Exact Solutions and Bounds for General Art Gallery Problems

ALEXANDER KRÖLLER, TOBIAS BAUMGARTNER, SÁNDOR P. FEKETE, and CHRISTIANE SCHMIDT, Braunschweig University of Technology

The classical Art Gallery Problem asks for the minimum number of guards that achieve visibility coverage of a given polygon. This problem is known to be NP-hard, even for very restricted and discrete special cases. For the case of vertex guards and simple orthogonal polygons, Cuoto et al. have recently developed an exact method that is based on a set-cover approach. For the general problem (in which both the set of possible guard positions and the point set to be guarded are uncountable), neither constant-factor approximation algorithms nor exact solution methods are known.

We present a primal-dual algorithm based on linear programming that provides lower bounds on the necessary number of guards in every step and—in case of convergence and integrality—ends with an optimal solution. We describe our implementation and give experimental results for an assortment of polygons, including nonorthogonal polygons with holes.

Categories and Subject Descriptors: F.2.2 [**Analysis of algorithms and problem complexity**]: Nonumerical Algorithms and Problems

General Terms: Algorithms, Experimentation, Theory

Additional Key Words and Phrases: Art Gallery Problem, linear programming, duality, separation, geometry, visibility

ACM Reference Format:
Kröller, A., Baumgartner, T., Fekete, S. P., and Schmidt, C. 2012. Exact solutions and bounds for general art gallery problems. ACM J. Exp. Algor. 17, 2, Article 2.3 (May 2012), 23 pages.
DOI = 10.1145/2133803.2184449 http://doi.acm.org/10.1145/2133803.2184449

1. INTRODUCTION

The classical Art Gallery Problem (AGP) asks for the minimum number of guards placed inside of a polygon that suffice to perceive the entire polygon (interior and edges). For several classes of polygons and variants on the placement of guards, this problem was shown to be NP-hard (e.g., Lee and Lin [1986]). Originally, interest from the theoretical side focused on extremal results, like the classical $\lfloor \frac{n}{3} \rfloor$ bound first established by Chvátal [1975] and very elegantly proven by Fisk [1978].

On the practical side, good solutions to the AGP have gained in importance, for example, for measuring (the interior of) buildings using a static laser scanner. For these measuring tasks, positions of the laser scanner must be identified that ensure coverage of the given environment, for example, a production hall, a tunnel or a bridge construction (Figure 1). Hence, a solution to the AGP or good upper bounds enable the company to reduce the working hours, both during the actual scan process on site and for the postprocessing (e.g., scan matching). This also applies to lower bounds: not only do they allow quality estimates of feasible solutions, but they are also of crucial

Authors' address: A. Kröller, T. Baumgartner, S. P. Fekete, and C. Schmidt, Braunschweig University of Technology.

DOI 10.1145/2133803.2184449 http://doi.acm.org/10.1145/2133803.2184449

(a) (b)

Fig. 1. A 360° laser scanner of inmetris3D placed in a production hall (a) and in a tunnel with the resulting scan data (b). (All images courtesy of inmetris3D.)

importance for contract bidding, allowing an estimate of the necessary expenses for personnel and equipment that cannot be avoided, neither by the company nor its competitors.

Our Contribution. We develop a primal-dual approach for general AGPs in arbitrary polygons with holes, in which guards can be placed anywhere, such that the entire interior of the polygon is guarded. Our method computes a sequence of lower and upper bounds on the optimal number of guards until—in case of convergence and integrality—eventually an optimal solution is reached. Our algorithm is based on a formulation of the problem as a (covering) linear program. It solves the problem using a cutting plane and column generation approach, that is, by solving the primal and dual separation problems. Computational results show the usefulness of our method.

The rest of the article is organized as follows. In Section 2, we describe related work; notation is provided in Section 3. In Section 4, the main part of our algorithm, using linear programming, is presented, together with a discussion of geometric aspects of separation. Section 5 discusses implementation aspects of the algorithm and presents heuristic ingredients. We evaluate our implementation using a set of test instances in Section 6. Problems of convergence for degenerate cases are discussed in Section 7. In Section 8, we present some geometry-based cutting planes. Finally, in Section 9, we discuss possible implications and extensions.

2. RELATED WORK

Art Gallery Problems. The question on which the AGP (see Urrutia [2000], O'Rourke [1987], and Shermer [1992]) is based was posed by Klee: How many (stationary) guards are needed in an art gallery in order to ensure that the guards can see all the exhibits? That is, one asks for the minimum number of immobile guards, $G(P)$, that cover all of P. The maximum $G(P)$ over all polygons of n vertices is denoted by $g(n)$. Two points in a polygon are defined to be visible to each other if the line connecting the points lies inside of P.

Chvátal [1975] was able to show that $\lfloor \frac{n}{3} \rfloor$ (stationary) guards are sometimes necessary and always sufficient so that every point of a polygon with n vertices is visible from at least one guard position ($g(n) \leq \lfloor \frac{n}{3} \rfloor$). Fisk [1978] gave a short and simple proof for Chvátal's result. The function $g(n)$ was also considered for more restricted classes of polygons, for example, Kahn et al. [1983] established $g(n) \leq \lfloor \frac{n}{4} \rfloor$ for orthogonal polygons.

While the work of Chvátal and Fisk concentrated on a sufficient number of guards and resulted in the Art Gallery Theorems mentioned earlier, the AGP asks for a minimum set of points (the guards, G) in a given polygon P, such that every point in P is visible from at least one point in G. O'Rourke and Supowit [1983] were able to show that

this problem is NP-hard by a reduction from 3SAT, for guards restricted to be located on vertices, and for polygons with holes. Lee and Lin [1986] were able to prove NP-hardness for simple polygons. This result was extended to point guards by Aggarwal (see O'Rourke [1987]); Schuchardt and Hecker [1995] gave NP-hardness proofs for rectilinear simple polygons, both for point and vertex guards.

Algorithms for placing $g(n)$ guards that see the entire polygon have also been considered. Avis and Toussaint [1981] gave an algorithm based on Fisk's triangulation proof of Chvátal's theorem to place $\lfloor \frac{n}{3} \rfloor$ guards in $O(n \log n)$ time.

Another approach is to consider the closely related problem of covering polygons. The part of a polygon that is visible to a single guard is a star-shaped polygon, so that covering with star-shaped pieces corresponds to placing a set of guards. Keil [1986] gave an $O(n^2)$ algorithm for covering a horizontally convex orthogonal polygon with star-shaped polygons, implying that for this restricted class of polygons, a minimum set of guards can be found in polynomial time. For other types of covers (e.g., covering with rectangles), the resulting number is not the optimal set of guards, but it may yield an approximation.

Beside various classes of polygons to be guarded (orthogonal polygons, polygons with or without holes etc.), variations on the abilities of the guards have been examined. In the classical AGP the guards are immobile, that is, bound to a single point that may be located at any possible position inside of P; these are called *point guards*. If the feasible locations are restricted to vertices, we deal with *vertex guards*. Moreover, guards may have the ability to move along certain structures. *Edge guards* are allowed to move along an edge and survey all points visible to some point on this edge. Instead of patrolling along an edge, *diagonal guards* move along diagonals; *mobile guards* are allowed to use both. See Shermer [1992] for these definitions. Alternatively, the guarding task may be varied: The guards may only be required to survey the edges (and not the interior) of the polygon [Laurentini 1999].

In recent years, there has been a growing amount of work dealing with algorithmic aspects of the AGP. Even for the restricted case of vertex guards and simple polygons, Eidenbenz et al. [2001] established lower bounds on the achievable approximation ratio. On the other hand, approximation algorithms are only known for restricted versions of the problem (e.g., Efrat and Har-Peled [2006]) and allow for a logarithmic approximation ratio; one of the reasons is that both the set of possible guard locations and the set that is to be covered have infinite cardinality, and no easy reduction to discrete sets is known.

Considering a single mobile guard leads to the *Watchman Problem*, which was first formulated by Chin and Ntafos [1986]. It asks for the shortest tour of a watchman inside of a polygon, such that each point of the polygon becomes visible at least once along this tour. This problem is NP-hard for some classes of polygons and polynomially solvable for several classes. For example, Chin and Ntafos showed that the watchman problem is NP-hard for polygons with holes (even for rectilinear polygons and holes) and gave polynomial algorithms for rectilinear polygons [Chin and Ntafos 1986], as well as for general simple polygons [Chin and Ntafos 1991].

Lower Bound on the Number of Guards. As described earlier, the AGP is NP-hard. For some classes of polygons, Eidenbenz et al. [2001] showed NP-hardness of getting better constant approximation factors below $1 + \delta$ for an appropriate $\delta > 0$. Therefore, the practical and theoretical relevance of the problem makes it important to consider approximation algorithms and good heuristics. To determine the quality of such solutions, we need a good reference point given by a lower bound. We will present such a lower bound in the next sections.

Previous work mostly consists of heuristics [Amit et al. 2007] with good practical performance. To the best of our knowledge, there is little work on good lower bounds. Amit et al. [2007] are the only ones to consider a lower bound for the general AGP, with the entire polygon to cover. They use the lower bound to compare heuristic solutions, with the guards taken from different candidate sets, such as vertices and more enhanced points.

