

Searching Polyhedra by Rotating Half-Planes

GIOVANNI VIGLIETTA
Department of Computer Science
University of Pisa
viglietta@gmail.com

April 29, 2012

Abstract

The SEARCHLIGHT SCHEDULING PROBLEM was first studied in 2-dimensional polygons, where the goal is for point guards in fixed positions to rotate searchlights to catch an evasive intruder. Here the problem is extended to 3-dimensional polyhedra, with the guards now boundary segments who rotate half-planes of illumination.

After carefully detailing the 3-dimensional model, several results are established. The first is a nearly direct extension of the planar one-way sweep strategy using what we call *filling* guards, a generalization that succeeds despite there being no well-defined notion in 3-dimensional space of planar “clockwise rotation.” Next follow two results: every polyhedron with $r > 0$ reflex edges can be searched by at most r^2 suitably placed boundary guards, whereas just r edguards suffice if the polyhedron is orthogonal. (Minimizing the number of guards to search a given polyhedron is easily seen to be NP-hard.) Finally we show that deciding whether a given set of boundary guards has a successful search schedule is strongly NP-hard. A number of peripheral results are proved en route to these central theorems, and several open problems remain for future work.

1 Introduction

Searchlight scheduling

The SEARCHLIGHT SCHEDULING PROBLEM (SSP) was first studied by Sugihara et al. as a search problem in simple polygons, where some stationary

guards are tasked to locate an evasive, moving intruder by hitting him with searchlights.[1] Each guard carries a searchlight, modeled as a 1-dimensional ray that can be continuously rotated, while the intruder runs unpredictably and with unbounded speed, trying to avoid the searchlights. Since the guards cannot know the position of the intruder until they catch him in their lights, the movements of the searchlights must follow a fixed *schedule*, which should guarantee that the intruder is caught in finite time, regardless of the path he decides to take. The search takes place in a polygonal region, whose sides act as obstacles both for the intruder's movements and for the guards' searchlights. In a way, the polygonal boundary benefits the intruder, who can hide behind corners and avoid scanning searchlights. But it can also turn into a *cul-de-sac*, if the guards manage to force the intruder into an enclosed area from which he cannot escape.

Thus SSP is the problem of deciding if there exists a successful search schedule for a given finite set of guards in a given simple polygon. Figure 1 shows an instance of SSP with a successful schedule.

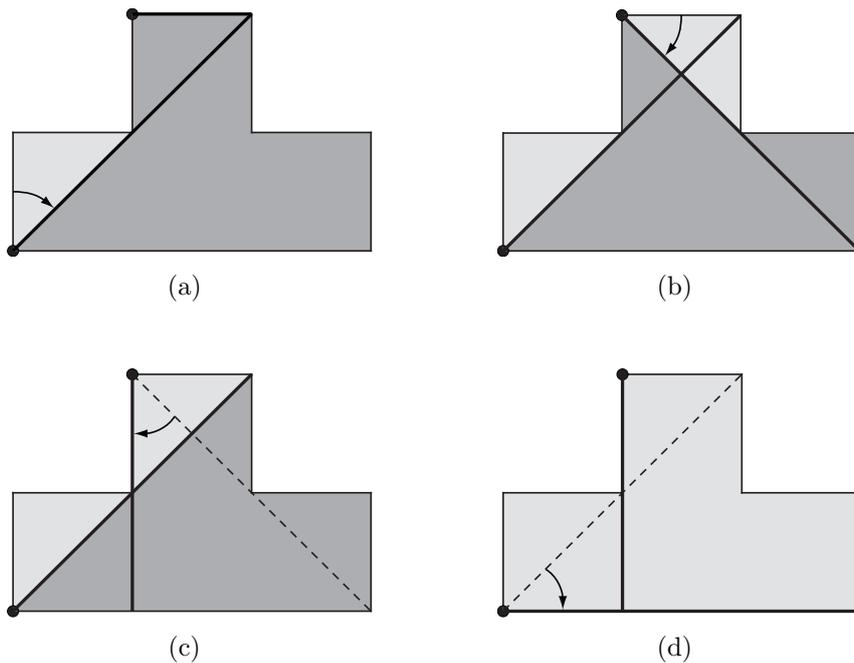


Figure 1: Search schedule for two guards in a polygon. At each stage, the dark area is still “contaminated,” the lighter areas have been cleared.

Previous work

We will now briefly review some well-known results pertaining SSP in 2-dimensional polygons, before summarizing our results for the 3-dimensional model.

A trivial necessary condition for searchability is that the guard positions should guarantee that no point in the polygon is invisible to all guards. In other words, the guards should at least solve the ART GALLERY PROBLEM in the given polygon (see Ref. 2), otherwise the intruder could sit at an uncovered point and never be discovered.

Another simple necessary condition is that every guard lying in the interior of the polygon (thus not on the boundary) should be visible to at least one other guard. Without this, the intruder could remain in a neighborhood of a guard and just avoid its rotating searchlight.

Several sufficient conditions for searchability were detailed by Sugihara et al., all employing a general search algorithm called the *one-way sweep strategy*. Most notably, if all the guards lie on a simple polygon's boundary and collectively see its whole interior, then they also have a successful search schedule.[1]

Concerning the problem of minimizing the number of guards to search a given polygon, Sugihara et al. gave a characterization of the simple polygons that are searchable by one or two suitably placed guards.[1] A similar characterization for three guards was also found by the same authors, but never published.[3] On the other hand, Ref. 4 contains some upper bounds on the minimum number of guards required to search a polygon (possibly with holes) as a function of the number of guards needed to solve the ART GALLERY PROBLEM in the same polygon.

The problem of determining the computational complexity of SSP was not directly addressed by Sugihara et al. in their seminal papers, but has acquired more interest over time, and remains open. Obermeyer et al. showed that SSP is solvable in double exponential time, by means of a cell decomposition technique that reduces the space of all possible searchlight schedules to a finite graph, which is then searched systematically.[5] It is also straightforward to prove that SSP belongs to PSPACE=NPSPACE, because the information contained in a node can be stored efficiently, and the graph can be searched nondeterministically.[6] It is still unknown whether SSP is NP-hard or even in NP.

However, in Ref. 6, the author introduced a generalized problem in which the guards have to clear only a given subregion of the polygon, while the rest may remain contaminated. This PARTIAL SEARCHLIGHT SCHEDULING PROBLEM was shown to be PSPACE-complete, even for orthogonal poly-

gons.

In Ref. 7, the author (with Monge) extended the basic searchlight model to 3-dimensional polyhedra, as opposed to polygons. The traditional point guards then became segment guards, casting half-planes (as opposed to 1-dimensional rays), which can rotate with one degree of freedom. After providing some geometric motivations for the choice of the model and pointing out some of its basic features, the authors considered the optimization problem of searching a polyhedron in the shortest time, and proved it to be strongly NP-hard.

Our contribution

In this paper we further develop the theory of searching polyhedra by rotating half-planes: we expand on the author's results contained in Ref. 7 and we prove several new theorems.

In Section 2 we give a careful and thorough definition of the model, which Ref. 7 lacked. Notably, we introduce the concept of *filling* guard (Definition 11), which is central for later theorems. (Subsequently, in Proposition 5, we also sketch an efficient algorithm determine if a guard is filling.)

In Section 3 we make preliminary observations and discuss the possibility of generalizing the main features of SSP to our model. Focusing on problem instances with only filling guards, in Theorem 2 we show how the *one-way sweep strategy* of Sugihara et al. (see Ref. 1) can be generalized to polyhedra, also obtaining an analogue of the main result of Obermeyer et al. (see Ref. 5) in Corollary 3. In addition, in Theorem 4 we characterize the searchable problem instances containing only one guard.

Section 4 is devoted to algorithms to place guards on a given polyhedron's boundary guaranteeing searchability, both for orthogonal and for general polyhedra. In Theorem 7 we prove that every polyhedron with $r > 0$ reflex edges can be searched by at most r^2 suitably placed guards. Theorem 8 shows how this bound can be lowered to r if the polyhedron is orthogonal, which solves Conjecture 1 in Ref. 7 (i.e., that any polyhedron with a guard on each reflex edge is searchable), in the special case of orthogonal polyhedra. In Theorem 6 and Corollary 9, we also show that search time can be drastically reduced to a small constant by adding r guards to the previous constructions.

In Section 5 we prove the strong NP-hardness of deciding if a polyhedron is searchable by a given set of boundary guards (Theorem 10), which greatly improves on the main result of Ref. 7 (i.e., that minimizing search time is NP-hard). Indeed, being able to minimize the search time, where infinite search time means unsearchable, allows *a fortiori* to decide searchability.

Finally, Section 6 contains concluding remarks and suggestions for further research.

2 Model Definition

Polyhedra

For our purposes, a *polyhedron* will be the union of a finite set of closed tetrahedra (with mutually disjoint interiors) embedded in \mathbb{R}^3 , whose boundary is a connected 2-manifold. As a consequence, a polyhedron is a compact topological space and its boundary is homeomorphic to a sphere or a g -torus. g is also called the *genus* of the polyhedron, and it is zero if and only if the polyhedron is homeomorphic to a ball. Moreover, the complement of a polyhedron with respect to \mathbb{R}^3 is connected. Since a polyhedron's boundary is piecewise linear, the notion of *face* of a polyhedron is well-defined as a maximal planar subset of its boundary with connected and non-empty relative interior. Thus a face is a plane polygon, possibly with holes, and possibly with some degeneracies, such as hole boundaries touching each other at a single vertex. Any vertex of a face is also considered a *vertex* of the polyhedron. *Edges* are defined as minimal non-degenerate straight line segments shared by two faces and connecting two vertices of the polyhedron. Since a polyhedron's boundary is an orientable 2-manifold, the relative interior of an edge lies on the boundary of exactly two faces, thus determining an internal dihedral angle (with respect to the polyhedron). An edge is *reflex* if its internal dihedral angle is reflex, i.e., strictly greater than π . Hence, convex polyhedra have no reflex edges. A polyhedron is said to be *orthogonal* if each one of its edges is parallel to some axis.

Visibility with respect to a polyhedron \mathcal{P} is a symmetric relation between points in \mathbb{R}^3 : point x *sees* point y (equivalently, y is *visible* to x) if the straight line segment joining x with y lies entirely in \mathcal{P} . Recall that \mathcal{P} is a closed set, therefore such a segment could touch \mathcal{P} 's boundary, or even lie on it. When \mathcal{P} is understood, we can safely omit any explicit reference to it.

Guards and searchplanes

Now we can state the problem-specific definitions.

