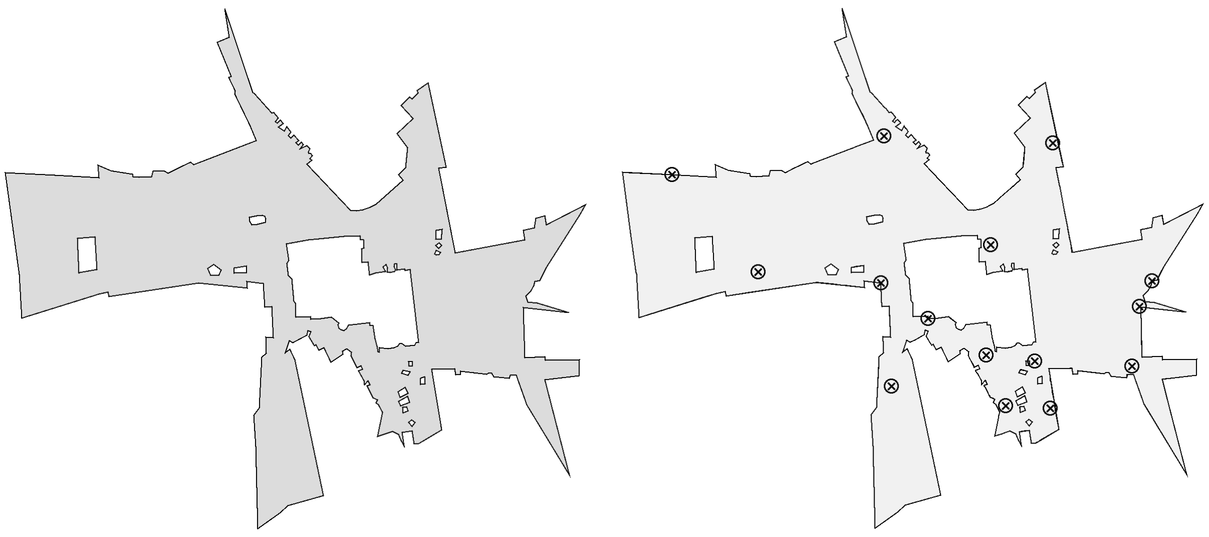

Fig. 8 Instance representing a square in Bremen, Germany (left); and an optimal guard positioning (right).

5.4 Discretization Techniques for the Witness Set

In Section 4.4, we presented three possible initial discretizations for the witness set. Natural questions that arise are: which of these leads to the highest number of exact solutions among multiple classes of instances of different sizes? With fewer iterations? With best time performance? Table 1 and Figure 9 display results obtained in this direction, which exemplify the behavior of the vast majority of the cases observed. In this summary, only ortho-ortho and simple-simple instances with sizes ranging from 100 to 500 vertices were displayed.

Instance Group	n	**# of Instances Solved**		
		All-Vertices	**Convex-Vertices**	**Chwa-Points**
Simple-simple (30 inst. per size)	100	30	30	30
	200	30	30	30
	300	29	30	30
	400	29	30	28
	500	26	27	28
Ortho-ortho (30 inst. per size)	100	30	30	30
	200	29	29	30
	300	29	29	30
	400	30	30	30
	500	27	27	28

Table 1 Number of instances solved to optimality when using each of the three initial discretization techniques.