Their lower bound is based on the visibility polygons of a certain set of points. Considering the visibility polygons of two points, we know that we need at least two guards to survey these points in case the visibility polygons are disjoint. Therefore, a candidate point set S is chosen, and we build a graph with vertices for these points, plus edges in case of intersecting visibility polygons. Then, we search for the maximum independent set. Because finding a maximum independent set is NP-hard, Amit et al. [2007] use a greedy strategy: Iteratively add the node with the smallest degree to the set I, and remove this node and its neighbors from S. This greedy approximation algorithm has a performance guarantee of only $1/(|E|/|V| + 1)$ [Jansen and Margraf 2008]. The cardinality $|I|$ of the set I gives a lower bound on the number of guards necessary to cover S, and thus a lower bound on the guards needed for P. Amit et al. include convex vertices and midpoints of edges incident to two reflex vertices to the candidate sets.

Lower bounds for the version of the AGP with guards surveying only the edges of the polygon are given by Bottino and Laurentini [2008]. The authors build an initial lower bound and apply rules to find indivisible edges in order to improve this bound. The computation of the initial lower bound is based on the same approach as the bound of Amit et al. [2007]: The maximum independent set problem for a graph with nodes for the edges of P and edges in case of nondisjoint weak visibility polygons is to be solved. A *weak visibility polygon* of an edge e is the polygon whose points see at least one point of e; points of the *integer visibility polygon* see all points of e. Bottino and Laurentini use an exact branch-and-bound algorithm for the maximum independent set problem. Furthermore, the authors aim at the improvement of the bound. For this purpose, they determine the indivisible edges: An edge is indivisible if there exists an optimal set of guards such that the edge is entirely observed by at least one guard. There is a simple rule to determine such an edge: In case the weak and the integer visibility polygon of an edge coincide, the edge is indivisible. Bottino and Laurentini give some more sophisticated rules.

Exact Solutions for Vertex Guards. Couto et al. [2007, 2008, 2009] consider exact solutions for a special case of the AGP. Compared to the general problem that we consider, their variant has two additional restrictions:

(1) polygons are simple, that is they have no holes, and
(2) guards may only be placed on polygon vertices.

Their algorithms, therefore, benefit from a small and finite set of covering points in combination with various ways of reducing the candidate set of points that need to be covered. Their approach is based on a set-cover integer program, applied to a grid discretization, which is iteratively refined if necessary. They are able to bound the number of iterations, but they have to solve an instance of the set-cover problem in each iteration. In experiments with polygons of up to 200 vertices [Couto et al. 2008], and later, with a variety of discretization strategies of up to 1,000 and 2,500 vertices [Couto et al. 2009], they are able to solve these instances within 100 to 1,000 seconds (for the different strategies and 1,000 vertices.) Note that instance sizes and computation times are not comparable to our results, as we consider a more general problem.

3. DEFINITIONS AND NOTATION

We consider a given polygon P, possibly with holes. For a point $p \in P$, the visibility polygon $\mathcal{V}(p)$ is the (star-shaped) set of all points of P visible from p. A guard set $G \subset P$ covers P if $\cup_{g \in G} \mathcal{V}(g) = P$. The AGP asks for such a guard set of minimum cardinality. Note that visibility is symmetric, that is, $p \in \mathcal{V}(q) \iff q \in \mathcal{V}(p)$. The inverse of $\mathcal{V}(\cdot)$ describes all points that can see a given point p. This is easily confirmed to be

$$\mathcal{V}^{-1}(p) := \{q \in P : p \in \mathcal{V}(q)\} = \mathcal{V}(p) .$$

The points of P have two roles in this problem. First, points are selected to place guards on them. To reflect this, we refer to them as *guard positions*. Second, every point needs to be covered by the visibility polygon of at least one guard. An uncovered point, therefore, certifies (witnesses) that a solution is infeasible. We refer to points that we watch for being covered as *witnesses*.

In the algorithm discussion, we frequently use vectors indexed by P, for example, $x = (x_p)_{p \in P}$. For such a vector and a set $Q \subseteq P$, we use the notation

$$x(Q) := \sum_{p \in Q} x_q ,$$

which is fairly common in the area of linear programming.

For linear programs, the *separation problem* is defined as follows: Given an instance of a linear programming problem, for example, $\max\{c^T x \mid Ax \leq b\}$ and a point y, determine whether y belongs to the polyhedron $\{x \mid Ax \leq b\}$ or not, and in the latter case, present a violated constraint. Depending on whether we consider the primal or dual linear program, we speak of the *primal separation problem* or *dual separation problem*. It is known that the ability to solve the separation problem in polynomial time is sufficient to efficiently solve a problem to optimality [Grötschel et al. 1981], even if the formulation contains an infinite number of constraints. This result is based on the ellipsoid method, which is not usable in practice. However, the simplex method is practically efficient and is able to incorporate additional constraints or variables quickly [Schrijver 1986]. It is possible to solve large problems by starting with a relaxation and then iteratively solve the primal and/or dual separation problem to add in primal or dual constraints that are violated in the current solution. In case of primal separation, the approach is called the *cutting-plane method* (as it cuts away infeasible solutions through new constraints). In case of the dual, it is called *column generation*, as adding violated dual constraints corresponds to introducing new primal variables (i.e., columns).

4. AN LP-BASED PROCEDURE

4.1. LP Formulations

Our approach to the AGP focuses on good lower bounds. It may also find upper bounds, that is, feasible solutions, although this cannot be guaranteed.

The AGP can be trivially formulated as a linear program with infinitely (actually uncountably) many binary variables and inequalities:

$$\min \quad \sum_{g \in P} x_g \tag{1}$$

$$\text{s.t.} \quad \sum_{g \in \mathcal{V}(w)} x_g \geq 1 \quad \forall w \in P \tag{2}$$

$$x_g \in \{0, 1\} \qquad \forall g \in P \tag{3}$$

Here a guard placement is modeled by setting $x_g = 1$ if an only if $g \in P$ is a guard position. Inequality (2) ensures that the polygon is fully covered. Note that the known upper bound of $\lfloor \frac{n}{3} \rfloor$ guarantees that the formulation is well defined.

Due to its infinite size, the formulation cannot be solved using linear programming techniques. Instead, we consider the relaxation AGR(G, W), in which we relax the integrality constraint (3), restrict the guard positions to be from a finite set $G \subset P$, and only require a finite set $W \subset P$ of witnesses to be covered. Therefore, AGR(P, P) is the standard LP relaxation of AGP, and AGR(G, W) is a relaxation of AGR(P, P). Throughout this article, we assume that every witness $w \in W$ is visible from at least one guard position $g \in G$. This ensures feasibility of the formulations.

AGR(G, W) can be formulated as follows:

$$\min \quad \sum_{g \in G} x_g \tag{4}$$

$$\text{s.t.} \quad \sum_{g \in G \cap \mathcal{V}(w)} x_g \geq 1 \quad \forall w \in W \tag{5}$$

$$0 \leq x_g \leq 1 \qquad \forall g \in G \tag{6}$$

The associated dual linear program of (4) through (6) reads as follows:

$$\max \quad \sum_{w \in W} y_w \tag{7}$$

$$\text{s.t.} \quad \sum_{w \in W \cap \mathcal{V}(g)} y_w \leq 1 \quad \forall g \in G \tag{8}$$

$$0 \leq y_w \leq 1 \qquad \forall w \in W \tag{9}$$

Unsurprisingly, the primal LP is a fractional covering problem using fractional guards, and the dual is a fractional packing problem.

An optimal solution to AGR(G, W) has no straightforward interpretation in terms of the original AGP. It is neither an upper bound (as there may be points in $P \setminus W$ that are not sufficiently covered), nor a lower bound (as there may be guard positions in $P \setminus G$ that allow for covering with fewer guards). We establish a connection between AGR(G, W) and AGP by analyzing the separation problems associated with the LP formulation. Let x^* be an optimal solution to (4) through (6) with associated dual solution y^*.

The primal separation problem for AGR(P, P) asks whether x^* violates a primal constraint. Given that x^* is feasible for AGR(G, W) by definition, and because the variable bounds are not relaxed in AGR(G, W), such a constraint can only be of type (5) for some witness $w \in P \setminus W$. The separation problem, therefore, asks to identify a point $w \in P \setminus W$, for which $x^*(G \cap \mathcal{V}(w)) < 1$ holds. The following cases result.

(a) If such a point w exists, it proves that x^* is not feasible for AGR($G, W \cup \{w\}$) (and, therefore, also AGR(P, P)).
(b) If no such point exists, it proves that x^* is feasible and optimal for AGR(G, P). It is, therefore, an upper bound for AGR(P, P).
Furthermore, if it additionally happens that x^* is integral (i.e., $x_g^* \in \{0, 1\}\ \forall g \in G$), it also defines a feasible solution for the original AGP (i.e., an upper bound).