Definition 1 (Guard, boundary guard, edge guard). *A guard in a polyhedron \mathcal{P} is a positive-length straight line segment without its endpoints, lying in \mathcal{P} . A boundary guard is a guard that lies entirely on \mathcal{P} 's boundary. An edge guard is a guard that coincides with the relative interior of an edge of \mathcal{P} .*

Throughout the paper, we will be concerned mainly with boundary guards. Unlike the situation in planar SSP, boundary guards already yield a rich and diverse theory.

In Definition 1, we exclude endpoints because we do not want guards to see beyond reflex edges or non-convex vertices, as the next definitions will clarify.

Definition 2 (Visibility region). *The visibility region $\mathcal{V}(\ell)$ of a guard ℓ in a polyhedron \mathcal{P} is the set of points in \mathcal{P} that are visible to at least one point in ℓ .*

Definition 3 (Searchplane). *A searchplane of a guard ℓ is the intersection between $\mathcal{V}(\ell)$ and any (topologically closed) half-plane whose bounding line contains ℓ .*

Consequently, every guard has a searchplane for every possible direction of the half-plane generating it, and the union of a guard’s searchplanes coincides with its visibility region.

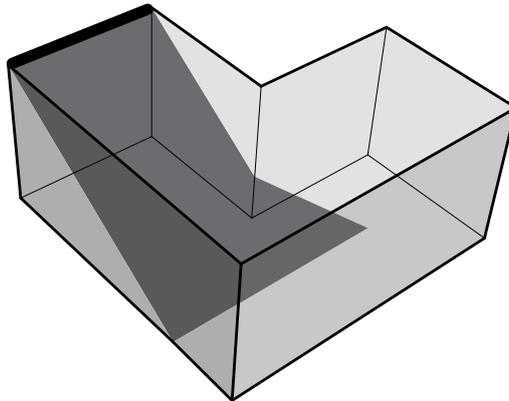


Figure 2: Edge guard with one of its searchplanes, depicted as a thick line and a dark surface, respectively.

Remark 1. *Throughout the paper, we will always use the term “searchplane” with the meaning of Definition 3, i.e., a static planar area visible to some guard. Nonetheless, we will keep using the term “searchlight” informally, referring to the tool that guards use to scan the environment. Thus, guards can turn searchlights around, but cannot turn searchplanes. While a searchplane is a well-defined mathematical object, a searchlight is not, in our terminology (in contrast with the standard terminology of 2-dimensional SSP).*

Blind directions

If a searchplane is just a line segment, it is said to be *trivial*, and the corresponding direction is said to be *blind* for its guard. A guard has blind directions if and only if it is a boundary guard. We arbitrarily define a *left* and *right* side for each boundary guard, and we call *leftmost position* the leftmost non-blind direction, for each boundary guard. Similarly, we define the *rightmost position* of every boundary guard. Observe that the leftmost and rightmost positions are well-defined, because the polyhedron is a closed set, and every direction aiming straight at its exterior is blind for a boundary guard, even if the endpoints of (the topological closure of) the guard lie on reflex edges or vertices. This is because we did not include endpoints in Definition 1, a choice motivated also by Theorem 4. Conversely, every other direction is not blind, since the corresponding searchplanes must contain either some internal points of the polyhedron, or some points in the relative interior of a face.

Search schedules

Definition 4 (Schedule). A schedule for a guard ℓ is a continuous function $f_\ell : [0, T] \rightarrow S^1$, where $T \in \mathbb{R}^+$ and S^1 is the unit circle.

Intuitively, $f_\ell(t)$ expresses the orientation of the guard at time $t \in [0, T]$, which is the angle at which ℓ is aiming its searchlight. In other words, ℓ is able to emit a half-plane of light in any desired direction, and to rotate it continuously about the axis defined by ℓ itself. We will say that, at time t , ℓ is *aiming its searchlight* at point x if the orientation expressed by $f_\ell(t)$ corresponds to a searchplane of ℓ containing x (assuming that one exists).

For the following definitions, we stipulate that a polyhedron \mathcal{P} is given, along with a finite *multiset* of guards, each of which is provided with a schedule.

Remark 2. *The reason why we use multisets rather than sets is that some guards may be coincident, yet distinct. This is the same as providing guards with unique identifiers, and also naturally models an analogous of the k -searcher of Ref. 4, i.e., a guard carrying k independent searchlights. However, we will need multisets of guards only to prove Theorem 6 and Corollary 9, which are of independent interest and may be safely disregarded without invalidating any other result in the paper.*

Definition 5 (Illuminated point). A point is illuminated at a given time if some guard is aiming its searchlight at it.

Definition 6 (Contaminated point, clear point). A point x is contaminated at time t , with respect to a given schedule, if there exists a continuous function $h : [0, t] \rightarrow \mathcal{P}$ such that $h(t) = x$ and there is no time $t' \in [0, t]$ at which $h(t')$ is illuminated. A point that is not contaminated is said to be clear.

It follows that, at any time in a schedule, a maximal connected region of \mathcal{P} without illuminated points is either all clear or all contaminated.

Definition 7 (Search schedule). A set of schedules of the form $f_\ell : [0, T] \rightarrow S^1$, where ℓ ranges over a finite guard multiset in a polyhedron \mathcal{P} , is a search schedule if every point in \mathcal{P} is clear at time T .

Next we define the 3-DIMENSIONAL SEARCHLIGHT SCHEDULING PROBLEM (3SSP).

Definition 8 (3SSP). 3SSP is the problem of deciding if a given multiset of guards in a given polyhedron has a search schedule.

An instance of 3SSP is said to be *searchable* or *unsearchable*, depending on the existence of a search schedule for its guards.

It is obvious, from these definitions, that 3SSP is not easier than SSP.

Proposition 1. $\text{SSP} \leq_L \text{3SSP}$.

Proof. Any polygon can be extruded to a prism, while each point guard can be transformed into a segment guard by stretching it parallel to the prism's sides. \square

Notice, though, that not all reasonable models of 3-dimensional guards would yield such an immediate reduction. See Ref. 7 for a brief discussion.

Since an instance of 3SSP is trivially unsearchable if its guards cannot see the whole polyhedron, it is worth singling out those instances.

Definition 9 (Guard-visible instance). An instance of 3SSP is guard-visible if every point of the polyhedron belongs to the visibility region of at least one guard.

Filling guards

Finally, a relevant role is played by a special type of (boundary) guard.

Definition 10 (Filling searchplane). A searchplane of a guard in a polyhedron \mathcal{P} is filling if it is a closed set whose relative boundary lies entirely on \mathcal{P} 's boundary.

Consider a guard ℓ with a non-trivial searchplane S , and let α be the plane containing S . Then S is filling if and only if it coincides with the connected component of $\alpha \cap \mathcal{P}$ containing ℓ .

Notice that the searchplane depicted in Figure 2 is not filling, because one of its edges lies in the interior of the polyhedron (let us call this edge e). In addition, this searchplane is not even a closed set: recall that guards have no endpoints, so edge e is not actually part of the searchplane (except for one endpoint).

Definition 11 (Filling guard). *A guard is filling if all its searchplanes are filling.*

Because searchplanes are induced by half-planes, it follows that only boundary guards can be filling. Intuitively, a filling guard is similar to a traditional boundary guard from SSP in simple polygons, in that its searchlight provides at any time an effective barrier which cannot be crossed by the intruder just by walking past its borders. The importance of such guards in developing search algorithms will be clear shortly.

3 Basic Results

Counterexamples

The most noteworthy aspect of our guard model is that searchplanes of boundary guards, as opposed to searchlight rays emanating from boundary guards in SSP, may fail to disconnect a polyhedron when aimed at its interior, regardless of its genus. As it turns out, this is the main reason why 3SSP seems harder than SSP, in that exploiting such a property will enable the relatively simple NP-hardness proof in Section 5, as well as the construction of several counterexamples to positive statements about SSP.

For example, the reduction of the search space to *sequential schedules* (i.e., schedules in which the guards sweep in turns) given by Obermeyer et al. is no longer possible.[5] Figure 3 shows an instance of 3SSP whose two guards are forced to turn their searchlights simultaneously, or else they would create gaps in the illuminated surface which would result in the recontamination of the whole polyhedron.

Moreover, in spite of the searchability of all SSP instances whose guards lie on the boundary and collectively see the whole polygon (see Ref. 1), it is easy to construct guard-visible but unsearchable instances of 3SSP with only boundary guards, such as those in Figure 4.[7] Indeed, whenever the two guards attempt to clear the center (in either of these two instances), they

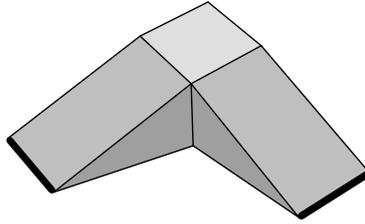


Figure 3: Searchable instance of 3SSP with no sequential search schedule. Thick lines mark guards.

fail to disconnect the polyhedron, since their searchplanes are not coplanar, which results in the recontamination of the entire instance. In Section 5 we will provide more sophisticated unsearchable but guard-visible instances of 3SSP with only boundary guards.

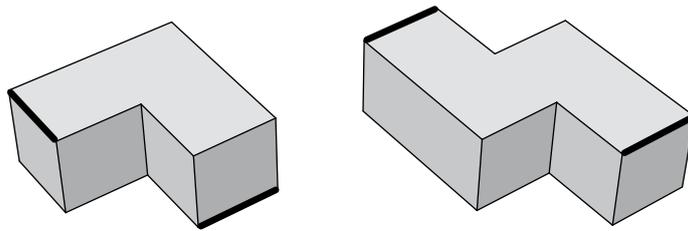


Figure 4: Two unsearchable guard-visible instances of 3SSP whose guards lie on the boundary.

Filling guards

Notice that all the previous counterexamples employ boundary guards that are not filling. Conversely, it comes as no surprise that employing only filling guards yields positive results. To see why, let us give a characterization of the different shapes a searchplane can take with respect to the surrounding polyhedral environment. The topological closure of a searchplane is always a polygon, perhaps with holes, perhaps with some additional segments sticking out radially, and the whole searchplane is visible to some line segment lying on its external boundary, which would be the guard generating it (refer to Figure 5). There may be intersections between a searchplane's relative interior and the polyhedral boundary, which could be collections of polygons, straight line segments, and isolated points. But what is central for our purposes is the searchplane's relative boundary, which may entirely lie on the polyhedron's boundary, or may not. If it does, and the searchplane is a closed set, then the only way an intruder could travel from one side of the

searchplane to the other, without crossing the light and being caught, would be to take a detour through a suitable handle of the polyhedron. In particular, in genus-zero polyhedra, that would be impossible. In other words, any searchplane emanating from a guard aiming its searchlight at the interior of a genus-zero polyhedron disconnects the polyhedron if and only if the searchplane is filling. Thus, any filling guard aiming its searchlight at the interior of a genus-zero polyhedron disconnects it.