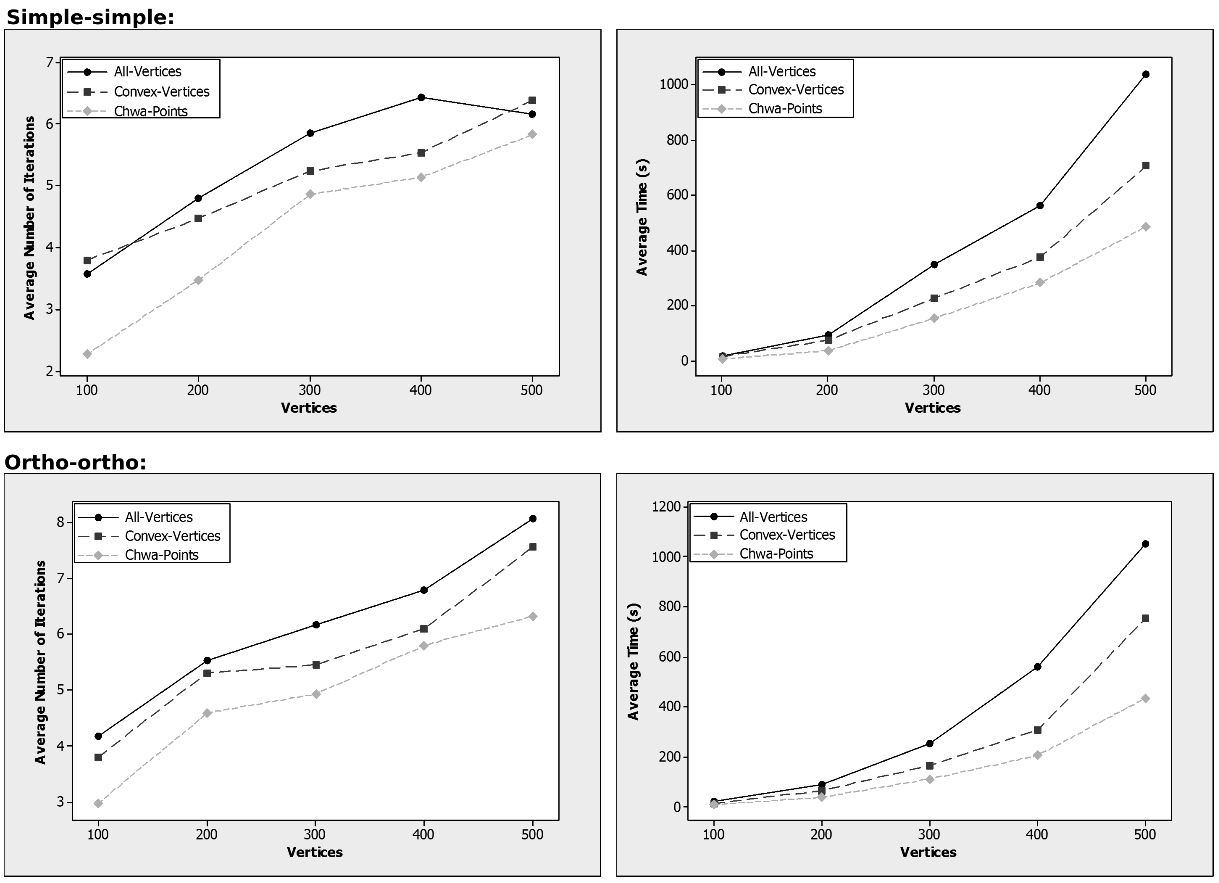

Fig. 9 Average number of iterations per size (left); and average time spent per size (right).

In Table 1 we have the number of instances that were solved to optimality using each discretization. As said before, the tests were halted after one hour of execution time. We can see that *Chwa-Points* and *Convex-Vertices* achieved the best results, solving 294 and 292 of the 300 instances considered, respectively.

By analyzing the charts in Figure 9, which presents the average number of iterations and the average time of successful runs, we see a reasonable advantage when using *Chwa-Points*. In the case of simple-simple polygons with 500 vertices, for example, we have a savings of more than 200 seconds when using Chwa-Points as opposed to Convex-Vertices. If we compare to the All-Vertices technique, this difference grows to more than 500 seconds, on average.

The graphs for All-Vertices and Convex-Vertices show that while they lead to very similar results in regard do the number of iterations, the latter has a considerable advantage over the former in run time and consequently in the number of instances solved to optimality (Table 1). This behavior stems from the fact that the discretization points in the Convex-Vertices strategy is a subset of the ones in the All-Vertices and clearly a larger witness set leads to more visibility polygon computations, more complex visibility arrangements and, consequently, larger SCP instances to be solved.

The Chwa-Points discretization technique, having been the most successful one, was chosen as the standard initial discretization for our method. In this context, throughout the following sections, all results presented were obtained using this initial discretization.