To solve the separation problem, we translate it to geometric terms. Consider the overlay of the visibility polygons of all points $g \in G$ with $x_g^* > 0$. This decomposes P into a planar arrangement. It is easy to see that the coverage function $c(p) := x^*(G \cap \mathcal{V}(p))$ is constant over every face and edge of the arrangement. Therefore, an algorithm merely

ALGORITHM 1: LP-based routine

```
1  Generate initial G ⊂ P, W ⊂ P.
2  repeat
3      Solve AGR(G, W), get optimal solution x*, dual y*, and objective value v
4
5      Run primal separation
6      if primal separation produces a point w then
7          W ← W ∪ {w}
8      else
9          Output "v is an upper bound to AGR(P, P)"
10         if x* is integral then
11             Output "x* is feasible for AGP, v is an upper bound for AGP"
12
13     Run dual separation
14     if dual separation produces a point g then
15         G ← G ∪ {g}
16     else
17         Output "v is a lower bound to AGR(P, P) and to AGP"
18 until both primal and dual separation failed, or lower and upper bounds meet
```

has to iterate over the faces, edges, and vertices of this arrangement to find one where coverage drops below 1. In fact, due to the bound $x^* \geq 0$, it is sufficient to just iterate over the faces, as the coverage function has its minima just there.

Now let us repeat these arguments for the dual problem. The dual separation problem is to identify a guard position $g \in P \setminus G$ with $y^*(W \cap \mathcal{V}(g)) > 1$. Again, two cases emerge.

(c) If such a point exists, it solves the separation problem for AGR($G \cup \{g\}, W$) and proves that x^* is not optimal for AGR(P, W).
(d) If no such point exists, x^* is optimal for AGR(P, W). As AGR(P, W) is a relaxation of AGR(P, P), the objective value is a lower bound for it and, therefore, also the original AGP.

Due to the symmetry of visibility, the dual separation problem can be solved in the same manner as the primal. We compute the overlay of the visibility polygons of all points $w \in W$ with $y^*_w > 0$, and check the arrangement for points g that satisfy $y^*(W \cap \mathcal{V}(g)) > 1$. This time, it is sufficient to consider only the vertices of the arrangement. Finally, if neither primal nor dual separation produce additional points, we can deduce that x^* is feasible and optimal for AGR(P, P).

The previously described observations translate directly into an algorithm (see Algorithm 1). It is trivial to see that, should the algorithm terminate, it will have found an optimal solution to AGR(P, P) and thus a lower bound for AG. Unfortunately, there are degenerate problem instances for which the algorithm will never terminate (see Section 7 for an example).

There are three steps in which the algorithm's behavior can be influenced. First, what should the initial G and W be? Guessing a hopefully good (or even optimal) solution to AG and using the guard positions for G can be a great speed-up for the algorithm. Similarly, a clever choice of W can improve algorithm runtime significantly. Finally, the separation phases leave several options: Prioritizing separating guards produces more lower bounds, while prioritizing witness separation produces more upper bounds. All of this is of purely heuristic nature and is discussed in Section 5.

4.2. Correctness of the Algorithm

LEMMA 4.1 (LOWER BOUND). *In case Algorithm 1 terminates, the returned objective value is a lower bound for AGP.*

PROOF. When Algorithm 1 terminates, neither a witness point for condition (a) nor a guard point for condition (c) exists. Therefore, Inequalities (5) and (8) hold for $W = P$ and $G = P$, that is, the current solution x^* is feasible and optimal for AGR(P, P). AGR(P, P) is the relaxation for AGP, the linear program formulated in Inequalities (1) through (3). If we denote the optimal objective values for AGR(P, P) and AGP by opt(AGR(P, P)) and $opt(AGP)$, respectively, we have: opt(AGR(P, P)) $\leq opt(AGP)$. Consequently, the objective value opt(AGR(P, P)) of x^* is a lower bound for $opt(AGP)$. □

LEMMA 4.2 (UPPER BOUND). *In case the algorithm outputs an upper bound for AGP in line 11, the returned value is an upper bound for AGP.*

PROOF. The algorithms gives an upper bound whenever condition (b) holds. If, in addition, x^* is integral, it is feasible for Inequality (3). Moreover, condition (b) assures feasibility for Inequality (2). Thus, x^* is a feasible solution for AGP, and hence it defines an upper bound for $opt(AGP)$. □

LEMMA 4.3 (OPTIMALITY). *In case Algorithm 1 terminates and the current solution x^* is integral, the returned objective value is the optimal value for AGP.*

PROOF. The current solution x^* is feasible and optimal for AGR(P, P). As AGR(P, P) is the relaxation of AGP, an optimal integer solution for AGR(P, P) is a feasible solution for AGP. We have opt(AGR(P, P)) $\leq opt(AGP)$; hence, it is also optimal for AGP. □

4.3. Geometric Aspects of the Separation Problem

Solving the primal and the dual separation problem means identifying a witness point $w \in W$ whose constraint is violated (i.e., $x(G \cap \mathcal{V}(w)) < 1$) or a guard position $g \in G$ whose constraint is violated (i.e., $y(W \cap \mathcal{V}(g)) > 1$). The coverage of a point in P depends on the visibility polygons of the guards (or witnesses) p with $x_p > 0$ ($y_p > 0$). Let $\mathcal{G}_s = \{p \in P : x_p > 0\}$, $g_s = |\mathcal{G}_s|$, $\mathcal{W}_s = \{p \in P : y_p > 0\}$, and $w_s = |\mathcal{W}_s|$.

This means that a crucial subroutine consists in computing visibility regions inside of the polygon P; an important complication arises from the fact that P may be a nonsimple polygon with h holes. This problem has received a considerable amount of attention. Suri and O'Rourke [1986] were the first who gave an algorithm for computing a visibility polygon inside a polygon P that may have holes. Their algorithm performs an angular plane sweep and runs in $O(n \log n)$ time. At this point, we have chosen a similar approach for ease of implementation. This theoretical runtime can be improved to $\Theta(n + h \log h)$ by using a method by Heffernan and Mitchell [1995].

For the primal and dual separation, we use the same polygon P. Hence, we can use the query versions with one preprocessing step. For each separation, we need to compute the arrangement of g_s (w_s for the dual separation) visibility polygons.

At this point, we have not made use of preprocessing approaches, as the most powerful methods require a large amount of memory. For example, using the algorithm of Zarei and Ghodsi [2005], we need $O(g_s(1 + h') \log n + \sum_{q \in \mathcal{G}_s} |\mathcal{V}(q)|)$ time to construct the arrangement for the primal separation, with $O(n^3 \log n)$ preprocessing and $O(n^3)$ space. Other options include the method of Pocchiola and Vegter [1993], which uses the visibility complex to determine the visibility polygon $\mathcal{V}(q)$ for a query point q in $O(|\mathcal{V}(q)| \log n)$ with $O(n \log n)$ preprocessing time using $O(n)$ space. Zarei and Ghodsi [2005] presented an algorithm that determines $\mathcal{V}(q)$ in time $O((1+h') \log n + |\mathcal{V}(q)|)$ (where $h' \leq \min(h, |\mathcal{V}(q)|)$) with $O(n^3 \log n)$ preprocessing time to construct a data structure of size $O(n^3)$. Another solution to the query version is given by Inkulu and

Kapoor [2009], who use a decomposition of the polygon into (simple) corridors and junctions and—depending on whether they use an approach from Aronov et al. [2002] or from Hershberger and Suri [1995] as a subprocedure for computing the visibility polygon in a simple polygon—achieve $O((1 + \min(h, |\mathcal{V}(q)|)) \log^2 n + h + |\mathcal{V}(q)|)$ or $O(|\mathcal{V}(q)| \log n + h)$ query time using $O(n^2 \log n)$ or $O(T + |E| + n \log n)$ preprocessing time and $O(n^2)$ or $O(\min(|E|, hn) + n)$ space, respectively. (Here, $|E|$ is the number of edges in the visibility graph, and $O(T)$ is the time to triangulate the given polygon).

5. IMPLEMENTATION

In this section, we describe implementation aspects of our algorithm. The software will be made available under the GNU General Public License.

Our software implements the LP-based procedure from Section 4. It uses standard linear-programming libraries to solve the LPs. Currently, SoPlex [Wunderling 1996] and ILOG CPLEX can be used for that matter, but generally any solver that provides an implementation of the primal-dual simplex algorithm could be included. Furthermore, we employ the Computational Geometry Algorithms Library [CGAL] for visibility queries and the separation problems. We make heavy use of the `Arrangement_2` package [Wein et al. 2008] for both.

The arrangements use exact rational arithmetic to avoid numerical issues. To keep the computational overhead low, a point-rounding heuristic is used. Every separated point, that is, every point that is identified by the primal or dual separation, will be perturbed to coordinates with smaller representation.

The implementation does not maintain the full overlay of all visibility polygons for G, respectively W, as these tend to become complex very quickly. We have observed that solution times improve when a different approach is used. For every point in G (W, respectively), we compute the visibility polygon and store it in a double-connected edge list (DCEL, see de Berg et al. [2008]). In the separation routines, we then compute the overlay only for those points that have a positive value in the current solution. As the cardinality of this point set is usually much smaller than the whole variable set, the overlay is much simpler.

5.1. Heuristic Ingredients

The algorithmic framework neither specifies how G and W should be initialized nor how the separation problem is to be solved. This is generally irrelevant for the theory of the algorithm, but not for the practice. We have implemented a number of different procedures for these problems.

Steering toward Bounds. Attempting to solve both the primal and the dual separation problem in each iteration is not neccessary. By adjusting which separator is used in an iteration, one can steer the algorithms toward lower or upper bounds. This steering is purely heuristic, and none of the strategies can provide guarantees. Our implementation offers four steering strategies.