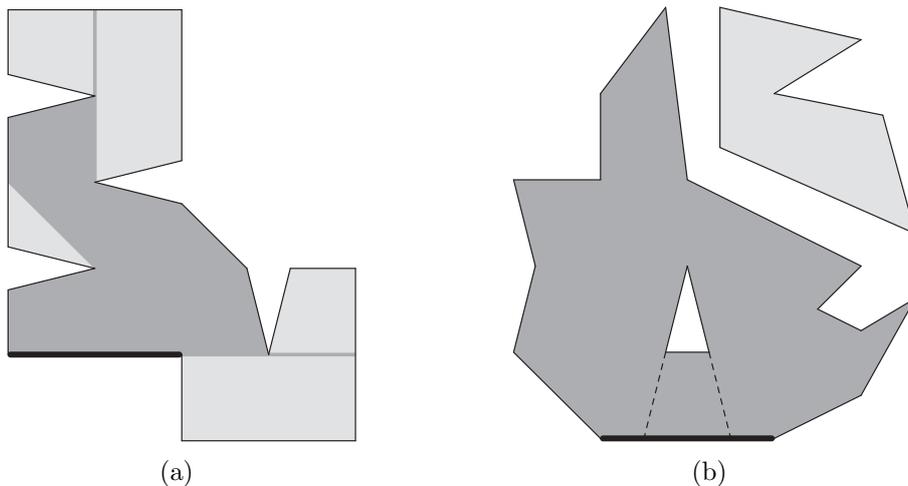


Figure 5: Two sections of polyhedra, with searchplanes represented as dark regions. The searchplane in (a) has two dangling segments, while the searchplane in (b) is filling but not simply connected.

On the other hand, if $\ell \subset \mathcal{P}$ is filling, the topological closure of $\mathcal{V}(\ell)$ is always a polyhedron, perhaps with some dangling polygons (which may originate from dangling segments of single searchplanes, such as those in Figure 5(a)). Of course, the boundary of $\mathcal{V}(\ell)$ may contain polygons that do not lie on \mathcal{P} 's boundary. But, because ℓ is filling, every such polygon is entirely contained in some searchplane of ℓ .

Definition 12 (Critical searchplane, critical position). *A searchplane S of a filling guard ℓ is critical if it shares a polygon with the boundary of $\mathcal{V}(\ell)$ that does not lie on \mathcal{P} 's boundary. Whenever ℓ is aimed at S , it is in a critical position.*

Every filling guard ℓ has only a finite number of critical searchplanes, because $\mathcal{V}(\ell)$'s surface is made of finitely many polygons. Equivalently, ℓ has a finite number of critical positions.

Now, as a filling guard ℓ in a genus-zero polyhedron starts turning from its leftmost position toward the right, every point that it illuminates will remain

clear forever, unless the illuminated searchplane becomes tangent to some region of the polyhedron that is not in $\mathcal{V}(\ell)$ and that would be responsible for recontamination, once the tangency is crossed by the searchlight. This happens exactly when ℓ reaches a critical position.

One-way sweeping

We now have the tools required to generalize the one-way sweep strategy for guards in simple polygons (outlined in Ref. 1 by Sugihara et al.) to work with filling guards in simply connected polyhedra.

Theorem 2. *Every guard-visible genus-zero instance of the 3-DIMENSIONAL SEARCHLIGHT SCHEDULING PROBLEM is searchable if its guards are filling.*

Proof. We first sketch a search schedule before detailing it further. Select any guard ℓ and start turning it rightward from its leftmost position. As soon as it reaches a critical position, it means that some *subpolyhedron* $\mathcal{R} \subset \mathcal{P}$ has been encountered that is invisible to ℓ . So stop turning ℓ and select another guard to continue the job. Proceed recursively until \mathcal{R} is clear, then turn ℓ rightward again, stopping at every critical position, until the entire polyhedron is clear.

At any time, there is a clear region of \mathcal{P} that is steadily growing, and a *semiconvex subpolyhedron* \mathcal{R} supported by a set of guards L that is being cleared by some guard not in L , while the guards in L hold their searchlights fixed. Intuitively, the term “semiconvex” is used because the only points of non-convexity of such a polyhedron lie on \mathcal{P} ’s boundary. For a formal definition of *semiconvex subpolygon* and *support*, refer to Ref. 1 or Ref. 5. Extending these definitions to polyhedra is straightforward, since we are considering only filling guards in genus-zero polyhedra. Thus, part of the boundary of \mathcal{R} coincides with \mathcal{P} ’s boundary, while the remaining part is determined by the searchplanes of the guards in L . Moreover, all the guards in L are in a critical position, waiting for \mathcal{R} (or some larger semiconvex subpolyhedron of \mathcal{P} , supported by a subset of L) to be cleared. It follows that none of the guards in L can see any point in the interior of \mathcal{R} .

Hence, there must be some guards not in L that can see an internal portion of \mathcal{R} , otherwise the problem instance would not be guard-visible. We select one of them, say ℓ' , and start sweeping \mathcal{R} from left to right (according to ℓ' ’s notion of left and right). Notice that a searchplane bounding \mathcal{R} could have holes, and thus \mathcal{R} itself could have strictly positive genus. But that does not affect our invariants, because every searchplane of ℓ' passing through \mathcal{R} ’s interior still disconnects it, or it would not even disconnect \mathcal{P} . As a conse-

quence, the points in \mathcal{R} that are illuminated by ℓ' never get recontaminated as ℓ' continues its sweep.

Again, whenever ℓ' reaches a critical position, it stops there until the semiconvex subpolyhedron $\mathcal{R}' \subset \mathcal{R}$ supported by $L \cup \{\ell'\}$ has been cleared by some other guard, and so on recursively. Since every guard has only a finite number of critical positions, eventually \mathcal{P} gets cleared. \square

Remarkably enough, the core argument supporting the planar one-way sweep strategy of Sugihara et al. (see Ref. 1) applies also to our polyhedral model, where there is no well-defined global concept of clockwise rotation.

Sequentiality

In addition to this, a version of the main result of Obermeyer et al. (see Ref. 5) also extends to instances of 3SSP with filling guards. Assuming that all the guards are filling, we say that a search schedule is *critical* and *sequential* if at most one guard is turning at any given time, and guards stop or change direction only at critical positions, or at their leftmost or rightmost positions.

Corollary 3. *Every searchable genus-zero instance of the 3-DIMENSIONAL SEARCHLIGHT SCHEDULING PROBLEM whose guards are filling has a critical and sequential search schedule.*

Proof. Since the instance is searchable, it must be guard-visible. Thus, the search schedule detailed in the proof of Theorem 2 applies, which is indeed critical and sequential. \square

Searching with one guard

As a further application of the concept of filling guard, we characterize the searchable instances of 3SSP having just one guard. We include polyhedra of any genus, not just zero (e.g., triangular tori).

Theorem 4. *An instance of the 3-DIMENSIONAL SEARCHLIGHT SCHEDULING PROBLEM (of any genus) with just one guard is searchable if and only if it is guard-visible and the only guard is a boundary guard.*

Proof. Both conditions are clearly necessary, for the same reasons why they are necessary in 2-dimensional SSP.[1] Specifically, if the guard has a point that does not lie on the polyhedron's boundary, then any small-enough ball centered at that point can never be cleared.

Conversely, since the instance is guard-visible, the visibility region of its only guard ℓ coincides with the whole polyhedron \mathcal{P} . Let ℓ' be the maximal

straight line segment contained in \mathcal{P} and containing ℓ . Then ℓ' entirely belongs to the boundary of \mathcal{P} . Indeed, if a point $x \in \ell'$ lay strictly in the interior of \mathcal{P} , then also some neighborhood of x would. Recall that ℓ has a range of blind directions past its leftmost and rightmost position, where all its searchplanes degenerate to the single line ℓ' . In all those directions, part of the neighborhood of x would lie outside $\mathcal{V}(\ell)$, contradicting the guard-visibility of the instance.

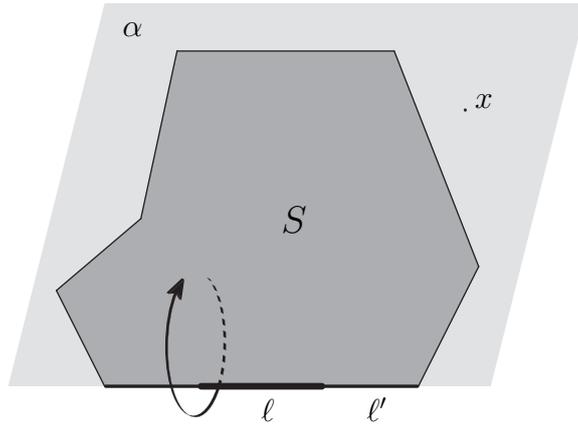


Figure 6: Illustration of the proof of Theorem 4.

Consider a searchplane S , lying on a half-plane α whose bounding line contains ℓ (refer to Figure 6). If a point $x \in \alpha \cap \mathcal{P}$ lay outside S , then x would necessarily be covered by another searchplane, because $\mathcal{V}(\ell) = \mathcal{P}$. But the only points in α that could lie on a searchplane different from S would be those in ℓ' , because any two searchplanes of ℓ intersect just on ℓ' . Nonetheless, ℓ' belongs to S too, which yields a contradiction. Hence $S = \alpha \cap \mathcal{P}$ and also ℓ' lies on the boundary of \mathcal{P} , implying that ℓ is a filling guard with no critical positions. Moreover, S disconnects \mathcal{P} if it intersects its interior. Suppose by contradiction that an intruder could walk from one side of S to the other side. Since $S = \alpha \cap \mathcal{P}$, the intruder would necessarily have to take the long route around ℓ' (i.e., opposite to α , see Figure 6), and by doing so it would cross all the half-planes in the blind directions of ℓ . But the only points of \mathcal{P} that belong to such half-planes are those in ℓ' , so the intruder would be caught in any case.

It follows that turning ℓ from its leftmost position to its rightmost position produces a search schedule. \square

Notice that, had we included endpoints in Definition 1, the statement of Theorem 4 would have been false. Indeed, the guard-visibility assumption would have been satisfied by more 3SSP instances, including unsearchable

ones. For example, consider a cube with a “pyramidal well” pointing inside, and an edge guard on a reflex edge. The entire polyhedron is visible to the pyramid’s vertex, which is an endpoint of the guard. However, the guard cannot search the polyhedron, because no searchplane disconnects it, in either guard model—with or without endpoints.