5.5 Lagrangian Heuristic

In Section 4.3, the Lagrangian heuristic was presented as a means to reach optimal solutions for AGP-WFC (SCP) instances faster. Lagrangian relaxation is reported (c.f., [2]) to perform well in many SCP applications and, therefore, we decided to test it on the SCP instances arising in the context of the AGP. In this section, we briefly analyze the pros and cons of using this technique in our implementation. For this comparison, four groups of instances, with sizes between 100 and 500, were considered: simple and orthogonal instances from Couto et al. [8] and the new simple-simple and ortho-ortho classes.

Table 2 shows results containing the number of instances solved when using (not using) the heuristic. Since optimal solutions were always found for all hole-free instances (with or without the heuristic), the table only show results for instances from the simple-simple and ortho-ortho classes. From these results, we can see that the number of instances solved within our timeframe of one hour did not vary considerably. Actually, in the case of the simple-simple group, the version without the heuristic was able to solve one additional instance.

So, at first glance, it does not seem to be advantageous to employ the Lagrangian heuristic. Nevertheless, the time used to compute viable solutions for SCPs using the heuristic is usually negligible with respect to the time spent in the other parts of our implementation. Besides, some instances benefited from using the heuristic and, in general, it yielded good primal bounds for SCP, which was its primary goal. Thus, in an attempt to understand why the optimality rate was not positively affected by the usage of the heuristic, we investigated this issue a little further. To this end, we looked at the solutions produced by both the heuristic and the ILP solver XPRESS for some instances. It turned out that, although they had equal sizes, these solutions were often not the same, resulting in different uncovered regions as well as different witness and guard candidate sets. As a consequence, the next sequence of SCPs to be solved changed considerably, deeply affecting the time required to find the optimum of the AGP. This observation shows that more research is needed to distinguish among the optimal SCP solutions those that are more promising for our algorithm, i.e., that allow it to solve the AGP quicker. For the time being, we turn our attention to the effects of the Lagrangian heuristic in the total running time.

Regarding the average run time when using the Lagrangian heuristic, the graphs in Figure 10 show a comparison for the case of each of the four groups tested. These graphs include only instances that were solved to optimality regardless of whether the Lagrangian heuristic was used.

In general, we perceive a modest advantage when using the Lagrangian heuristic in our procedure. Therefore, we decided to maintain the usage of the Lagrangian heuristic as the default option in all the tests reported in the following sections.

Instance Groups	n	Instances solved	
		Without LH	With LH
Simple-Simple (30 inst. per size)	100	30	30
	200	30	30
	300	30	30
	400	28	28
	500	29	28
Ortho-ortho (30 inst. per size)	100	30	30
	200	30	30
	300	29	30
	400	30	30
	500	29	28

Table 2 Number of instances solved to optimality when using or not the Lagrangian Heuristic method.

