Coordinated Search With Multiple Robots Arranged in Line Formations

Andreas Kolling, *Member, IEEE,* Alexander Kleiner, *Member, IEEE,* and Stefano Carpin, *Senior Member, IEEE*

***Abstract*—In this paper we address the problem of detecting intruders in complex bidimensional environments with a team of robots arranged in line formations called *sweep lines*. Sweep lines are used to coordinate the motion of multiple robots and guarantee the detection of any number of arbitrarily fast intruders, even when each robot has a limited sensor footprint. We present a formalization of the problem, coined *Line-Clear*, which requires the computation of *sweep schedules* to coordinate the motion of multiple sweep lines using the fewest robots possible. We provide a proof of NP-hardness of the general Line-Clear problem based on results from graph-searching. An algorithm to compute sweep schedules for simply-connected environments, which additionally guarantees that the cleared area is connected and not recontaminated, is then presented. We analyze its complexity formally and in simulation experiments and present solutions for a number of subproblems required for an implementation of the algorithm. The analysis provides a formal criterion for when the algorithm runs in polynomial time and the experiments indicate that this criterion may be satisfied for most environments in practice.**

I. Introduction

This paper considers the problem of detecting all intruders in a bounded, planar environment with obstacles using a team of robots with a limited sensor footprint. Our goal is to develop methods with guaranteed performance, i.e., all intruders shall be detected in finite time irrespective of their capability for evasive maneuvers. If each robotic searcher is equipped with a sensor covering only a small subset of the environment, in order to successfully complete this task robots need to tightly coordinate their actions (see Figure 1). The concept of a *sweep line* has been extensively used in computational geometry [1] and is here adopted as the building block for robot cooperation. Robots arrange themselves in configurations such that the union of their sensor footprints covers a set of sweep lines that are then moved through the environment. This arrangement is relatively easy to implement in practice, and has found extensive applications in search and rescue efforts by human teams. In the past we have investigated this paradigm for the problem where multiple robots are tasked with the detection of all intruders in an indoor environment. In [2] we have shown that given an occupancy grid of the environment it is possible to algorithmically determine *sweep schedules* for lines, i.e., a temporal sequence of coordinated sweeps such that all intruders are eventually detected. We have also presented a system

Andreas Kolling is with iRobot Corporation, Pasadena, CA, USA, e-mail: akolling@irobot.com (corresponding author).

Alexander Kleiner is with FaceMap LLC, Malibu, CA, USA, e- mail: a.kleiner@facemap.com.

Stefano Carpin is with the University of California, Merced, CA, USA, e-mail: scarpin@ucmerced.edu.

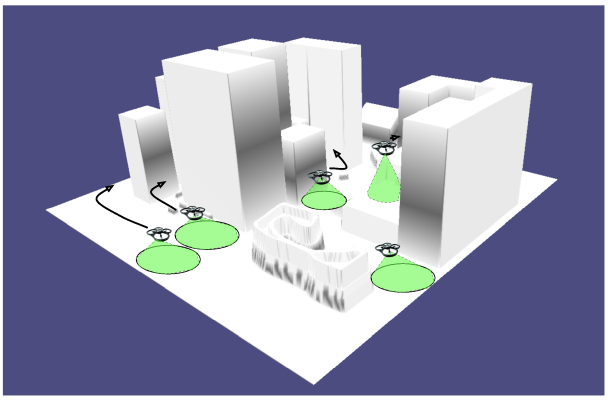

Fig. 1. An illustration of the motivation for the formal model presented in this paper and applied to a 2.5D environment, as in [4], [5]. Multiple unmanned aerial vehicles are clearing an environment by following moving sweeps lines and covering them with their sensors. The coordination of these movements is essential for an efficient search strategy that uses fewer searchers. The figure shows that a Line-Clear approach can be used also for cases like the one depicted in the figure, where the sensors are not bound to the plane, but their footprint can be abstracted with a line in the plane.

where robots achieve the same objective even if they are not given any map [3] upfront and can only form sweep lines by sensing their neighbors and obstacle boundaries locally. A variant that considers 2.5D environments and unmanned aerial vehicles has been presented in [4], including an experimental validation. Initial attempts to optimize not only the number of robots but also the time for sweep schedules have been presented in [5], including experiments with simulated and real unmanned aerial vehicles. This prior work confirmed the practical importance of the sweep line paradigm in robotic search, but a rigorous formal analysis was missing. In this manuscript we fill this gap by providing a thorough theoretical analysis of the approach to clear large environments using robots arranged on moving sweep lines. We dub this strategy *Line-Clear*. The contributions of this paper are the following:

- the Line-Clear problem is defined and formalized (Section III);
- we prove that Line-Clear is NP-hard in Section IV;
- we analyze the special case of simply-connected environments, i.e., possibly non-convex environments with no holes, and we show that in this case the Line-Clear problem is equivalent to a combinatorial problem whose solution determines how to move sweep lines, where to set them up, and when to remove them (Section V);
- we show how to solve the combinatorial problem and

compute sweep line schedules in simply-connected environments (Section VI);
- we provide solutions for the subproblem of computing shortest sweep lines and show experimental support for our conjecture that connected and monotone Line-Clear in simply-connected environments can be solved in polynomial time.