Checking for fillingness

We conclude this section by sketching an argument supporting the claim that the conditions of Theorem 2 are practically checkable.

Proposition 5. *Whether a guard $\ell \subset \mathcal{P}$ is filling can be decided in time polynomial in the size of \mathcal{P} .*

Proof. First of all, if ℓ is not a boundary guard, then it cannot be filling and may be discarded.

Then, observe that the fillingness of a searchplane S generated by a half-plane α can be efficiently checked by computing the polygonal section $P = \alpha \cap \mathcal{P}$, and then computing the boundary of S by drawing lines through every pair of vertices of P . Once the boundary of S is known, its fillingness can be checked easily.

Now, determining if ℓ is filling can be carried out by inspecting just a polynomial number of its searchplanes. For every face $F \subset \mathcal{P}$ and every edge $e \subset \mathcal{P}$ that is not parallel to F , we call the point of intersection between the plane containing F and the line containing e a *critical point* (not to be confused with “critical position,” see Definition 12). In particular, every vertex of \mathcal{P} is a critical point. Imagine turning the searchlight of ℓ from its leftmost position to the rightmost: it is straightforward to see that the fillingness of the illuminated searchplane can change only when the searchlight crosses a critical point. Hence, it suffices to check the fillingness of every searchplane corresponding to a critical point, plus one searchplane for each interval between two consecutive critical points. The number of searchplanes to check is thus polynomial. \square

Making the above proposition into an optimal algorithm goes beyond the scope of this paper. However, the interested reader can readily see that even a naive implementation yields a time complexity of $O(n^4)$, regardless of the actual data structures used to represent polyhedra.

4 Search Strategies

In this section we tackle the problem of efficiently placing boundary guards in a given polyhedron in order to make it searchable, while partially settling Conjecture 1 in Ref. 7 (i.e., that any polyhedron with a guard on each reflex edge is searchable). Our general approach employs r^2 boundary guards for polyhedra with $r > 0$ reflex edges, which we believe to be asymptotically suboptimal. However, we also prove that just r edge guards are sufficient for orthogonal polyhedra. A related goal is to minimize the *search time*, which is the total time of a search schedule, assuming that the maximum angular speed of every guard is 2π rad/sec. We will show that occasionally it is possible to trade guards for search time, to some extent.

Remark 3. *Formally, when considering search time, the set of a guard's legal schedules is restricted to Lipschitz continuous functions, whose Lipschitz constant matches an angular speed of 2π rad/sec.*

Minimizing guards

The problem of minimizing the number of (boundary) guards required to search a given polyhedron is strongly NP-hard, even for genus-zero orthogonal polyhedra. The problem is strongly NP-hard also restricted to edge guards, or to edge guards lying on reflex edges. Indeed, the same approach used by Lee and Lin in Ref. 8 to show the strong NP-hardness of the ART GALLERY PROBLEM with vertex guards in general polygons can be employed, with some additional adjustments, to construct a simple orthogonal polygon with the same properties. By subsequently extruding the polygon into an orthogonal prism and after some minor modifications, the same hardness result can be obtained for edge guards, as well. Finally, passing from the ART GALLERY PROBLEM to 3SSP is almost automatic because boundary point guards can search a simple polygon if and only if they solve the ART GALLERY PROBLEM.[1]

4.1 Searching general polyhedra

Open edge guarding

We first need consider a planar ART GALLERY PROBLEM for *open edge guards* (i.e., edge guards without endpoints), which has apparently not been previously explored. Specifically, we want to select some edges of a given polygon P , in such a way that each point of P is visible to an *internal* point

of at least one of the selected edges. For technical reasons (see the proof of Theorem 7), we also want to force the selection of a specific edge.

Lemma 1. *Every polygon with r reflex vertices, h holes and a distinguished edge e is guardable by at most $r - h + 1$ open edge guards, one of which lies on e .*

Proof. Let P be a polygon, select any reflex vertex v and draw the bisector of the corresponding internal angle, until it again hits the boundary of P . If the ray hits another vertex, slightly rotate it about v , so that it instead hits the interior of an edge. Two situations can occur: either P gets partitioned in two parts, with $r - 1$ total reflex vertices and h total holes, or two connected components of the boundary are joined, so that P loses both a hole and a reflex vertex.

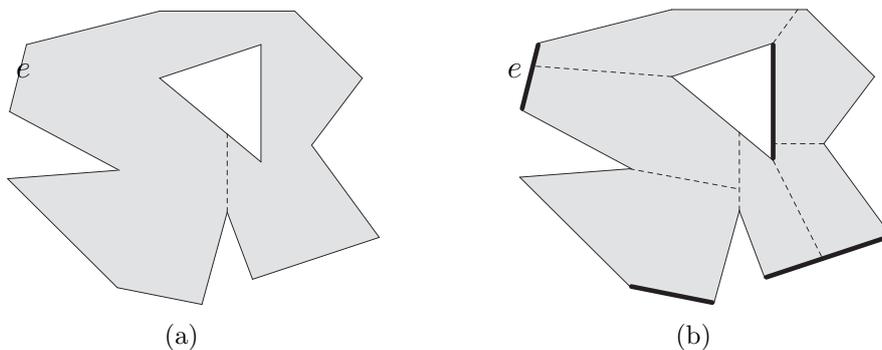


Figure 7: Example of the construction in Lemma 1. In (a) the first step is shown, where the dashed line acts as a degenerate edge of the resulting polygon. In (b) the final partition is shown, with the selected edges represented as thick lines.

Repeat the process inductively on the resulting polygons, until no reflex vertex remains. Notice that polygonal boundaries may be *degenerate* in the intermediate steps of this construction, meaning that a single segment should occasionally be regarded as two coincident segments (refer to Figure 7). P is now partitioned into convex pieces, which of course have no holes. Thus, during the process, the number of holes decreased h times, while the number of pieces in the partition increased $r - h$ times, resulting in $r - h + 1$ convex polygons. Additionally, each polygon has at least one edge lying on P 's boundary, and conversely every internal point of each edge of P sees at least one complete region. Place a guard on the distinguished edge e , thus guarding at least one region of the partition. For each unguarded (or partially guarded) region, choose an edge of P that completely sees it, and place a guard on it.

As a result, \mathcal{P} is completely guarded and at most $r - h + 1$ open edge guards have been placed. \square

The previous bound is asymptotically best possible (as a function of r), because polygons with r reflex vertices can be constructed that require r open edge guards, for every r (e.g., the “shutter polygons” in Ref. 2, Fig. 1.27).

Basic search strategy

To begin with, we argue that $r^2 + r$ boundary guards are sufficient to search any given polyhedron with $r > 0$ reflex edges. Subsequently, we show how to lower this bound to r^2 , at the expense of increasing search time.

Theorem 6. *Any polyhedron with $r > 0$ reflex edges can be searched in less than 1 second by at most $r^2 + r$ suitably chosen boundary guards.*

Proof. Let \mathcal{P} be a non-convex polyhedron. We first partition it into convex regions \mathcal{C}_i , showing how this partition is induced by at most r^2 boundary guards. Next, we obtain a superpartition \mathcal{D}_j with some better properties that we exploit to obtain a search schedule, using r new boundary guards (a very similar partition was described by Chazelle in Ref. 9, under the name of “naive decomposition revisited”).

First partition. Let e be a reflex edge of \mathcal{P} , and let α_e be a plane through e , close enough to its angle bisector, but not containing any vertex of \mathcal{P} other than e ’s endpoints. Let \mathcal{Q}_e be the connected component of $\alpha_e \cap \mathcal{P}$ containing e . We claim that \mathcal{Q}_e is a polygon with at most $r - 1$ reflex vertices, possibly with holes. Indeed, e is an edge of \mathcal{Q}_e , and each reflex vertex of \mathcal{Q}_e lies on a reflex edge of \mathcal{P} . Moreover, if an endpoint of e is a reflex vertex of \mathcal{Q}_e , then it belongs at least to another reflex edge of \mathcal{P} , different from e . But α_e intersects every edge of \mathcal{P} other than e in at most one point (otherwise it would contain its endpoints, as well), hence there are at most $r - 1$ reflex vertices of \mathcal{Q}_e , i.e., one for every reflex edge of \mathcal{P} other than e .

By Lemma 1, \mathcal{Q}_e can be guarded by at most r (2-dimensional) open edge guards, one of which lies on e . Equivalently, \mathcal{Q}_e can be completely illuminated by at most r suitably placed boundary guards (with searchlights), lying on α_e and aimed parallel to it, one of which is an edge guard lying on e . By repeating the same construction with every other reflex edge, at most r^2 boundary guards are placed, and \mathcal{P} is partitioned by illuminated searchplanes into convex polyhedra \mathcal{C}_i . We remark that, during our construction, previously placed searchplanes are not considered as part of \mathcal{P} ’s boundary. Hence, every \mathcal{Q}_{e_k} is indeed bounded by \mathcal{P} . This is because we need place guards on the boundary of \mathcal{P} , and not in its interior.

A coarser partition. Consider now a slightly different partition: proceed as above by drawing angle bisectors through reflex edges, but this time every previously drawn splitting polygon acts as a boundary. In other words, as soon as \mathcal{P} splits into several subpolyhedra, we partition them recursively one by one, confining each split to just one subpolyhedron. So, if we consider some intermediate subpolyhedron $\mathcal{P}' \subseteq \mathcal{P}$ and select a reflex edge $e' \subset \mathcal{P}'$, we look at the reflex edge e of \mathcal{P} that contains e' , and let $\mathcal{R}_{e'}$ be the connected component of $\alpha_e \cap \mathcal{P}'$ (as opposed to $\alpha_e \cap \mathcal{P}$) containing e' . As a result, \mathcal{P} is again partitioned into convex polyhedra \mathcal{D}_j , in such a way that every \mathcal{C}_i is contained in some \mathcal{D}_j (i.e., $\{\mathcal{C}_i\}$ is a *refinement* of $\{\mathcal{D}_j\}$). Notice that, even though two different *splitting polygons* $\mathcal{R}_{e'}$ and $\mathcal{R}_{e''}$ may correspond to the same reflex edge e of \mathcal{P} (because e itself has been split by a previous cut), they are nonetheless coplanar, as they both belong to the same plane α_e .