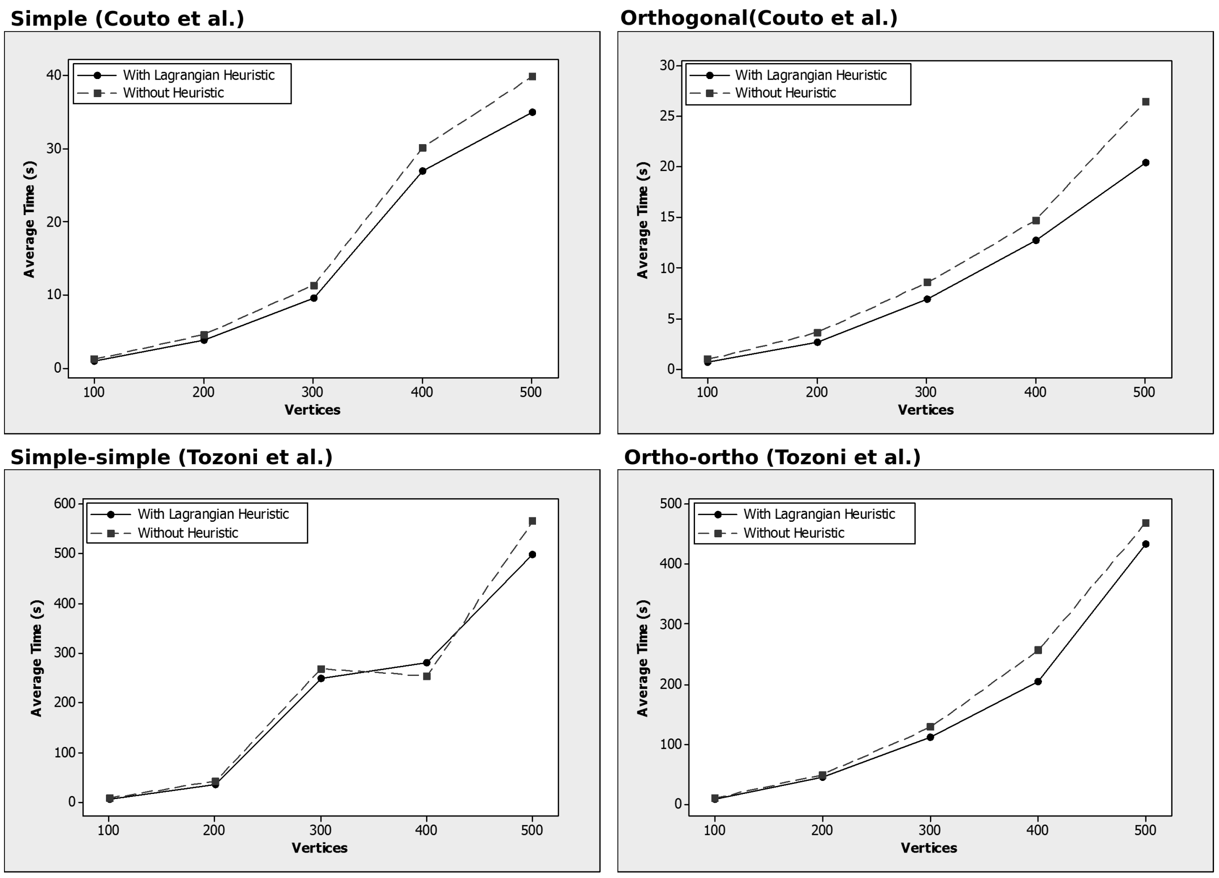

Fig. 10 Comparison between using and not using the Lagrangian heuristic on four different types of polygons.

5.6 Comparison with other State of Art Techniques

Recently, some techniques achieved good practical results in the search for optimal solutions for the AGP. In 2011, Bottino and Laurentini [5] presented a heuristic based on an edge covering algorithm that lead to some nice results, while still not able to achieve optimality for a large set instances. A full comparison between our technique and that heuristic appeared in [17].

Through a rather different approach, Kröller et al. [12], in 2012, presented a method that reached optimality for many large and complex instances. For the sake of comparison, we also ran our implementation on the same instances used in that paper. Table 3 shows the average running times and optimality rates of the two methods. In comparing these results, one has to bear in mind the computational environments in which the two experiments were conducted were rather different.

Instance Group	n	**Optimality Rates**		**Average Time (s)**	
		Method [12]	**Our Method**	**Method [12]**	**Our Method**
Simple (30 inst. per size)	60	80%	100%	0.70	0.26
	100	64%	100%	29.40	0.94
	200	44%	100%	14.90	3.77
	500	4%	100%	223.30	35.04
Orthogonal (30 inst. per size)	60	80%	100%	0.40	0.15
	100	54%	100%	1.10	0.64
	200	19%	100%	4.30	2.66
	500	7%	100%	25.30	20.44
Von Koch (30 inst. per size)	60	77%	100%	2.40	0.26
	100	71%	100%	5.50	0.87
	200	70%	100%	18.90	6.63
	500	15%	100%	205.00	76.45
Spike (30 inst. per size)	60	67%	100%	1.70	0.49
	100	68%	100%	3.90	1.40
	200	61%	100%	15.30	7.30
	500	52%	100%	190.20	173.23

Table 3 Comparison between the method of Kröller et al. [12] and ours.