—The *Both* strategy always runs both primal and dual separators in each iteration. The idea is to reach an optimal solution quickly, without caring to find bounds before the final step.
—*Primal* focuses on finding upper bounds (i.e., solutions). It always runs the primal separation routine. Dual separation only happens if the primal fails. This is motivated by the idea to find solutions often and gradually turn them into optimal ones.
—*Dual* is the counterpart of Primal. It favors dual separation over primal, in the hope of finding lower bounds quickly.

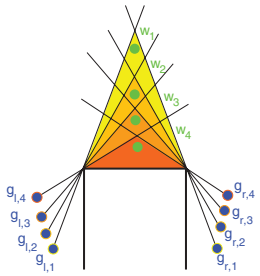


Fig. 2. "Creeping shadow" effect caused by placing witnesses in faces.

—*Stay* alternates between the two previous ones. It will start by only running primal separators, until this fails for the first time. A fractional solution is thus found. It then heads for a lower bound by using dual separators only, until they fail. This is repeated, resulting in an alternating sequence of upper and lower bounds. This separator exists in two flavors: $Stay_P$, which starts with primal separation, and $Stay_D$, starting with dual separation.

Point Separators. To solve the separation problems, we have implemented three different heuristics for each problem. Note that for the primal separation problem it is sufficient to consider only points in faces of the guard arrangement. This is because $x(G \cap \mathcal{V}(w))$ is constant for points w on such a face, and of same or higher value on the face's boundary. A similar argument holds for the dual separation problem. There it suffices to consider only vertices of the witness arrangement.

We provide the following separators. Two of them separate several points on each call, to avoid having to reoptimize the LP for every single new column or row.

—*Bulk* simply searches through the faces (vertices) of the guard (witness) arrangement to find points defining violated constraints. It returns an arbitrary selection of these points, with a user-provided limit on the number of points.
—*Maximally Violated* follows the principle of always separating using the maximally violated constraint, that is, $\arg\min_{w \in P} x(G \cap \mathcal{V}(w))$ for the primal problem and $\arg\max_{g \in P} y(W \cap \mathcal{V}(g))$ for the dual.
—*Greedy* attempts to find just a few points, which ideally become independent rows (columns) with large support in the LP. We model dual separation as a set-cover problem, where a minimal number violated guard points are used to cover the witnesses. The standard greedy algorithm is used to solve it. For the primal separation, we use an analogous procedure.
—*Edge Bulk* is available for primal separation only. It is based on Bulk and Maximally Violated, but it directly addresses the "creeping shadow" phenomenon. Consider Figure 2 for a visualization. If there are two guards $g_{\ell,1}$ and $g_{r,1}$ placed next to an obstacle, they cast an unguarded shadow behind it. Face-based primal separation solves this problem by placing a witness w_1 in the shadow. Dual separation can guard this witness by shifting the guards a little bit, leading to $g_{\ell,2}$ and $g_{r,2}$ that cover w_1, but still cast a shadow behind the obstacle, leading to another witness w_2. This can be iterated, resulting in a shrinking but never disappearing shadow. Simply placing a witness on the obstacle edge in the shadow stops this behavior. For this end, we implemented a heuristic that favors uncovered polygon edges over faces, and returns points based on a priority function.

Vertex Guards. The implementation offers solving the problem variant in which only vertex guards are allowed. This allows for direct comparison with the algorithm

by Couto et al. [2009]. It is implemented as a steering strategy, *Primal Only*. This strategy runs only the primal separator, thereby limiting the algorithm to the guard positions that are initially available. It is combined with initially placing a guard on every polygon vertex.

Initial Placements. There are several ways to initialize G and W. We have implemented and evaluated five heuristics.

—In the *All* heuristics, we put both a guard (a witness, respectively) on each polygon vertex. This means that the initial guard set allows for a feasible solution, although it is not neccessarily optimal.
—The *Other* heuristic places a guard (witness) on every other vertex, in the hope of speeding up the algorithm by having a smaller start set than with All.
—Another useful strategy would be to start with no guards and witnesses at all, shifting the work completely to the separators. For technical reasons, our implementation does not allow for such a state. Instead, we pick a single arbitrary vertex and put a guard (witness) on it. This method is called *One*.
—Another solution to have a guard set that suffices for a feasible solution is by placing guards on all reflex vertices of the polygon. This strategy is called *Reflex Vertices*. It is not useful for witnesses, as these vertices tend to have large visibility ranges (i.e., they are easy to cover).
—Finally, *Reflex Edges* is available for witnesses only. It places a witness on every polygon edge that is incident to a reflex vertex. The motivation behind this is to "hide" a witness behind every reflex vertex. This prevents creeping shadows.

6. COMPUTATIONAL RESULTS

We tested our implementation on a variety of different classes of polygons, including random polygons already tested by Couto et al. [2009]. In this section, we analyze how our algorithms perform on all these instances. For all figures in the following text, we show solutions to the primal problem (guards) and to the dual problem (witnesses). The coloring of the polygons makes it easier to visually test for feasibility. For the primal problem, gray denotes covering with a value of exactly 1, solutions with bluish coloring are feasible (they indicate a value >1), and solutions with a yellow-green coloring (indicating a value <1) are not. For the dual problem, gray again denotes covering with a value of exactly 1, solutions with a yellow-green coloring are feasible (they indicate a value <1), and solutions with bluish coloring (indicating a value >1) are not.

6.1. Examples

The first example (Figure 3) shows that switching from vertex guards to general guards may considerably improve the optimal value, at the expense of higher computational difficulty. Figure 3 depicts a random spike polygon (see Section 6.2): Long polygonal corridors crossing some "center" polygons. While using vertex guards, a single guard may only cover spikes whose boundary crosses precisely at the boundary of the centers (i.e., in most cases one or two points guards can be located on arbitrary positions in intersecting regions) and are such more likely to cover more spikes. In particular, point guards that cover spikes may also be used to guard areas around possible holes in the centers. In the example presented in Figure 3, we can observe a lower bound of 23 on the number of vertex guards necessary to guard the polygon (Figure 3, top), while 10 point guards suffice.

The second example (Figure 4) shows the floor plan of our institute, a nonorthogonal polygon with a single hole and a number of columns. We start with a greedily obtained set of witnesses (left), corresponding to an (infeasbile) dual solution; a corresponding set of guards is shown on the right. In iteration #3, the dual problem is feasible, but the

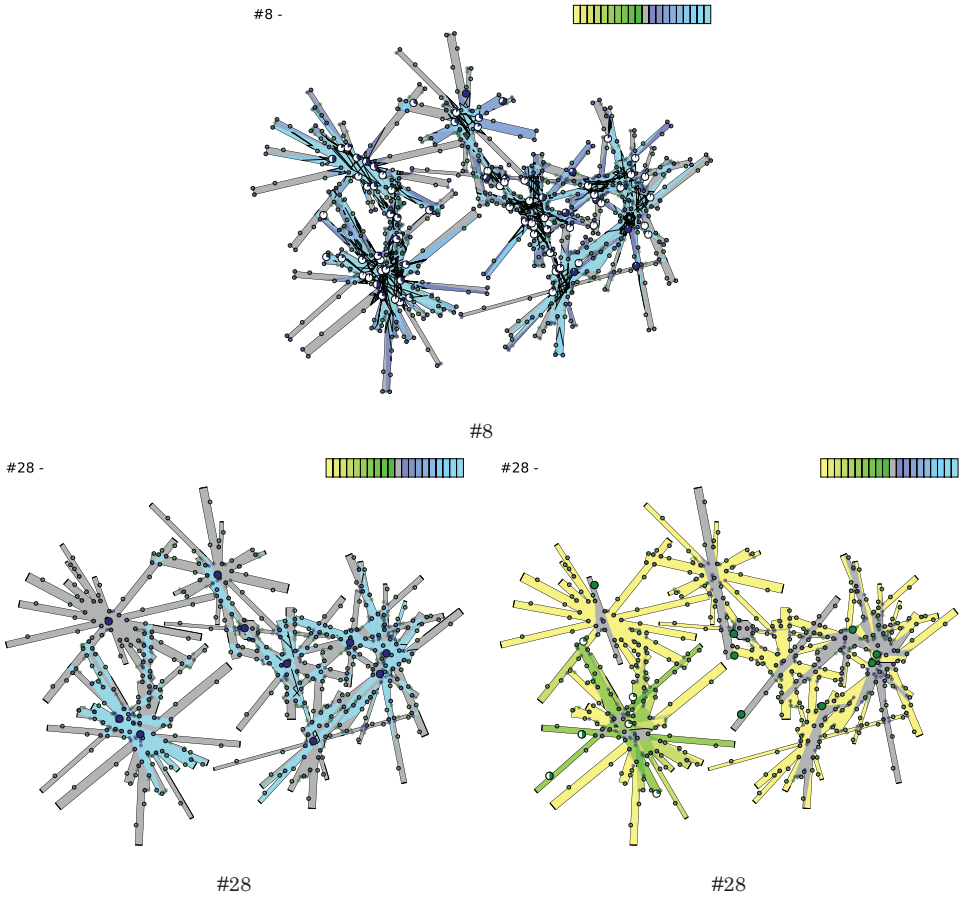


Fig. 3. Top: A lower bound of 23 on the number of vertex guards. Bottom left: A a feasible solution with 10 general guards (big blue points). Bottom right: A matching set of witnesses showing optimality of the general solution. Witness points are shown in green, and points are shown in blue. In case of a guard $g \in G$ with $x_g \geq 0$ (an active guard position), or a witness $w \in W$ with $y_w \geq 0$, the circle denoting the guard or witness is drawn larger than the circles for inactive points. The amount of filling of these circles indicates the value of the corresponding variable (a fully filled circle indicating a value of 1.)

primal problem still has a tiny region near the central column that is not sufficiently covered. Finally, in iteration #5, the primal and the dual problem are feasible, so we are in cases (b) and (d) of Section 4.1 and have achieved an optimal feasible solution with 10 general guards.