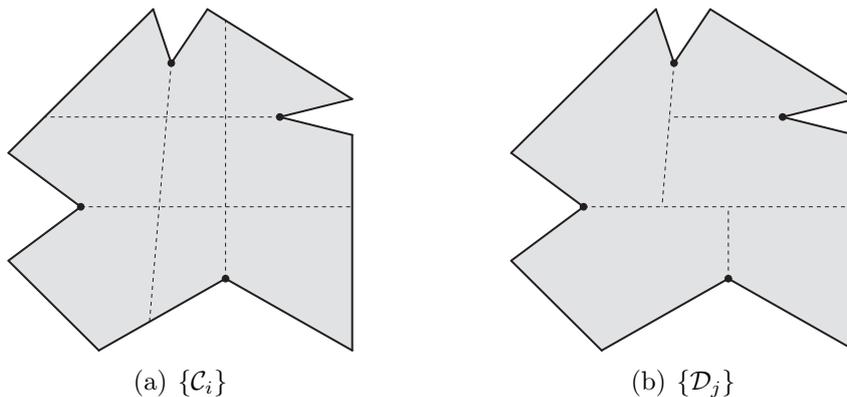


Figure 8: Comparison of the partitions $\{\mathcal{C}_i\}$ and $\{\mathcal{D}_j\}$ in a section of a polyhedron. For simplicity, only the splitting planes corresponding to visible reflex edges are shown.

Search schedule. The point of having this coarser partition is that every \mathcal{D}_j contains a subsegment of a reflex edge of \mathcal{P} (see Figure 8). This property can be checked by a straightforward induction on the construction steps.

Therefore, we add a new edge guard on each reflex edge, and we turn these additional guards simultaneously, from their leftmost position to the rightmost. While this happens, the r^2 previously placed guards are never moved, so that the partition $\{\mathcal{D}_j\}$ is always preserved. Because each \mathcal{D}_j is completely visible to some reflex edge, at the end of the schedule every \mathcal{D}_j is successfully cleared.

Since every guard turns by less than 2π rad, the search time is less than 1 second. \square

Improved search strategy

We can slightly lower the number of guards used in the previous theorem, at the cost of increasing the search time. Next we show how to use only r^2 boundary guards, although the search time of our schedule could be quadratic in r , as well.

Theorem 7. *Any polyhedron with $r > 0$ reflex edges can be searched by at most r^2 suitably chosen boundary guards.*

Proof. As in the proof of Theorem 6, we partition the polyhedron \mathcal{P} into convex regions \mathcal{D}_j by placing at most r^2 boundary guards. Then we show that some of these guards can be turned in a certain order to clear every piece of the partition.

In the following, we use the same notation as in the proof of Theorem 6.

Partition tree. During the splitting process of \mathcal{P} that culminates in the partition $\{\mathcal{D}_j\}$, we build a tree whose nodes represent the intermediate subpolyhedra, and whose arcs are marked by the splitting polygons \mathcal{R}_e (see Figure 9). Thus, the root of the tree is \mathcal{P} and its leaves are the \mathcal{D}_j 's. Every time we draw a splitting polygon \mathcal{R}_e for a subpolyhedron \mathcal{P}' corresponding to some node v of the tree, we could either decrease the genus of \mathcal{P}' , or we could partition it into two subpolyhedra \mathcal{P}'_1 and \mathcal{P}'_2 , one for each side of \mathcal{R}_e . In the first case we just attach to v an arc labeled \mathcal{R}_e with a node labeled as the new polyhedron, say \mathcal{P}'' . In the second case we attach to v two arcs labeled \mathcal{R}_e , with two sibling nodes labeled \mathcal{P}'_1 and \mathcal{P}'_2 . This structure is somewhat reminiscent of a Binary Space Partitioning tree. Again, as in the proof of Lemma 1, we have to slightly extend the notion of polyhedron, to include those with internal polygons acting as faces, resulting from non-disconnecting splits.

Search schedule. Next we describe a search schedule for the r^2 boundary guards we placed. Recall that each \mathcal{D}_j contains a subsegment of a reflex edge of \mathcal{P} , and that on each such reflex edge lies one edge guard. We turn only *some* of these r guards, one by one, while all the other guards stay still. The order of activation of the guards is determined by the above partition tree, and the same guard could be activated more than once. The subpolyhedra of \mathcal{P} are cleared recursively, following a depth-first walk of the partition

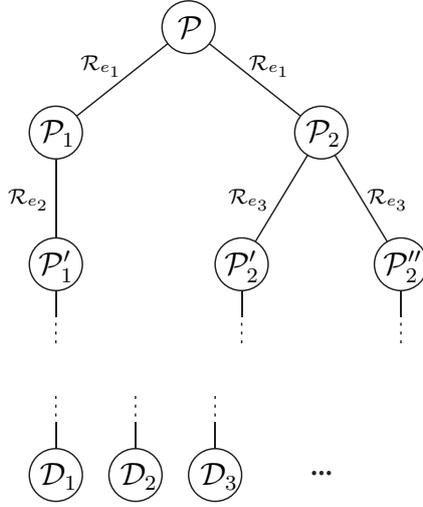


Figure 9: Sketch of a partition tree.

tree, starting from the root. Every time a leaf is reached through an arc labeled \mathcal{R}_e , its corresponding \mathcal{D}_j is swept by the edge guard lying on e . This is feasible because e lies on the boundary of \mathcal{D}_j , which is convex. After \mathcal{D}_j is clear, the guard moves back to its initial position and the depth-first walk proceeds.

Correctness. It remains to show that no *significant* recontamination can occur among the \mathcal{D}_j 's while their bounding searchlights are rotated. Suppose the depth-first walk of the partition tree reaches a leaf labeled \mathcal{D}_j , and let ℓ_e be the edge guard whose duty is to clear \mathcal{D}_j . Perhaps the corresponding edge e of \mathcal{P} is divided several times by the partition, so let E be the set of subsegments of e whose corresponding splitting planes actually appear as labels of the edges of the partition tree. Let $e' \in E$ be the subsegment of e that is also an edge of \mathcal{D}_j . On the other side of $\mathcal{R}_{e'}$ lies a subpolyhedron \mathcal{P}' , such that \mathcal{D}_j and \mathcal{P}' are represented by sibling nodes in the partition tree. In order to clear \mathcal{D}_j , ℓ_e turns its searchlight from $\mathcal{R}_{e'}$ to sweep over \mathcal{D}_j , and then back to $\mathcal{R}_{e'}$. Since the *restriction* of ℓ_e to \mathcal{D}_j is filling, no recontamination occurs between \mathcal{D}_j and \mathcal{P}' during this back-and-forth sweep (see Figure 10).

Nonetheless, recontamination could still occur between other subpolyhedra bounded by α_e . Let $\{e_1, e_2\} \subseteq E$, and let the subpolyhedra \mathcal{P}_1 and \mathcal{P}_2 be partitioned by \mathcal{R}_{e_1} , and \mathcal{R}_{e_2} , respectively (observe that the relative interiors of e_1 and e_2 must be disjoint, by construction). Obviously, no recontamination between \mathcal{P}_1 and \mathcal{P}_2 is possible while ℓ_e sweeps, because α_e is not a common boundary (refer to Figure 11). However, recontamination

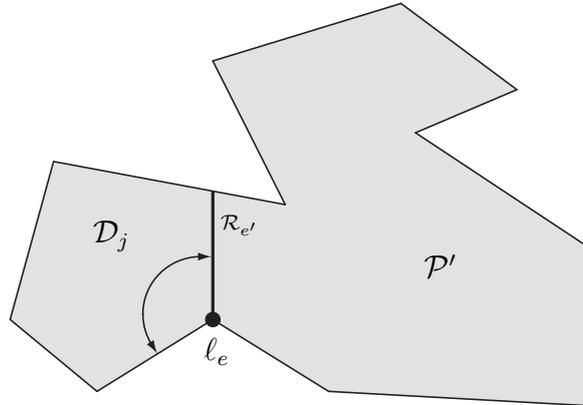


Figure 10: \mathcal{P}' is not recontaminated while ℓ_e sweeps \mathcal{D}_j .

could occur within \mathcal{P}_1 (or within \mathcal{P}_2), provided that $e_1 \neq e'$. As it turns out, this type of recontamination is irrelevant, because it would imply that the node labeled \mathcal{P}_1 (call it p) has not been reached yet by the depth-first walk in the partition tree. We set up an induction argument on the partition tree, assuming that all the subpolyhedra corresponding to previously visited leaves are still clear before ℓ_e sweeps. By construction, p cannot be an ancestor of the node labeled \mathcal{D}_j , otherwise \mathcal{D}_j would be a subpolyhedron of \mathcal{P}_1 , while in fact their interiors are disjoint. So, had the depth-first walk reached p , it would have also visited its entire dangling subtree and, by inductive assumption, \mathcal{P}_1 would still be all clear. Moreover, since \mathcal{P}_1 is not bounded by α_e , it cannot get recontaminated while ℓ_e sweeps. On the other hand, if the walk has not reached p yet, then perhaps some portions of \mathcal{P}_1 have been accidentally cleared, but can safely be recontaminated, because the systematic clearing process of \mathcal{P}_1 has yet to start. \square

4.2 Searching orthogonal polyhedra

The previous results can be greatly improved if the polyhedron is orthogonal.

We define the *vertical* direction (or *up-down* direction) as the direction parallel to the z -axis. Accordingly, any direction parallel to the xy -plane is called *horizontal*, and the direction parallel to the x -axis (resp. y -axis) is called *left-right* (resp. *front-back*) direction. Thus, the edges and faces of any orthogonal polyhedron are either vertical or horizontal, and the x -orthogonal (resp. y -orthogonal) faces are called *lateral faces* (resp. *front faces*).

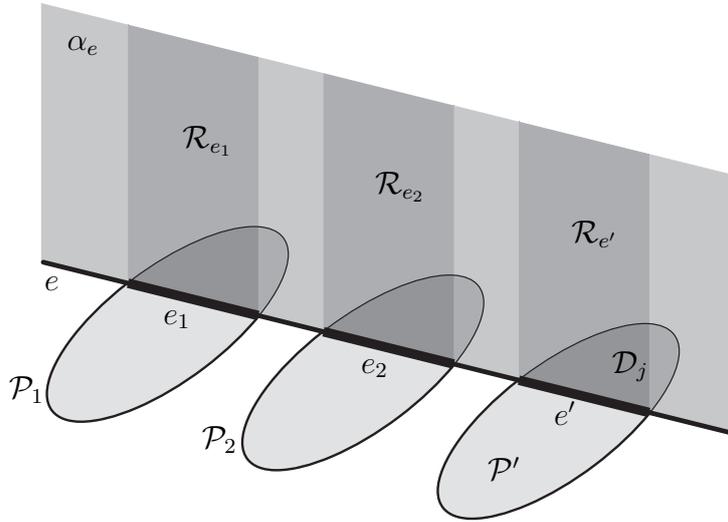


Figure 11: Sketch of a splitting plane.