It can be seen from Table 3 that our technique reached optimal solutions for *all* instances within a small average running time. While we initially allowed a time limit of 60 minutes, as opposed to Kröller et al.'s (see [12]) 20 minute upper bound on the total running time, no more than ten minutes were actually required for our method to converge in all those instances. As said before, this statistic has to be read with caution since the computing environments used were not comparable. A deeper look into the time performance our method, however, dismisses the hasty conclusion that this difference might explain its higher success rate. In the histogram of Figure 11 we show the total number of instances solved by our algorithm within certain discrete amounts of time. Note the remarkable fact that $\sim 90\%$ of the instances were solved roughly within two minutes.

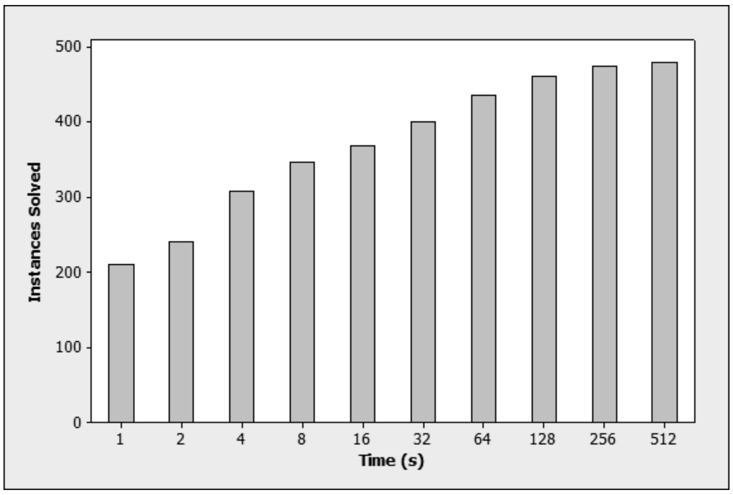

Fig. 11 Number of instances solved by our method within discrete amounts of time.

5.7 General Experiments

Having analyzed our method in comparison to other approaches, let us focus on the overall performance of our algorithm on instances from all available sources.

Firstly, Table 4 sums up the results for polygons of three different sizes from each instance group, in order to characterize the behavior of our method on the widest group of instances possible.

Source	**Instance Group**	n	**Optimality Rate**	**Guards**	**Iterations**	**Time (s)**
Couto et al.	Simple (30 inst. per size)	100	100.00%	13.17	1.93	0.94
		1000	100.00%	126.57	4.50	181.27
		2500	76.67%	312.39	5.78	2349.84
	Orthogonal (30 inst. per size)	100	100.00%	14.63	2.50	0.64
		1000	100.00%	147.90	5.40	120.29
		2500	100.00%	371.40	6.87	1139.80
	Von Koch (30 inst. per size)	100	100.00%	7.83	1.70	0.87
		1000	100.00%	56.80	2.37	723.54
		1500	73.33%	95.00	2.67	2584.83
Bottino and Laurentini	Simple (20 inst. per size)	40	100.00%	5.55	1.15	0.10
		50	100.00%	6.60	1.60	0.24
		60	100.00%	8.35	1.30	0.27
	Orthogonal (20 inst. per size)	40	100.00%	6.00	1.40	0.07
		50	100.00%	7.70	1.55	0.12
		60	100.00%	9.10	1.50	0.16
Kröller et al.	Spikes (30 inst. per size)	100	100.00%	4.37	1.07	1.40
		200	100.00%	6.87	1.00	7.30
		500	100.00%	9.77	1.00	173.23
Tozoni et al.	Simple-simple (30 inst. per size)	100	100.00%	13.13	2.27	4.92
		200	100.00%	24.30	3.47	34.68
		500	93.33%	59.68	5.82	498.69
	Ortho-ortho (30 inst. per size)	100	100.00%	12.27	2.97	7.88
		200	100.00%	24.97	4.90	45.29
		500	93.33%	62.46	6.29	433.03

Table 4 Summary of results.