6.2. Evaluation

For a thorough evaluation of our implementation, we used four different sets of instances.

—Random orthogonal simple polygons from the art gallery instances page [Couto et al. 2009]. See Figure 6 (left) for an example.
—Random orthogonal von Koch polygons, from the art gallery instances page [Couto et al. 2009]. A von Koch polygon is a polygon with a boundary constructed by modified versions of the fractal von Koch curve starting from a square. At each level, a side is partitioned in three intervals of equal length ℓ, the center interval is then replaced by

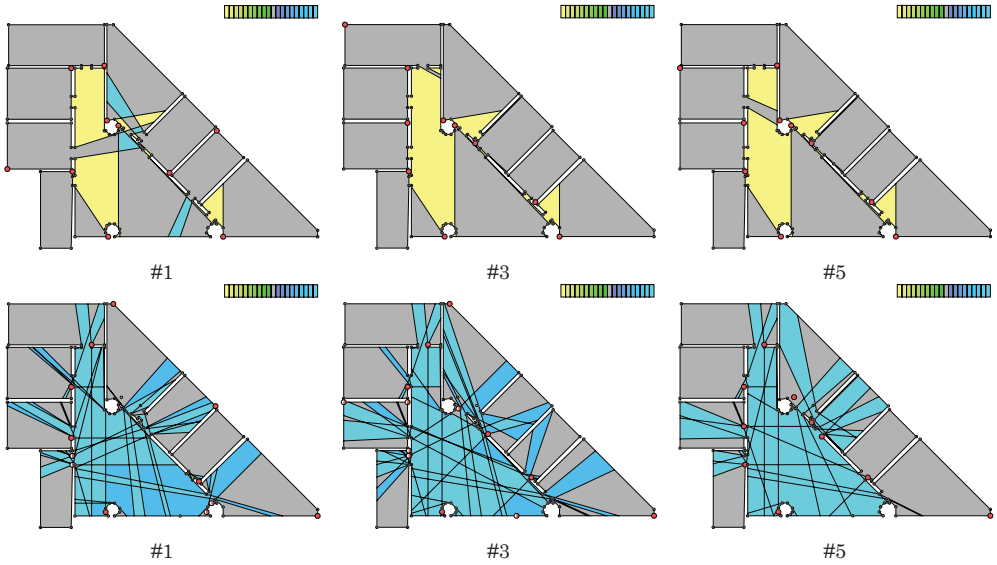


Fig. 4. Floor plan of our institute. The top row shows witness sets, while the bottom row shows guard sets for iterations #1, #3, #5. Fractional weights are indicated by partially filled red dots. An optimal solution and a matching lower bound are found in the fifth iteration.

Fig. 5. Different levels of the boundary of modified von Koch polygons: level 1 (left) and level 2 (right).

three sides. Those build a rectangle with the removed center interval: The opposite side is of equal length ℓ, and the other two sides have length $3/4\ell$. See Figure 5 for an example of the changes applied to the boundary. Then a random (orthogonal) von Koch polygon is constructed starting from a square, randomly choosing a side of the current polygon which is replaced by the modified von Koch curve (in case level 4 is reached a side will not be chosen anymore). See Figure 6 (center) for an example.

—Random nonorthogonal simple polygons, again from the art gallery instances page. See Figure 6 (right) for an example.

—Random spike polygons with holes. For those, we created star-shaped polygons consisting of a central octagon, to which we added random spikes that can be guarded from the octagonal center. Each problem instance is the overlay of several such star-shaped pieces. See Figure 3 for an example. These are constructed to evaluate the difference between vertex and point guards. Placing a point guard in each octagon center produces an almost-optimal solution, whereas there are more guards needed when they are restricted to the vertices. Furthermore, these instances have holes in order to evaluate the influence of them on the algorithmic performance.

Each of these four sets consists of 150 instances, with about 60, 100, 200, or 500 vertices.

Each instance was run with different configurations. The tests were conducted on Linux PCs with 3.0GHz CPUs and 2GB of RAM. We used CGAL 3.4, CPLEX 12.1, and SoPlex 1.4.1. In a prestudy, we found that the algorithm would usually terminate within a few seconds to a minute, or run for a very long time (hours to days). Therefore,

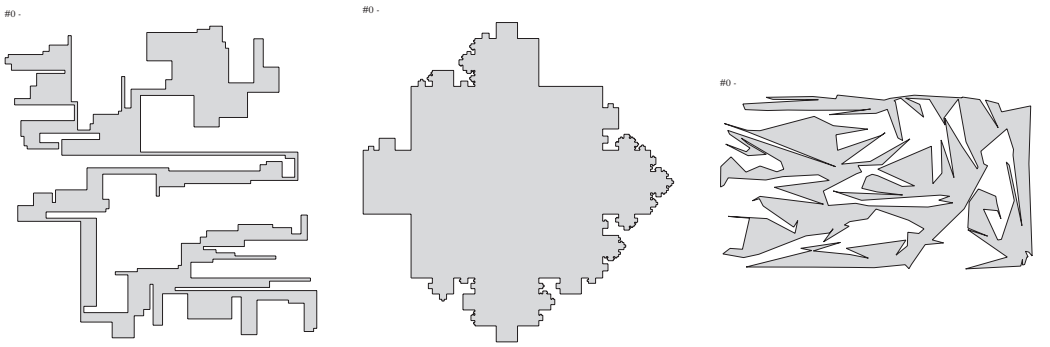


Fig. 6. Left: A random nonorthogonal simple polygon with 200 vertices. Center: A random orthogonal "von Koch" polygon with 500 vertices. Right: A random nonorthogonal simple polygon with 200 vertices.

we chose to abort each experiment after some defined time bound in the final study. This bound was set to 20 minutes of CPU time, which is sufficiently long not to abort runs that finish quickly and, at the same time, allowed us to run a total of 28,000 experiments within a reasonable time. Therefore, some runs did not finish with an optimal fractional solution. In the remainder of this section, we report on the percentages of runs that lead to optimal fractional solutions and to optimal integral solutions. Furthermore, we list how long it took to finish, on average, not counting the runs that were aborted. Note that the algorithm can be interrupted earlier if optimal solutions are not neccessary, as we produce lower and upper bounds throughout the runtime.

Most comparisons are based on a "default configuration," which we believe to be the best combination of parameters. It initially places guards on all reflex vertices, and witnesses on all reflex edges (see Section 5.1). The separation strategy is Stay_{P}, using the Edge Bulk primal separator and Greedy dual separator. The default LP solver is CPLEX.

Several of the results in this section report on the impact of certain parameter choices on the total runtime. We use only pairwise comparison, where we change a single parameter and record the change in runtime. To account for the large diversity in runtimes, we average over the quotient of these runtime pairs. Due to the multiplicative nature of such quotients, we report the log-average (geometric mean): When we compare configurations p and q over n runs, leading to runtimes $(p_i)_{i=1,\ldots,n}$ and $(q_i)_{i=1,\ldots,n}$, the average is $\exp((1/n)\sum_{i=1}^{n}\log(p_i/q_i))$. Therefore, a value of 1 indicates that both configurations perform the same, a value below 1 shows that p is faster, and a value over 1 shows that q is faster. To see why we favor the log-average over the standard average, consider the following example. Suppose we have two runs, where q takes twice as long as p on the first (i.e., $p_1/q_1 = \frac{1}{2}$), and the situation is reversed on the second (i.e., $p_2/q_2 = 2$), the average runtime factor would be $(\frac{1}{2} + 2)/2 = \frac{5}{4}$, indicating that p is slower than q. Even worse, the result depends on the order of p and q—if we exchange p and q, the average is still $\frac{5}{4}$, now indicating that p is faster than q. The log-average will be 1 in either order, correctly indicating that, on average, both configurations have the same speed.

Timing. Table I lists the average amount of time spent in the three major procedures of the algorithm. It is obvious that data structure updates and geometric support procedures are—by far—computationally most expensive, whereas solving the LPs has virtually no runtime impact. This indicates that from a software engineering standpoint, improving the geometry routines should yield the biggest overall speed-up. However,

Table I. Average Allocation of CPU Time to the Three Major Subprocedures

	Runtime	Purpose
State updates	80.02%	Compute visibility polygons, update LP
Separation	19.50%	Find new guards and witnesses
LP solving	0.29%	Solve current LP, provide fractional solution
Rest	0.19%	

Table II. Optimality Rates for Default Configuration

(a) Fractional Optimality

n	60	100	200	500
Orth.	90%	71%	44%	15%
Simple	83%	82%	74%	48%
Koch	87%	89%	93%	81%
Spikes	100%	100%	100%	100%
Total	90%	86%	78%	61%

(b) Integer Optimality

n	60	100	200	500
Orth.	80%	54%	19%	7%
Simple	80%	64%	44%	4%
Koch	77%	71%	70%	15%
Spikes	67%	68%	61%	52%
Total	76%	64%	49%	19%

Table III. Average Algorithm Runtimes in the Default Configuration

n	60	100	200	500
Orth.	0.4s	1.1s	4.3s	25.3s
Simple	0.7s	29.4s	14.9s	223.3s
Koch	2.4s	5.5s	18.9s	205.0s
Spikes	1.7s	3.9s	15.3s	190.2s
Avg.	1.3s	9.4s	14.8s	189.3s

as is made clear in further analyses in the following text, this is not the most important issue.