Erecting fences

We first obtain a partition of a given orthogonal polyhedron \mathcal{P} into *cuboids* (i.e., rectangular boxes) by constructing vertical *fences* in a 3-step process. Then we will argue that these fences may be regarded as (parts of) search-planes of suitably oriented edge guards lying on the reflex edges of \mathcal{P} . Finally, after pointing out some basic properties of our partition, we will show that rotating all the searchlights one by one clears every cuboid.

(The term “fence” is borrowed from Ref. 10, where Chazelle and Palios described a partition of general polyhedra into prisms that somewhat resembles that of our Step 1.)

1. Each horizontal reflex edge r of \mathcal{P} has a horizontal adjacent face and a vertical adjacent face F , going upwards or downwards. From every point on r , send a vertical ray in the direction opposite to F , until it again reaches the boundary of \mathcal{P} . Repeat the process with every horizontal reflex edge, so that each generates a vertical *fence*, either upwards or downwards. It is easy to see that \mathcal{P} is partitioned by fences into orthogonal *prisms* (i.e., extruded polygons) with horizontal bases.
2. Consider any vertical reflex edge r of \mathcal{P} with an internal point lying on a lateral face (i.e., an x -orthogonal face) of a prism \mathcal{Q} generated in Step 1. By construction, r must lie entirely on the boundary of \mathcal{Q} . Extend r to a maximal segment r' contained in the boundary of \mathcal{Q} . If possible, send a horizontal ray from every point of r' , going through the

interior of Q in the left-right direction, until it reaches Q 's boundary, as shown in Figure 12. The rectangle so formed is again called a fence. Repeating the same construction with every other vertical reflex edge of \mathcal{P} lying on a lateral face of some prism, we obtain a refinement of the initial partition by these new fences. Clearly, to every such reflex edge corresponds just one fence, which in turn goes through the interior of just one prism.

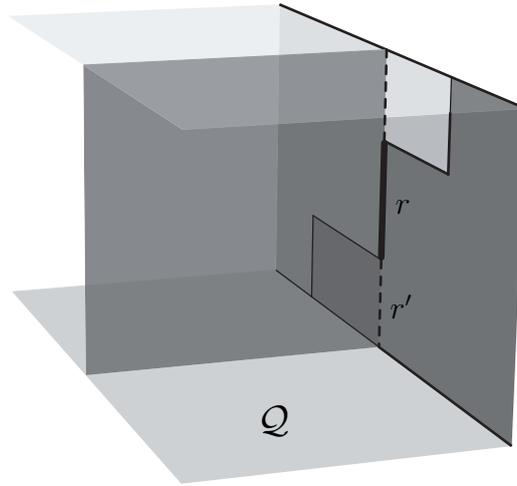


Figure 12: Fence generated by r during Step 2. Fences are shown as darker regions.

3. The pieces resulting from Step 2 are once again prisms, perhaps containing some additional vertical fences that are not faces. Indeed, some fences constructed in Step 2 split a prism in two subprisms, while others just lower a prism's genus and act as degenerate faces (which will become legitimate faces of cuboids shortly). By construction, each reflex edge of such a prism is vertical, with no edges of \mathcal{P} lying on it. Repeat the procedure in Step 2 also with these reflex edges, sending horizontal rays and thus building vertical fences that extend laterally the front faces of the prisms, as shown in Figure 13. As a result, \mathcal{P} gets partitioned into cuboids.

Observe that all the fences generated in Steps 2 and 3 are y -orthogonal rectangles. The reason why we distinguish three steps is that the respective fences need be treated differently in the proofs to come.

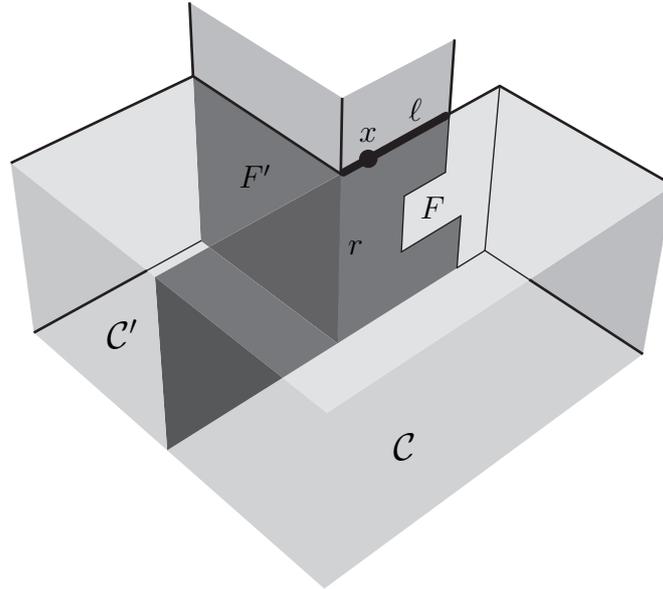


Figure 13: Fence generated by r during Step 3. Fences are shown as darker regions. The nomenclature comes from Lemmas 2 and 3.

Placing guards

Now place an edge guard on each reflex edge of \mathcal{P} . Aim horizontal guards vertically and aim vertical guards in the left-right direction, in such a way that every guard aims at the interior of \mathcal{P} . We first want to prove that the fences in our 3-step construction are *coherent* with searchlights.

Lemma 2. *Every fence is contained in some illuminated searchplane.*

Proof. The claim is obvious for fences constructed in Step 1 and Step 2. As for fences constructed in Step 3, consider an edge r generating one of them (see Figure 13). Recall that r is a reflex edge of a prism, let it be \mathcal{Q} , in the partition obtained in Step 1. By the construction in Step 2, r contains no reflex edge of \mathcal{P} . Hence its adjacent front face $F \subset \mathcal{Q}$ has no intersection with the boundary of \mathcal{P} , in a thin-enough neighborhood of r . But both bases of \mathcal{Q} belong to the boundary of \mathcal{P} , because no horizontal fences were constructed (recall that the upper face and the bottom face of a prism are both called “bases”). Then, at least a subsegment of a horizontal edge of F , sharing an endpoint with r , belongs to a reflex edge of \mathcal{P} , and hence belongs to a guard ℓ . As a consequence, ℓ illuminates the whole fence generated by r in Step 3. \square

Thus, every fence *belongs* to one guard. We could also incorporate each

fence built in Step 3 into its adjacent vertical fence built in Step 1 that belongs to the same guard (e.g., in Figure 13, the two fences parallel to face F are “merged” and belong to guard ℓ). As a result, every guard *generates* at most one fence.

Lemma 3. *Every cuboid in the partition belongs entirely to the visibility region of any guard whose fence bounds it.*

Proof. If a fence built in Step 1 bounds a cuboid, then it belongs to a guard ℓ located, at least partially, on its border. Indeed, fences are vertical and the upper and lower faces of a cuboid belong to faces of \mathcal{P} , and therefore ℓ cannot lie entirely outside the cuboid. It follows that ℓ sees the whole cuboid.

Fences built in Step 2 bound exactly two cuboids each, because the fences built in Step 3 are all parallel to them. Again, any guard generating one such fence belongs to a common edge of the cuboids that it bounds, which of course are entirely visible to the guard.

Also fences built in Step 3 bound exactly two cuboids each. With the same notation as in the proof of Lemma 2, the fence generated by r bounds a cuboid \mathcal{C} that is also bounded by F , and therefore contains, at least partially, guard ℓ on one of the horizontal edges of \mathcal{C} (again, ℓ is defined as in the proof of Lemma 2). ℓ also shares one endpoint with r . Moreover, the other cuboid \mathcal{C}' is bounded also by a lateral face $F' \subset \mathcal{Q}$ adjacent to r . The interior of F' has no intersection with the boundary of \mathcal{P} . Indeed, if the two had any intersection, then there would be also a vertical reflex edge of \mathcal{P} lying in the interior of F' , or on r . But this disagrees with the construction in Step 2, which eliminates all such reflex edges. Let x be a point in ℓ that is close enough to r (e.g., whose distance to r is lower than the minimum positive difference between any two coordinates of vertices of \mathcal{P} —such minimum exists due to the finiteness of \mathcal{P} 's vertices). Of course, x completely sees \mathcal{C} , because it lies on its boundary. But x also sees every point in \mathcal{C}' , through the fence F' (see Figure 13). Indeed, by our choice of x , the pyramid determined by F' and x is completely contained in \mathcal{P} . \square

Search strategy

Theorem 8. *Any non-convex orthogonal polyhedron with an edge guard on each reflex edge is searchable.*

Proof. Aim the guards as described above, in such a way that every fence is illuminated by some guard, by Lemma 2. Clearly, illuminated searchplanes induce the same partition of \mathcal{P} into cuboids (perhaps even a finer partition). Now pick a guard ℓ generating a fence F , and let \mathcal{Q} be the union of the cuboids

bounded by F . Since F is connected and has cuboids on both sides, it follows that \mathcal{Q} is connected as well, and therefore it is a polyhedron. Moreover, by Lemma 3, \mathcal{Q} entirely belongs to $\mathcal{V}(\ell)$. Hence, by Theorem 4, \mathcal{Q} can be cleared by ℓ while all the other guards keep their searchlights fixed. Turn ℓ to clear \mathcal{Q} , put ℓ back in its original position, and repeat the procedure for all the other guards, one at a time. Notice that every turning guard clears all the cuboids that it bounds, while the other cuboids cannot be recontaminated, because their boundaries remain fixed. Since \mathcal{P} is both connected and non-convex, every cuboid is bounded by at least one fence, which in turn is generated by some guard. Thus, after the last guard has finished sweeping, \mathcal{P} is completely clear. \square

Improving search time

The search time of the schedule given by Theorem 8 is linear in the number of reflex edges of \mathcal{P} , but once again we can achieve constant search time by doubling the number of guards.

Corollary 9. *Any non-convex orthogonal polyhedron with two (coincident) edge guards on each reflex edge is searchable in $\frac{3}{4}$ seconds.*

Proof. Half of the guards are positioned as in the proof of Theorem 8 and never move, thus preserving the partition into cuboids. The other guards simultaneously sweep their visibility region from the leftmost position to the rightmost. Since every guard lies on a reflex edge, they all have to turn by an angle of $3/2 \pi$ rad, which can be done in $\frac{3}{4}$ seconds. \square

5 Complexity of Searchability

Next we prove that 3SSP is strongly NP-hard by a reduction from 3SAT, thus converting a formula φ in 3-conjunctive normal form into an instance of 3SSP that is searchable if and only if φ is satisfiable.

Building blocks

A *variable gadget* is a cuboid with a guard lying on one edge. The two faces adjacent to the guard are called *A-side* and *B-side*, respectively. The guard itself is called *variable guard*.