Clearly, some classes of polygons seem easier than others: orthogonal polygons tend to take less time and reach higher optimality rate than simple or von Koch polygons. The latter are clearly the hardest ones: within the average time taken to solve a von Koch instance of 1000 vertices, we were able to solve simple polygons of double that size. In the same vein, it is evident that polygons with holes are at least an order of magnitude harder than their hole-free counterpart.

Although average time information can be useful for a general analysis, it is often necessary to observe the behavior on particular instances. In Figure 12, three box-plot charts present the running time results for different classes of polygons. In these charts, it is possible to perceive that the average times (from Table 4) for a given class and size of polygons may be quite disparate from the median, or even larger than the third quartile, due to outliers. For instance, consider the Ortho-ortho polygons with 200 vertices: the average time to solve the AGP is 45.29 seconds, while the median and the third quartile are 24.96 and 41.38 seconds, respectively. It is clear from these numbers that the instance classes contain polygons which differ considerably from each other, leading to AGP instances with very diverse degrees of difficulty. The high success rate of our algorithm in such a heterogeneous benchmark indicates that the method is very robust.

Quite informative also is an analysis of how the total running time is divided between the different computational tasks, in practice. While one would expect that the larger amount of time should be spent solving the ILP (SCP) instances, after all these are $\mathbb{NP}$-hard problems, we observe from the histogram in Figure 13 that the time spent solving ILP models is usually lower than half of the total time. This can be credited to the high quality of modern solvers, such as XPRESS, but, in our case, also to the improvements made in our implementation, such as matrix reduction and the Lagrangian heuristic. Of course, were we to consider ever growing instance sizes, this behavior is expected to change, asymptotically.