Optimality Rates. Table II lists the percentage of runs in which the default configuration could identify an optimal fractional solution (Table II(a), respectively), optimal integral solution (Table II(b)) within the 20-minute time frame. Clearly, these rates are very high. This is especially surprising for integral solutions, as this problem is not directly addressed by the algorithm. Table III reports the average time for completion of the algorithm, counting only those that finished in time. Note the 29.4 seconds average time for polygons with 100 vertices results from one instance with a runtime of 577 seconds. Most of the other instances are solved in 1 to 2 seconds (Figure 7, left). The polygons with 200 vertices mostly have a runtime between 4 and 8 seconds (and a "smaller" outlier instance with 147 seconds; see Figure 7, right), resulting in an average time of 14.9 seconds.

The runtimes stay well beyond the maximum of 1,200 seconds, indicating the algorithm will either terminate quickly or run into problems it cannot solve at all. Such problems usually occur when a huge number of iterations is needed to have guards or witnesses slowly converge to specific points. Section 7 discusses an example for this. Based on these tables and manual inspection of the according algorithm runs, we conclude that there is a great need for additional separators that are able to detect such situations and take appropriate action by placing guards (or witnesses) at specific points. As we are currently not aware of how to define such a separator, this is left for future work.

Steering Strategies. We evaluated the influence of different steering strategies on the algorithm runtime (Table IV). We conclude that Dual and Stay_D usually outperform the other strategies. Both start by producing additional guards, indicating that our primal initial placement strategies could be improved. Also, the poor performance of Primal and Stay_P indicate that the strategy for placing initial witnesses is very good. Furthermore, we did observe that Dual and Primal produce more bounds of

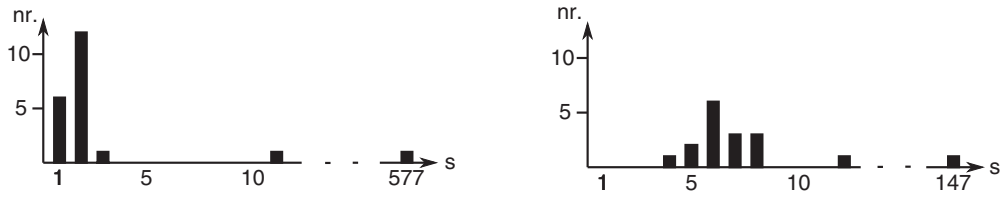


Fig. 7. Histogram of algorithm runtimes in the default configuration for random simple polygons with 100 (left) and 200 (right) vertices.

Table IV. Influence of Steering Strategy on Total Runtime. (log-average of relative time)

	Primal	Stay_P	Both	Dual	Stay_D
Primal	1.00	1.24	1.31	1.35	1.33
Stay_P	0.81	1.00	1.01	1.08	1.12
Both	0.77	0.99	1.00	1.05	1.09
Dual	0.74	0.93	0.95	1.00	1.02
Stay_D	0.75	0.89	0.92	0.98	1.00

Table V. Influence of Primal Separator on Total Runtime. (log-average of relative time)

	Max. V.	Greedy	F. Bulk	E. Bulk
Max. V.	1.00	1.16	1.27	1.62
Greedy	0.86	1.00	1.09	1.41
F. Bulk	0.78	0.92	1.00	1.30
E. Bulk	0.62	0.71	0.77	1.00

their respective kind. Stay_P and Stay_D provide alternating bounds of both kinds, as expected.

Point Separators. Comparisons for choosing primal and dual separators different from the default can be found in Tables V and VI. It is obvious that the primal separator Edge Bulk outperforms the others, which confirms the motivation of avoiding the creeping shadow effect (see Section 5.1). For dual separators, the Greedy strategy is the clear winner, although the numbers suggest that it could be worth evaluating additional separators in future work.

Initial Placement. Next we evaluated the influence of the initial choice of guard and witness points. This is reported in Table VII. We did not run every possible combination of guard/witness initialization strategies, as that would require 25 times as many runs. Instead we selected five seemingly useful combinations and evaluated those.

It is obvious that a good choice of initial points can have a substantial improvement in runtime, given that the best and worst strategies are, on average, by a factor 2 apart. The combination of reflex vertices and reflex edges is best, substantially outperforming the other combinations. This confirms the intuition behind these strategies—placing guards where they can see as much as possible and hiding witnesses at the boundaries. An interesting conclusion can be drawn from these results together with the analysis of separation strategies described earlier. Strategies that initially produce guards turned out to be beneficial. This indicates that the witnesses are well chosen, whereas a set of guards in the polygon's interior is needed to speed up the algorithm. Identifying a useful strategy following this scheme is left for future work.

Influence of LP Solver. As described earlier, solving LPs consumes almost no CPU time during the course of the algorithm, so switching to a different solver should not have a direct influence on the outcome. However, we found that using SoPlex instead of

Table VI. Influence of Dual Separator on Total Runtime. (log-average of relative time)

	Max. V.	V. Bulk	Greedy
Max. V.	1.00	1.17	1.41
V. Bulk	0.86	1.00	1.17
Greedy	0.71	0.86	1.00

Table VII. Impact of Initial G and W on Total Runtime. (log-average of relative time)

	One/One	Oth/Oth	All/All	All/RfE	RfV/RfE
One/One	1.00	1.59	1.64	1.74	2.02
Other/Other	0.63	1.00	1.04	1.13	1.30
All/All	0.61	0.96	1.00	1.10	1.27
All/Reflex E.	0.57	0.88	0.91	1.00	1.21
Reflex V./Reflex E.	0.50	0.77	0.79	0.82	1.00

Table VIII. Integer Solutions Obtained Using SoPlex

n	60	100	200	500
Orth.	86%	61%	15%	7%
Simple	79%	64%	33%	0%
Koch	71%	57%	67%	15%
Spikes	64%	46%	46%	52%
Total	75%	57%	40%	19%

CPLEX resulted in 17% longer solution times, on average. This must be due to indirect effects. Further evaluations revealed that CPLEX tends to produce fractional solutions with a smaller support (i.e., number of nonzero variables), which decreases the time spent to compute the associated primal and dual arrangements. We believe this is due to superior preprocessing in CPLEX, which leads to combinatorial fixation of some variables.

Neither of the two LP solvers produces integer solutions more frequently than the other. Table VIII lists how often SoPlex-based configuration could identify optimal integer solutions; compare with Table II(b) for CPLEX. The lower percentage of 57% of Random orthogonal von Koch polygons with 100 vertices in comparison to 67% of Random orthogonal von Koch polygons with 200 vertices is caused by stochastic effects.

Vertex Guards. Using the Primal Only steering strategy lets the algorithm solve the vertex guard variant of the problem, in order to allow for a comparison with the algorithm by Couto et al. [2009]. Note that there are two important differences: Our algorithm can deal with polygons with holes; on the other hand, their's always produces integral solutions. As it was not possible to repeat the reported experiments on identical platforms, a direct comparison of runtimes is impossible.

Table IX shows how often the algorithm finished with a fractional or integer optimal solution. It is striking how the restriction to vertex guards lets convergence problems disappear—all runs finished within time (compare to Table II(a) for the general case). The solution times are reported in Table X. Obviously, the vertex-guard problem can be solved faster by our algorithm, and there are no convergence issues. However, it should also be noticed that optimal integer solutions are found less frequently. Given that there is no direct mechanism for integer solutions in our algorithm, addressing this issue is left for future work.

Two conclusions can be drawn. First, our algorithm has a speed well comparable to that by Couto et al. [2009]. Second, as success rates increase and runtimes drop in this variant, we can see that the apparant loss of performance in the general case

Table IX. Optimality Rates for Vertex Guards

n	60	100	200	500
Orth.	100%	100%	100%	100%
Simple	100%	100%	100%	100%
Koch	100%	100%	100%	100%
Spikes	100%	100%	100%	100%
Total	100%	100%	100%	100%

(a) Fractional Optimality

n	60	100	200	500
Orth.	79%	71%	22%	11%
Simple	64%	64%	22%	7%
Koch	32%	36%	19%	4%
Spikes	0%	25%	4%	0%
Total	44%	49%	17%	6%

(b) Integer Optimality

Table X. Total Runtimes Solving the Vertex-Guard Variant. (Last row taken from Table III for comparison with the general problem.)

n	60	100	200	500
Orth.	0.4s	1.0s	3.8s	23.2s
Simple	0.7s	1.5s	5.4s	32.4s
Koch	0.6s	1.6s	6.4s	43.1s
Spikes	1.5s	2.9s	11.8s	136.4s
Avg.	0.8s	1.8s	6.9s	58.8s
P.Guards	1.3s	9.4s	14.8s	189.3s

stems from the increased complexity of the problem, and not from implementation or fundamental algorithmic issues.