A *clause gadget* is made of several parts, shown in Figure 15. The first part is a prism, shaped like a wide cuboid with 3 *cavities* on one side. On its top there is a *nook* shaped as a triangular prism lying on a side face,

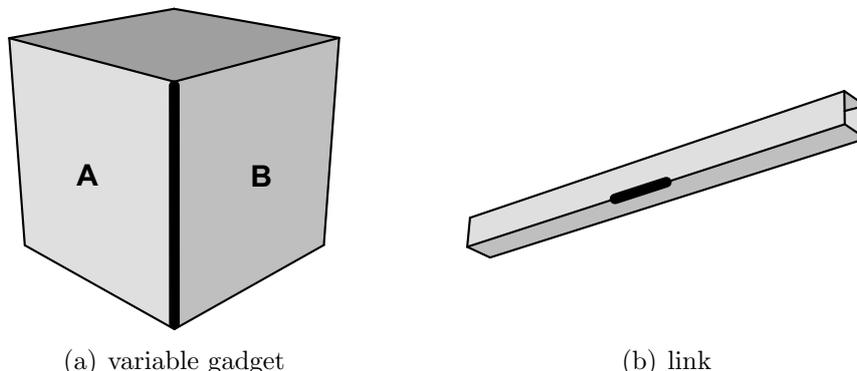


Figure 14: Two building blocks of the reduction.

with a guard on the upper edge. The guard is called *separator*, because its searchplanes partition the clause gadget in two regions. One of the two regions contains none of the 3 cavities, and its top face is called *A-side*. On the other hand, the back face of each cavity is a *B-side*, and a *literal guard* lies on the top edge of each B-side. When any literal guard is aiming at the A-side of its clause gadget, it also completely closes the nook containing the separator. We define the leftmost position of the separator to be the one that is closer to the A-side. All 3 cavities are then pairwise connected by V-shaped prisms with vertical bases. When two literal guards from the same clause gadget are both aiming at their B-sides, their searchplanes intersect in an area around the bottom of the V: such area is called *C-side*. Thus, every clause gadget has 3 B-sides and 3 C-sides, all coplanar.

The A-sides, B-sides and C-sides of all the gadgets are collectively referred as *distinguished sides*.

To connect together all the different gadgets we use structures called *links*. A link is a very thin prism with its two bases removed, and with a short *link guard* lying in the middle of an edge (see Figure 14). In the following, whenever we wish to connect two gadgets, we will cut a hole in their surfaces (where indicated from time to time), and place a link stretching from one hole to the other. The angle of incidence does not matter, as long as no guard, other than its link guard, can see inside a link. Conversely, we will arrange the links in such a way that every link guard's searchlight will not interfere with the gadget guards. So, in every gadget there will be a thin illuminated polygon jutting from each of its links, which will be easily avoidable by the intruder. As a consequence, a link can be cleared by its guard only while both its bases are *capped* by some guards lying in the adjacent gadgets.

Given a boolean formula φ in 3-conjunctive normal form, we construct a row of variable gadgets, one for every variable of φ , and a row of clause

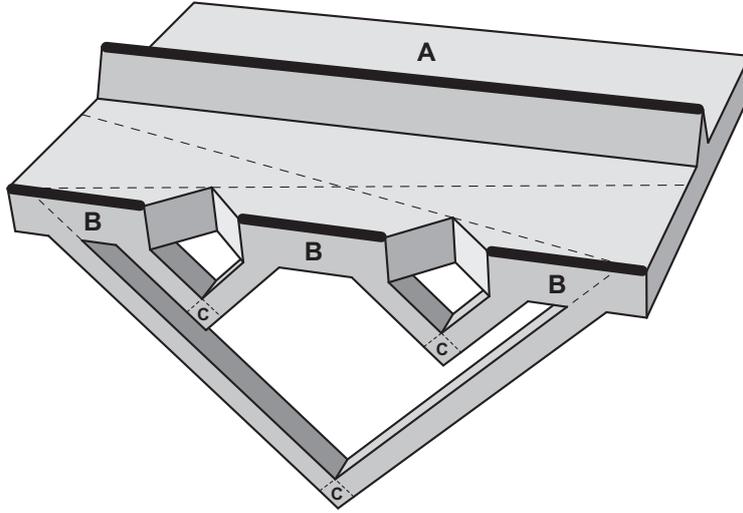


Figure 15: Clause gadget.

gadgets, one for every clause of φ . We arrange the variable gadgets so that all the A-sides are coplanar, and all the B-sides are coplanar. We arrange the clause gadgets similarly, and we place the two rows of gadgets in such a way that every distinguished side of every variable gadget can see every distinguished side of every clause gadget. We also associate the i -th B-side of a clause gadget to the i -th literal in the corresponding clause of φ , for $1 \leq i \leq 3$.

Finally we add a *bridge*, whose purpose is to prevent variable gadgets from being cleared if the corresponding literal guards do not behave “coherently,” as explained in the proof of Theorem 10. The bridge is constructed like a variable gadget, but it is not associated to any variable of φ . It is shaped as a long, thin pole, whose guard lies on one of the long edges, and it is arranged in such a way that its distinguished sides can see the B-side of every clause gadget.

Connections

Then we connect the distinguished sides of our gadgets by placing links. The exact points of connection and angles of incidence do not matter, as long as guards are non-interfering, as specified earlier, when introducing links. Mutual intersections of links will be resolved later.

Referring to Figures 16 and 17, we make the following connections.

- Connect the A-side of every clause gadget to both the A-side and the B-side of every variable gadget.

- Connect all the B-sides of every clause gadget to both the A-side and the B-side of the bridge.
- Connect all the C-sides of every clause gadget to both the A-side and the B-side of every variable gadget.
- Connect each B-side of each clause gadget to the A-side (resp. B-side) of the variable gadget corresponding to its associated literal, if that literal is negative (resp. positive) in φ .

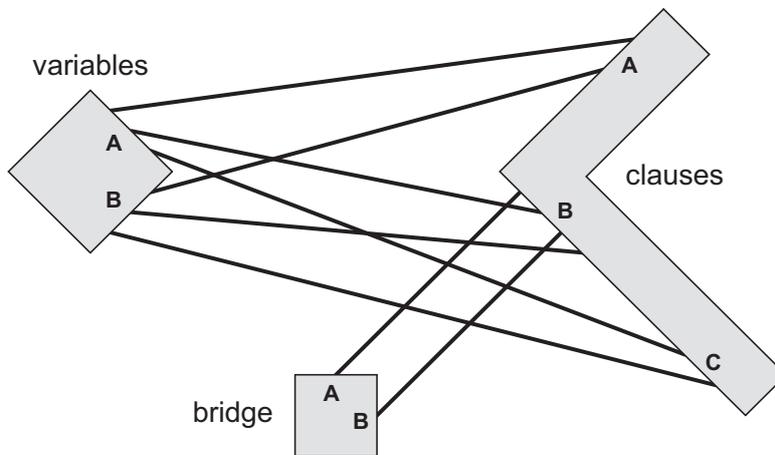


Figure 16: Relative positions of the gadgets and the bridge, with their links.

We can easily position the bridge so that it is not accidentally hit by any link running between a variable gadget and a clause gadget, such as in Figure 16. We also want links to be pairwise disjoint. To achieve this, we consider any pair of intersecting links, and shrink them while translating them slightly, until their intersection vanishes. This can be accomplished without creating new intersections with other links, for example by making sure that the “new version” of each link is always strictly contained in its “previous version.”

Reduction

Theorem 10. *The 3-DIMENSIONAL SEARCHLIGHT SCHEDULING PROBLEM is strongly NP-hard.*

Proof. Given an instance φ of 3SAT, we construct the instance of 3SSP described above. It is indeed a polyhedron, because the bridge and all the

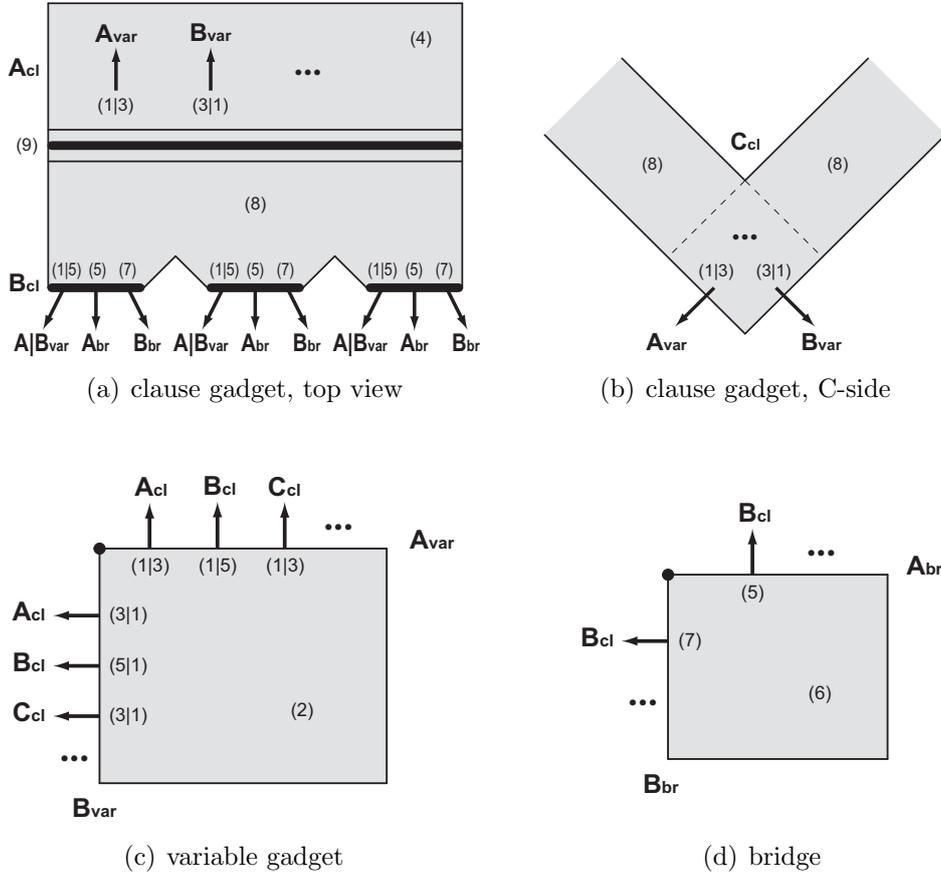


Figure 17: Connections among gadgets, and their clearing order given in Theorem 10. The abbreviation “ A_{cl} ” stands for “A-side of a clause gadget,” etc..

variable gadgets are connected to all the clause gadgets. Moreover, the number of links is quadratic in the size of φ , and we may also assume that the coordinates of every vertex are rationals, with a number of digits that is polynomial in the size of φ . Removing the intersections between links takes polynomial time as well, hence the whole construction is computable in P.