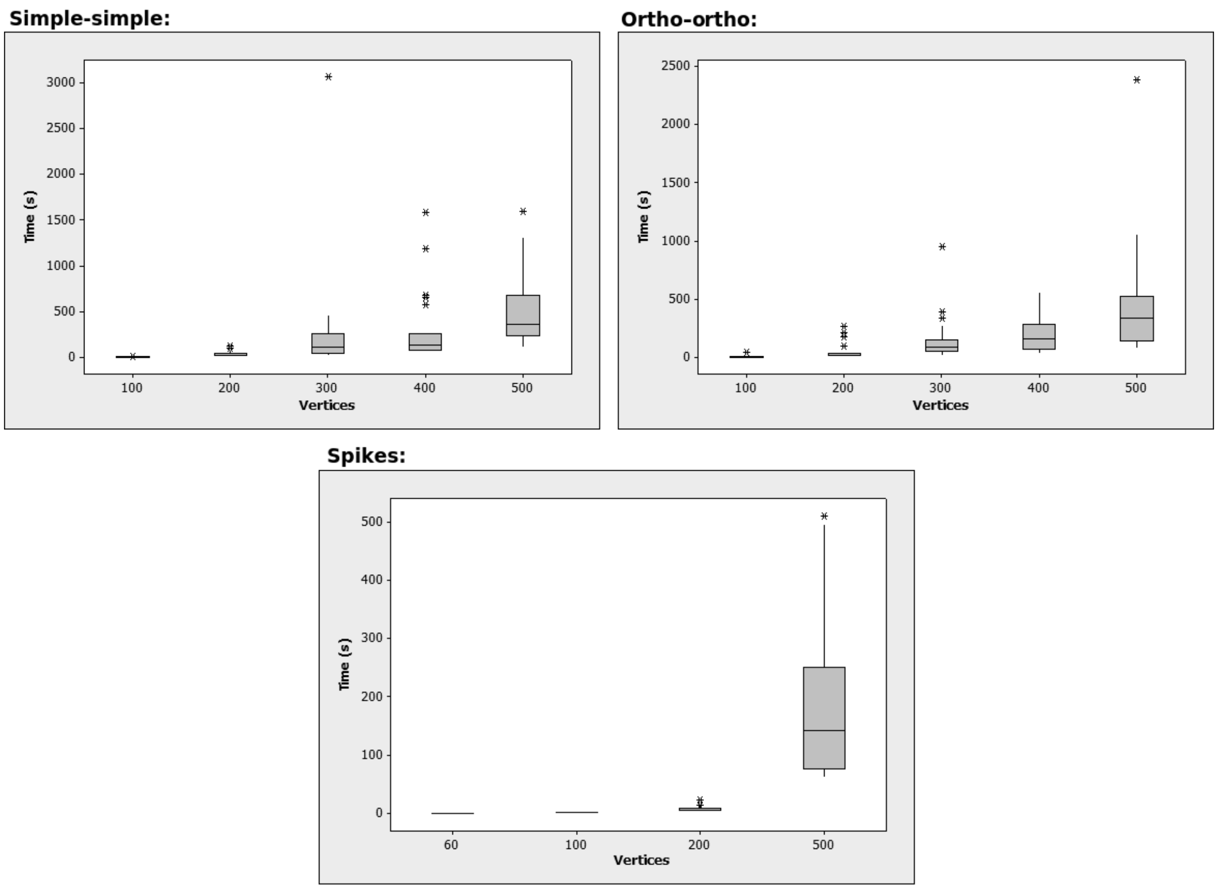

Fig. 12 Box-plot charts summarizing information about the running times to solve Simple-simple, Ortho-ortho and Spike instances.

Needless to say, the amount of time spent in the resolution of ILP models is proportionally larger for instances with holes as a consequence of the denser and more complex arrangements, which lead to larger and harder ILP instances.

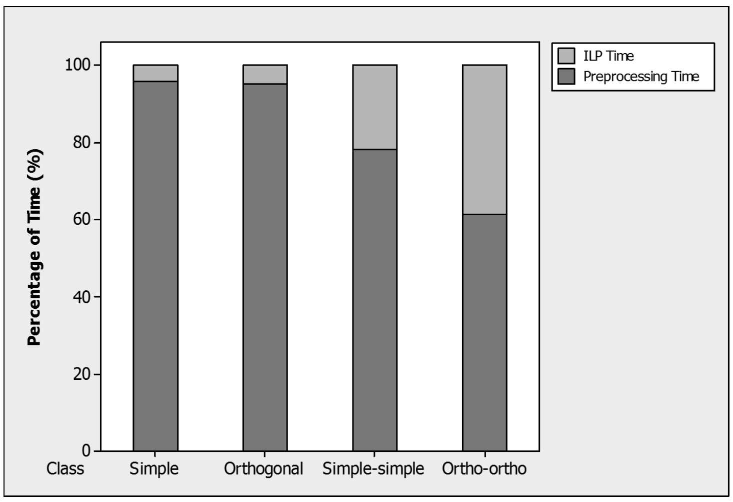

Fig. 13 Comparison between preprocessing time and solver time for polygons of 500 vertices.

6 Conclusion

For many years, much theoretical knowledge has been gathered on a number of hard geometric problems. More recently, the need for solving these problems to optimality in practice has gained much interest. In particular, the use of ILP formulations associated with state of the art solvers has proven very successful in attaining exact solutions for large instances efficiently.

Following the trend of [8,12,17], this paper shows that this approach is proficuous to solving the Art Gallery Problem with point guards to optimality. The ILP model presented here has been implemented and tested in a total of 2440 instances and, for all but 32 of them, optimal solutions were obtained in less than an hour of computational time.

While convergence remains dependent on the initial discretization, the current results already show an efficacy rate of more than 98%, which is an achievement unattained up to this point in time.

It remains a challenge for the future to study additional geometric properties that may assist in designing a discretization strategy that leads to proven convergence of the method described in this work.

In the near future, we will make our code available to other researchers and practitioners. This will require some testing with free ILP solvers such as GLPK (GNU Linear Programming Kit). Although some loss of performance is expected after the replacement of the commercial solver, we believe our implementation will be an invaluable tool for those interested in studying the AGP since, to the best of our knowledge, to this date, no robust code to tackle this problem has been made available.