7. PROBLEMS OF CONVERGENCE

For some polygons, some guards in an optimal solution for the AGP must be located at certain points or on a line segment (Figure 8). This means that the measure of the solution in an appropriate high-dimensional space is zero; as there is no known simple description of these sets (in particular, they are not intersections of diagonals or edge extensions), it is not surprising that dealing with these situations is problematic.

Our method is based on introducing guards (or witnesses) in insufficiently covered areas. We introduce a witness $w \in W$ in case its constraint is violated (i.e., $x(G \cap \mathcal{V}(w)) < 1$). Hence, looking at the overlay of the visibility regions of the guards, this coverage value is smaller on faces of the arrangement than on its edges or vertices. Consequently, we will mostly find witnesses with violated constraints in those faces. Accordingly, as we look for a guard $g \in G$ with $y(W \cap \mathcal{V}(g)) > 1$, guards will be introduced most likely (the inequality may be violated on vertices and not on faces, but not vice versa) on vertices of the visibility region overlay defined by the witnesses.

We are able to identify a region around the guard's only possible position and shrink its size during the course of the algorithm. In order to obtain the optimal solution, the newly chosen guard needs to be located exactly at a certain position, which need not be describable by some simple characteristics of the polygon, such as edge extensions or diagonals, but may be located, such as the center guard of the example of Figure 8, on the intersection of visibility lines. The region will shrink to a point after infinitely many iterations, but not within finite time. In order to obtain a finite time, we would have to be able to find the unknown point after a limited number of iterations. As our point separating strategies can only depend on the current overlay and the polygon, but not on an intersection defined by the final set of guards, we will often fail to "guess" exactly this point. First, in the primal separation, we need to identify witnesses in faces that then define an arrangement for the dual separation with a vertex exactly at the desired position. For the witnesses in faces, we have an infinite number of positions to choose from.

Using a different strategy for the initial guard placement, we may obtain convergence toward optimal fractional solutions, but this is not guaranteed. Obviously, this issue

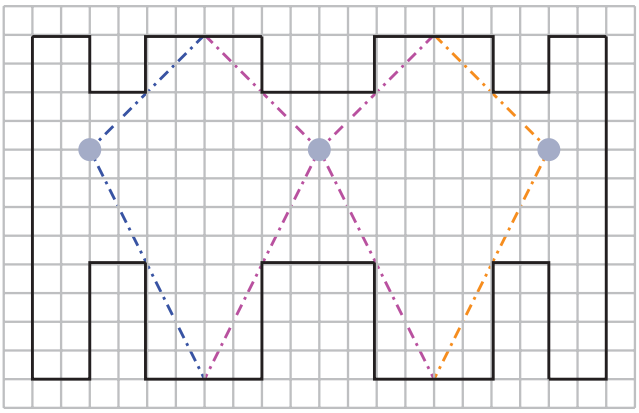


Fig. 8. For this polygon guards cannot be moved in any direction. The dash-dotted lines indicate lines of sight of the guards.

should become less problematic if the found solutions are also required to be robust, implying that small perturbations leave their feasibility intact, eliminating degenerate solutions. This requirement is quite natural in practice, not only for being able to implement optimal positions, but also because scan registration usually requires some overlap between visibility regions.

For the polygon in Figure 8 and only one initial guard, we have to deal with these problems of convergence (Figure 9). In the solution to the primal problem after 407 iterations (see Figure 9, top), we can identify many guard positions inserted "around" the only feasible locations of the optimal solution, but still the primal and the dual problem are not feasible. On the other hand, for the same polygon, we can find the optimal solution value, based on an optimal fractional solution, if we start with guards placed at every vertex.

8. GENERATING INTEGER SOLUTIONS

The AGP asks for integer solutions. Hence, improving their generation is of interest. Here, we briefly discuss some basic ideas. The development of strategies to obtain integer solutions is in the focus of our current and future work.

An improvement should come from combining geometry with polyhedral combinatorics, refining the polyhedral description in order to remove large classes of nonintegral vertices.

The idea is to add inequalities that are intrinsic to our geometrical problem, but not given for the general covering problem. The cuts we consider are geometric cuts, based on sets of witness points.

As shown in Figure 10, we have identified various classes of cutting planes that may turn out to be useful for quicker resolution of fractional vertices, at the expense of solving the corresponding separation problems (which are different from what we described in Section 4.) Those cutting planes are based on a common idea: If we can identify three $(k+1)$, respectively witnesses, such that there exists no point in the polygon that can guard more than two of these, we need at least two $(\frac{k+1}{2})$, respectively guards for complete coverage. In the example of Figure 10(a), we have exactly three witnesses $(w_1, \ldots, w_3)$ fulfilling this condition, while the fractional optimal solution places three guards, each with a value of $\frac{1}{2}$, resulting in a objective value of $\frac{3}{2}$. The

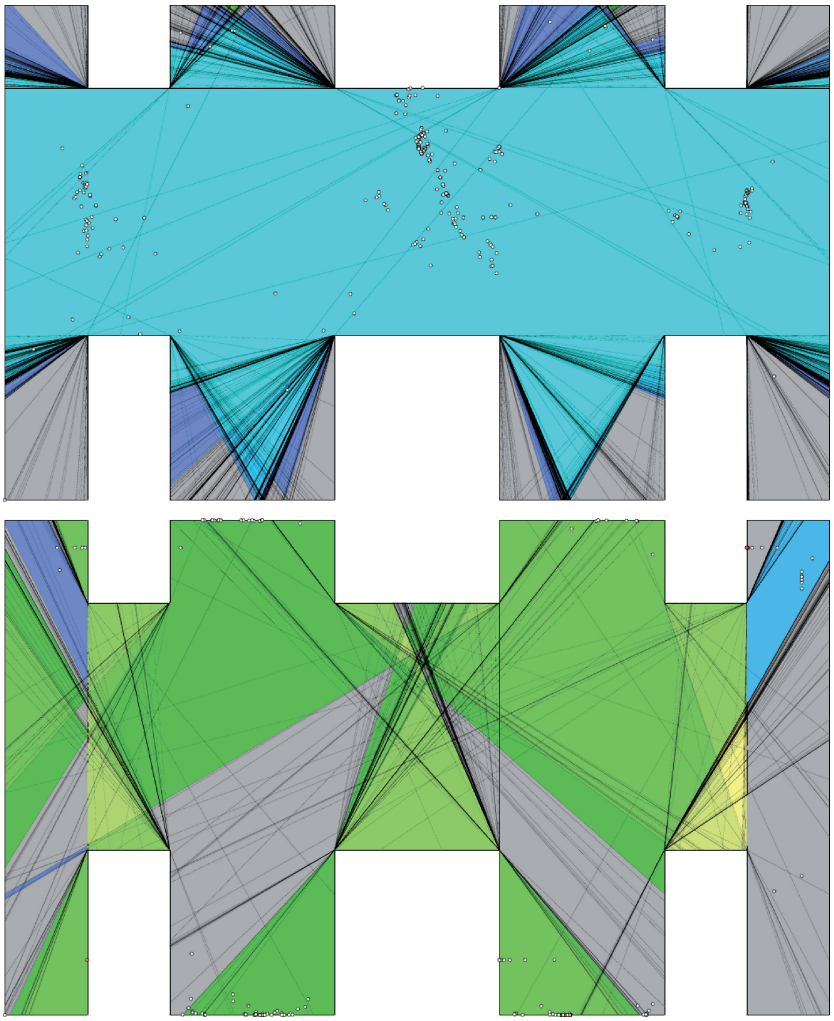


Fig. 9. Worst-case problem instance with zero-dimensional optimal solution space. Top: The primal solution after over 400 iterations. (Gray denotes covering with a value of exactly 1, solutions with bluish coloring are feasible (value >1), solutions with yellow-green coloring (<1) are not.) Bottom: The corresponding dual solution. (Gray denotes covering with a value of exactly 1, solution with yellow-green coloring are feasible (value <1), solutions with bluish coloring (>1) are not.)

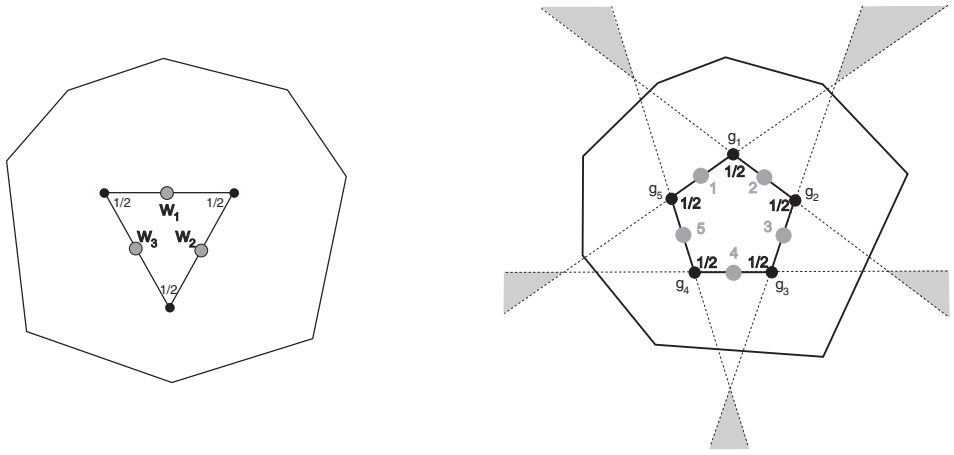
cutting plane we introduce is:

$$\sum_{g \in \cup_{i=1}^{3} \mathcal{V}(w_i)} x_g \geq 2. \tag{10}$$

In general, we identify cutting planes of this type:

$$\exists w_1, \ldots, w_{k+1} : \forall i \neq j \neq \ell \in \{1, \ldots, k+1\} : \quad \mathcal{V}(w_i) \cap \mathcal{V}(w_j) \cap \mathcal{V}(w_\ell) = \emptyset$$
$$\Rightarrow \quad \sum_{g \in \cup_{i=1}^{k+1} \mathcal{V}(w_i)} x_g \geq \tfrac{k+1}{2}. \tag{11}$$

Those cutting planes are not yet integrated into the implementation. Solving the resulting separation problems and identifying further cutting planes and the implementation are left for future work.