Positive instances. If φ is satisfiable, we choose a satisfying assignment for its variables and give a search schedule that clears our polyhedron. Initially we aim each variable guard at its A-side (resp. B-side) if the corresponding variable is true (resp. false) in the chosen assignment. By assumption there is at least a true literal in every clause of φ . For each clause, we pick exactly one true literal, and aim the corresponding literal guard at the A-side of its clause gadget. We aim all the other literal guards at their

respective B-sides. As a result, the A-side and the 3 C-sides of every clause gadget have the corresponding links capped by the literal guards. Finally, we aim the bridge guard at its A-side and put every separator in its leftmost position.

From this starting position, we specify a search schedule in 9 steps (refer to Figure 17).

1. Clear all the links that are capped by some variable guard. This is possible because, by construction, the other end of every such link is capped by some literal guard as well.
2. Clear every variable gadget by turning its guard. While this happens, the literal guards retain caps on their own side of the links cleared during Step 1, thus preventing recontamination.
3. Clear the remaining links connected to the A-side or to a C-side of a clause gadget. This is now possible because all the variable guards switched side in Step 2.
4. Aim at its B-side every literal guard that is currently aiming at its A-side. One half of each clause gadget gets cleared as a result, while the separator prevents the still uncapped links on the B-side from recontaminating the clear links on the A-side.
5. Clear the remaining links connected to a variable gadget, and clear the links capped by the bridge guard.
6. Clear the bridge by turning its guard to the B-side.
7. Clear the remaining links that connect the bridge with the clause gadgets.
8. Turn all the literal guards simultaneously, thus clearing the last half of each clause gadget, and capping the upper nooks. Since the three literal guards of a clause gadget are collinear, when they move together they act as a single filling guard.
9. Clear every nook by turning the separators.

When this is done, the whole polyhedron is clear, which proves that the instance of 3SSP is searchable.

Negative instances. Conversely, assuming that φ is not satisfiable, we claim that the variable gadgets can never be all simultaneously clear, no matter what the guards do.

Recall that every A-side of every clause gadget is linked to both sides of every variable gadget. Hence, as soon as the A-side of any clause gadget is not covered by at least one literal guard, all the variable gadgets get immediately recontaminated, unless they were all clear in the first place. For the same reason, no variable gadget can ever be cleared while the A-side of some clause gadget is uncovered. Similarly, if a C-side of any clause gadget is not covered by at least one literal guard, all variable gadgets get recontaminated, and none of them can be cleared.

It follows that, in order for a schedule to start clearing any variable gadget, it must ensure that each clause gadget has exactly one literal guard covering the A-side and exactly two literal guards covering the C-sides. Moreover, the literal guards that cover the A-sides must be chosen once and for all. Indeed, whenever a schedule attempts to “switch gears” in some clause gadget and cover the A-side with a different literal guard, all the variable gadgets become immediately contaminated, and the search must start over.

Suppose that a schedule selects exactly one literal guard for each clause gadget, to cover its A-side. Since φ is not satisfiable, there exist two selected literal guards ℓ_1 and ℓ_2 whose corresponding literals in φ are a positive occurrence and a negative occurrence of the same variable x . Otherwise, if all selected literals were *coherent*, setting them to true would yield a satisfying assignment for the variables of φ , which is a contradiction.

But in this case, it turns out that the variable gadget corresponding to variable x is impossible to clear. Indeed, there is a non-illuminated path connecting its A-side with its B-side, passing through the B-side of ℓ_1 , the bridge, and the B-side of ℓ_2 . Since ℓ_1 and ℓ_2 correspond to incoherent literals, their B-sides are connected to opposite sides of the same variable gadget, by construction.

Summarizing, all the variable gadgets are initially contaminated. In order to clear some of them, a schedule must first select a literal guard from each clause gadget and put it on its A-side. While that position is maintained, there is at least one variable gadget that is impossible to clear. As soon as one literal guard is moved, all the variable gadgets get recontaminated again. It follows that the variable gadgets can never be all clear at the same time, and in particular the polyhedron is unsearchable.

We gave a polynomial time reduction from 3SAT to 3SSP whose generated numerical coordinates are polynomially bounded in size, hence 3SSP is strongly NP-hard. \square

Optimization problems

Obviously, the previous theorem implies that the problems of minimizing search time and minimizing *total angular movement* are both NP-hard to approximate. The first problem stays NP-hard even when restricted to searchable instances.[7] By further inspecting the construction given in that paper, it is clear that the problem does not even have a PTAS, unless $P = NP$. Indeed, satisfiable boolean formulas are transformed into polyhedra searchable in 3 seconds, while the unsatisfiable ones are transformed into polyhedra that are unsearchable in $3 + \varepsilon$ seconds, for a suitable small-enough $\varepsilon > 0$. On the other hand, similar results can be obtained also for the problem of minimizing total angular movement. There are several ways to rearrange the links in the construction employed in Theorem 10, so that the unsatisfiable boolean formulas are mapped into polyhedra that are indeed searchable, but only with very demanding schedules.

6 Concluding Remarks

Summary

We modeled the 3-DIMENSIONAL SEARCHLIGHT SCHEDULING PROBLEM as a searching problem in polyhedra, where guards are line segments emanating orientable half-planes of light. We showed that filling guards act as the natural counterparts of boundary guards in the 2-dimensional version of the problem, in that the main positive results about SSP still hold for 3SSP restricted to instances with just filling guards. By further exploiting the concept of filling guard, we characterized the searchable instances of 3SSP with just one guard. Then we proved that r^2 boundary guards are sufficient to search any polyhedron with $r > 0$ reflex edges, and that just r guards lying on reflex edges are sufficient to search an orthogonal polyhedron, thus partially settling Conjecture 1 in Ref. 7 (i.e., that any polyhedron with a guard on each reflex edge is searchable). We also discussed some methods to speed up the search time by placing additional guards. Finally, we proved that deciding searchability of an instance of 3SSP is strongly NP-hard, and briefly discussed the hardness of approximation of two related optimization problems.

Further work

Several problems remain open. Settling Conjecture 1 in Ref. 7 would be the main priority. This has been accomplished for orthogonal polyhedra in

Theorem 8, but for general polyhedra it turns out to be a surprisingly deep problem. One way to lower the quadratic bound on the number of guards would be to slightly modify the partition used in Theorem 7. Instead of aiming the guards at the angle bisectors of the reflex edges, we could aim them in any direction, cut the polyhedron with the “extended” searchplanes, and still eliminate all the reflex edges. The advantage would be that, by carefully choosing a plane for each reflex edge that minimizes the intersections with other reflex edges, the overall number of intersections could be significantly less than quadratic. Even if it is still quadratic in the worst case, it is likely much lower on average, with respect to any reasonable probability measure over polyhedra.

In the case of orthogonal polyhedra, the upper bound of r guards given by Theorem 8 is also not known to be optimal. In fact, we believe that it can still be lowered by a constant factor.

The search time of the schedules given in Theorems 7 and 8 could be dramatically reduced by clearing several regions in parallel. For instance, in Theorem 8 we could turn in concert two guards whose fences do not bound a same cuboid. Generalizing, we could construct a graph G on the set of reflex edges, with an edge for every pair of reflex edges whose fences bound a same cuboid. Then the search time would be proportional to the chromatic number $\chi(G)$, which is likely sublinear, at least on average.

The complexity of 3SSP could be considered also for a restricted set of instances, such as genus-zero polyhedra or orthogonal polyhedra. Moreover, the technique used for Theorem 10 could perhaps suggest a way to prove the NP-hardness of SSP. Adding new elements to SSP in order to increase its expressiveness, while preserving its 2-dimensional nature, could be an intermediate step. For example, we could introduce *curtains*, i.e., line segments that block visibility but do not block movement. Then we could attempt to replicate the construction of Theorem 10 for this extension of SSP.

Similarly, we may try to modify the construction used in Ref. 6 to prove the PSPACE-hardness of 3SSP as well. We observe that such a construction, suitably extruded and regarded as an instance of 3SSP, is not guard-visible, as in Definition 9. As a matter of fact, incorporating regions that can never be cleared is an effective way to force the recontamination of other critical regions when certain conditions are met. However, if we want our constructions to be guard-visible instances of 3SSP, no such expedient is of any use, and cleverer tools have to be devised.

Observe that, throughout the paper, we focused primarily on boundary guards (with the exception of Theorem 4, which holds also for general guards). This is certainly a bonus for Theorems 7, 8, and 10, in which restricting to boundary guards yields stronger statements. On the other hand,

extending the concept of filling guard (Definition 11) to guards not lying on the boundary may lead to interesting generalizations of Theorem 2.

Acknowledgements

The author wishes to thank Joseph O’Rourke and the anonymous reviewer for their insightful comments.

This work was supported in part by MIUR of Italy under project Algo-DEEP prot. 2008TFBWL4.

References

- [1] K. Sugihara, I. Suzuki, and M. Yamashita, The searchlight scheduling problem, *SIAM J. Comput.* **19** (1990) 1024–1040.
- [2] J. O’Rourke, *Art gallery theorems and algorithms* (Oxford University Press, New York, 1987).
- [3] M. Yamashita, Private communication (Aug. 2010).
- [4] M. Yamashita, I. Suzuki, and T. Kameda, Searching a polygonal region by a group of stationary k -searchers, *Inform. Process. Lett.* **92** (2004) 1–8.
- [5] K. J. Obermeyer, A. Ganguli, and F. Bullo, A complete algorithm for searchlight scheduling, *Int. J. Comput. Geom. Ap.* **21** (2011) 101–130.
- [6] G. Viglietta, Partial searchlight scheduling is strongly PSPACE-complete, in *Proc. 28th Eur. Workshop Comput. Geom.*, Assisi (Mar. 2012) pp. 101–104.
- [7] G. Viglietta and M. Monge, The 3-dimensional searchlight scheduling problem, in *Proc. 22nd Canadian Conf. Comput. Geom.*, Winnipeg, MB (Aug. 2010) pp. 9–12.
- [8] D. T. Lee and A. K. Lin, Computational complexity of art gallery problems, *IEEE T. Inform. Theory* **32** (1986) 276–282.
- [9] B. Chazelle, Convex partitions of polyhedra: a lower bound and worst-case optimal algorithm, *SIAM J. Comput.* **13** (1984) 488–507.
- [10] B. Chazelle and L. Palios, Triangulating a nonconvex polytope, *Discrete Comput. Geom.* **5** (1990) 505–526.