Acknowledgements We would like to thank Alexander Kröller and Sandór Fekete for fruitful discussions and for providing additional sets of instances for our experiments. We are also grateful to Andrea Bottino for promptly dispensing the instances used in his work for the tests conducted in our investigation.

References

1. Y. Amit, J. S. B. Mitchell, and E. Packer. Locating guards for visibility coverage of polygons. In *ALENEX*, New Orleans, Lousiana, January 2007.
2. J. E. Beasley. Modern heuristic techniques for combinatorial problems. chapter Lagrangian relaxation, pages 243–303. John Wiley & Sons, Inc., New York, NY, USA, 1993.
3. D. Borrmann, P. J. de Rezende, C. C. de Souza, S. P. Fekete, S. Friedrichs, A. Kröller, A. Nüchter, C. Schmidt, and D. C. Tozoni. Point guards and point clouds: solving general art gallery problems. In *Proceedings of the twenty-ninth annual symposium on Computational geometry*, SoCG '13, pages 347–348, New York, NY, USA, 2013. ACM.
4. P. Bose, A. Lubiw, and J. I. Munro. Efficient visibility queries in simple polygons. *Computational Geometry*, 23(3):313–335, 2002.
5. A. Bottino and A. Laurentini. A nearly optimal algorithm for covering the interior of an art gallery. *Pattern Recognition*, 44(5):1048–1056, 2011.
6. CGAL. Computational Geometry Algorithms Library. www.cgal.org (last access January 2012).
7. K.-Y. Chwa, B.-C. Jo, C. Knauer, E. Moet, R. van Oostrum, and C.-S. Shin. Guarding art galleries by guarding witnesses. *Intern. Journal of Computational Geometry And Applications*, 16(02n03):205–226, 2006.
8. M. C. Couto, P. J. de Rezende, and C. C. de Souza. An exact algorithm for minimizing vertex guards on art galleries. *International Transactions in Operational Research*, 18(4):425–448, 2011.
9. S. K. Ghosh. Approximation algorithms for art gallery problems. In *Proc. Canadian Inform. Process. Soc. Congress*, 1987.
10. S. K. Ghosh. *Visibility Algorithms in the Plane.* Cambridge University Press, New York, 2007.
11. S. K. Ghosh. Approximation algorithms for art gallery problems in polygons. *Discrete Applied Mathematics*, 158(6):718–722, 2010.
12. A. Kröller, T. Baumgartner, S. P. Fekete, and C. Schmidt. Exact solutions and bounds for general art gallery problems. *J. Exp. Algorithmics*, 17(1):2.3:2.1–2.3:2.23, May 2012.
13. D. T. Lee and A. Lin. Computational complexity of art gallery problems. *Information Theory, IEEE Transactions on*, 32(2):276–282, March 1986.
14. J. O'Rourke. *Art Gallery Theorems and Algorithms.* Oxford University Press, New York, NY, 1987.
15. T. Shermer. Recent results in art galleries. *Proceedings of the IEEE*, 80(9):1384–1399, September 1992.
16. D. C. Tozoni, P. J. de Rezende, and C. C. de Souza. The Art Gallery Problem project, 2013. *www.ic.unicamp.br/∼cid/Problem-instances/Art-Gallery/AGPPG.*
17. D. C. Tozoni, P. J. de Rezende, and C. C. de Souza. The quest for optimal solutions for the art gallery problem: A practical iterative algorithm. In V. Bonifaci, C. Demetrescu, and A. Marchetti-Spaccamela, editors, *SEA*, volume 7933 of *Lecture Notes in Computer Science*, pages 320–336. Springer, 2013.
18. J. Urrutia. Art gallery and illumination problems. In J. R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 973–1027. North-Holland, 2000.
19. J. van Leeuwen and A. Schoone. *Untangling a Traveling Salesman Tour in the Plane.* Rijksuniversiteit. Vakgroep Informatica, 1980.
20. XPRESS. *Xpress Optimization Suite*, 2009. http://www.fico.com/en/Products/DMTools/Pages/FICO-Xpress-Optimization-Suite.aspx (access January 2012).