(a) (b)

Fig. 10. Examples for cutting planes. Left: An example with a fractional optimal value of $\frac{3}{2}$, with three guards of value $\frac{1}{2}$ each. Using the witnesses $w_1, \ldots, w_3$, we can introduce a new cutting plane. Right: An example with a fractional optimal value of $\frac{5}{2}$, using five guards of value $\frac{1}{2}$ each. The witnesses $w_1, \ldots, w_5$ allow us to bound the number of guards with three from below. The shaded areas indicate "forbidden" areas for the polygon's interior, as points located in these regions can guard more than two of the witnesses considered in this example.

9. CONCLUSION AND OUTLOOK

We have presented an LP-based procedure for obtaining bounds for the AGP that—in case of convergence and integrality—provides optimal solutions. Our algorithm is based on a formulation as a fractional covering problem and the corresponding dual fractional packing problem. For the separation problem, we use properties of visibility polygons, that is, information that is intrinsic to the AGP but not to the underlying covering formulation.

There is a variety of directions for extending our work. As discussed earlier, improving the separation routine may lead to considerable speed-up; however, this may come at the expense of tremendous memory requirements, so careful balancing will be necessary. As presented in Section 6, the updates of the overlay cause around 80% of the runtime; more sophisticated primal and dual point separators should increase the success rate. Consequently, a main focus of our future work will be on the development of separators.

Other extensions arise from fine-tuning the choice of guards and witnesses, making more extensive use of the intrinsic geometric structure of the underlying set cover instances.

Finally, discussions with our inmetris3D partners have revealed a variety of practical conditions and relaxations on guards and environment. In particular, there is a strong interest in fractional solutions instead of integral ones, making our LP-based approach even more powerful. We are optimistic that we can report on these and other improvements in the foreseeable future.

REFERENCES

AMIT, Y., MITCHELL, J. S. B., AND PACKER, E. 2007. Locating guards for visibility coverage of polygons. In *Proceedings of the Workshop on Algorithm Engineering and Experiments (ALENEX '07)*. SIAM, 120–134.

Aronov, B., Guibas, L. J., Teichmann, M., and Zhang, L. 2002. Visibility queries and maintenance in simple polygons. *Discrete Comput. Geom. 27,* 4, 461–483.

Avis, D. and Toussaint, G. 1981. An efficient algorithm for decomposing a polygon into star-shaped pieces. *Pattern Recog. 13*, 395–398.

Bottino, A. and Laurentini, A. 2008. A nearly optimal sensor placement algorithm for boundary coverage. *Pattern Recog. 41,* 11, 3343–3355.

CGAL. Computational Geometry Algorithms Library. http://www.cgal.org.

Chin, W.-P. and Ntafos, S. 1986. Optimum watchman routes. In *Proceedings on the 2nd Annual ACM Symposium on Computational Geometry*. ACM, 39–44.

Chin, W.-P. and Ntafos, S. C. 1991. Shortest watchman routes in simple polygons. *Discrete Comput. Geom. 6*, 9–31.

Chvátal, V. 1975. A combinatorial theorem in plane geometry. *J. Comb. Theor. Ser. B 18,* 1, 39 – 41.

Couto, M. C., de Rezende, P. J., and de Souza, C. C. 2009. An exact algorithm for an art gallery problem. Tech. rep. IC-09-46, Institute of Computing, University of Campinas.

Couto, M. C., de Rezende, P. J., and de Souza, C. C. 2009. Instances for the art gallery problem. http://www.ic.unicamp.br/∼cid/Problem-instances/Art-Gallery.

Couto, M. C., de Souza, C. C., and de Rezende, P. J. 2008. Experimental evaluation of an exact algorithm for the orthogonal art gallery problem. In *Proceedings of the 7th International Conference on Experimental Algorithms (WEA'08)*. Springer-Verlag, Berlin, 101–113.

Couto, M. C., Souza, C. C. d., and Rezende, P. J. d. 2007. An exact and efficient algorithm for the orthogonal art gallery problem. In *Proceedings of the XX Brazilian Symposium on Computer Graphics and Image Processing (SIBGRAPI '07)*. IEEE Computer Society, Los Alamitos, CA, 87–94.

de Berg, M., Cheong, O., van Kreveld, M., and Overmars, M. 2008. *Computational Geometry: Algorithms and Applications*. Springer.

Efrat, A. and Har-Peled, S. 2006. Guarding galleries and terrains. *Inf. Process. Lett. 100,* 6, 238–245.

Eidenbenz, S., Stamm, C., and Widmayer, P. 2001. Inapproximability results for guarding polygons and terrains. *Algorithmica 31,* 1, 79–113.

Fisk, S. 1978. A short proof of Chvátal's watchman theorem. *J. Comb. Theor. Ser. B 24,* 3, 374–375.

Grötschel, M., Lovász, L., and Schrijver, A. 1981. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica 1*, 169–197.

Heffernan, P. J. and Mitchell, J. S. B. 1995. An optimal algorithm for computing visibility in the plane. *SIAM J. Comput. 24,* 1, 184–201.

Hershberger, J. and Suri, S. 1995. A pedestrian approach to ray shooting: shoot a ray, take a walk. *J. Algor. 18,* 3, 403–431.

Inkulu, R. and Kapoor, S. 2009. Visibility queries in a polygonal region. *Comput. Geom. 42,* 9, 852–864.

Jansen, K. and Margraf, M. 2008. *Approximative Algorithmen und Nichtapproximierbarkeit*. de Gruyter.

Kahn, J., Klawe, M., and Kleitman, D. 1983. Traditional art galleries require fewer watchmen. *SIAM J. Algebraic Discrete Methods 4,* 2, 194–206.

Keil, J. M. 1986. Minimally covering a horizontally convex orthogonal polygon. In *Proceedings of the 2nd Annual Symposium on Computational Geometry (SoCG'86)*. ACM, 43–51.

Laurentini, A. 1999. Guarding the walls of an art gallery. *Visual Comput. 15,* 6, 265–278.

Lee, D.-T. and Lin, A. K. 1986. Computational complexity of art gallery problems. *IEEE Trans. Inf. Theory 32,* 2, 276–282.

O'Rourke, J. 1987. *Art Gallery Theorems and Algorithms*. International Series of Monographs on Computer Science. Oxford University Press, Oxford, UK.

O'Rourke, J. and Supowit, K. 1983. Some NP-hard polygon decomposition problems. *IEEE Trans. Info. Theory 29,* 2, 181–190.

Pocchiola, M. and Vegter, G. 1993. The visibility complex. In *Proceedings of the 9th Annual Symposium on Computational Geometry (SoCG'93)*. ACM, New York, 328–337.

Schrijver, A. 1986. *Theory of Linear and Integer Programming*. John Wiley & Sons, Chichester.

Schuchardt, D. and Hecker, H.-D. 1995. Two NP-hard art-gallery problems for ortho-polygons. *Math. Logic Quart. 41*, 261–267.

Shermer, T. C. 1992. Recent results in art galleries. *Proc. IEEE 80,* 9, 1384–1399.

Suri, S. and O'Rourke, J. 1986. Worst-case optimal algorithms for constructing visibility polygons with holes. In *Proceedings of the 2nd Annual Symposium on Computational Geometry (SoCG'86)*. ACM, New York, 14–23.

URRUTIA, J. 2000. Art gallery and illumination problems. In *Handbook on Computational Geometry*, J.-R. Sack and J. Urrutia, Eds. Elsevier Science Publishers, 973–1026.

WEIN, R., FOGEL, E., ZUKERMAN, B., AND HALPERIN, D. 2008. 2D arrangements. In *CGAL User and Reference Manual* 3.4 Ed., C. E. Board, Ed.

WUNDERLING, R. 1996. Paralleler und objektorientierter Simplex-Algorithmus. Ph.D. thesis, Technische Universität Berlin. http://www.zib.de/Publications/abstracts/TR-96-09/.

ZAREI, A. AND GHODSI, M. 2005. Efficient computation of query point visibility in polygons with holes. In *Proceedings of the 21st Annual Symposium on Computational Geometry (SoCG'05)*. ACM, New York, 314–320.

Received April 2010; revised November 2011; accepted February 2012