Not all questions we raise in this paper can be answered at this time. We restrict our attention to sweep schedules that are *connected* and *monotone*. This means that the area of the environment that has been cleared from intruders is connected (in a topological sense) and that every area is cleared only once (monotone). For other related pursuit-evasion problems it has been shown that imposing monotonicity can lead to solutions with higher cost but for others this is not the case (see Section II). So far we have not investigated how lifting these requirements would impact the algorithms we developed.

II. Related work

Our work is connected to a wide range of applications, such as search, exploration, tracking, and surveillance. Each of these has a long history in the robotics literature, and is often connected to graph theory and computational geometry. A comprehensive review is beyond the scope of this section, and we provide only pointers to selected relevant contributions. We focus on visibility-based pursuit-evasion as a natural predecessor to our work and we also discuss other approaches for search, pursuit and capture that arrange robots on lines, as well as graph-based search models. Since our approach is deterministic, we skip the rich body of literature related to probabilistic approaches. From a robotics perspective, surveys such as [6], [7], and [8] provide an excellent overview of the different variants and assumptions that are made with regard to search problems. An intereseting recent tutorial was also presented in [9].

One of the first graph-based pursuit-evasion models, also know as graph-searching, was introduced by Parsons [10]. In graph-search a number of searchers moves along edges to catch an omniscient intruder with unbounded speed. The concept of *contamination* is introduced to represent the possibility of the intruder being located in a part of the graph. If an area is not contaminated it is said to be *clear*, and the goal then becomes to turn all contaminated areas into clear areas using the least number of searchers. A great deal of work has been done in this domain and an overview of results can be found in [11] and [12]. Starting from this model, numerous variations were proposed. Node search, mixed search, and connected search consider contamination on edges or vertices, different capture conditions such as being on the same vertex or on adjacent vertices, or require the cleared parts of the graph to be connected. The most prominent question for these graph-based problems usually relates to the number of searchers needed to clear the graph. This value is commonly referred to as the *search number*. The first complexity result is due to Meggido et al. [13] who proved that finding the search number for the model from [10] is NP-hard. Another important question is whether recontamination matters for optimal solutions. Recontamination occurs when a previously cleared area becomes again contaminated during the search. It is then interesting to know whether one can always find a strategy that does not generate recontamination and only uses as many searchers as given by the search number of the graph. Graph-searching models that have this property are called monotone and Parson's model was shown to be monotone in [14]. It is worth noting that connected edge-searching is not monotone as shown in [15], and connectedness comes at an increased cost, but this increase is conjectured to be no more than two searchers [12] for any graph. For trees, however, requiring connectedness will not lead to more costly strategies. There is also a rich vein of results connecting global graph parameters to search numbers of different search variants. Parameters such as cutwidth and treewidth also play a role in the monotonicity proofs, culminating into a generalization of such proofs given in [16]. Graph-based models have found various applications in robotic systems, such as in [17]–[20] which used and modified existing edge-searching and node-searching models. A new graph-based model for robotic applications, called Graph-Clear, was developed in [2], [3], [21], [22] in which vertices are cleared and edges are blocked to prevent recontamination. Therein edges and vertices have weights representing the number of robots needed to clear a vertex or prevent recontamination through an edge. The problem of finding search strategies was shown to NP-hard on graphs, but for trees with n vertices an $O(n^2)$ monotone algorithm was shown to exist. Interestingly, the weighted version of edge-searching, introduced in [23], turns out to be NP-hard even on trees as shown in [24] and not polynomial time as formerly reported in [23].

In robotics, pursuit-evasion has primarily been studied in the form of visibility-based pursuit-evasion and with an emphasis on single searchers with infinite line of sight. In this setting an intruder is detected when it falls within the unlimited range sensor of a pursuer in a 2D environment. The field was pioneered by Suzuki and Yamashita [25] who considered polygonal environments only, and a sensor model (*flashlight*) with unlimited range but incapable of detecting intruders behind obstacles. They introduced the concept of k-searcher, where k is the number of sensing beams emitted by the searcher, with the special case of an ∞-searcher being a searcher with an omnidirectional sensor with infinite range. As for graphs, a 2D environment also has a search number, i.e., the minimum number of searchers needed to detect all intruders. Also similar to graphs, the possibility of an intruder being located in the environment is often represented by contamination that is then cleared by searchers. The visibility-based pursuit-evasion problem for a single ∞-searcher was solved by LaValle et al. [26]. The problem of finding the minimal number of ∞-searchers for any polygon is proven to be NP-hard in [27]. The authors also show that recontamination is sometimes useful in order to find the optimal solution, i.e., visibility-based pursuit-evasion with a single searcher is not monotone. An important result regarding the required capability to search environments was presented in [28] and it states that a searcher that can only detect gaps, i.e., discontinuities in the environment, and move towards these gaps is sufficiently capable to clear any

environment that a more powerful robot can. Extending complete and optimal visibility-based pursuit-evasion algorithms to multiple searchers has proven to be challenging. The approach presented in [29] uses a cylindrical algebraic decomposition to generate strategies for multiple pursuers, but the best upper bound for the computational complexity of this approach is doubly exponential in the number of agents. Related results were presented in the most recent extension [30].

Other line-based approaches have been investigated as well. Obermeyer et al. consider the problem of detecting intruders with a sweeping ray [31]. However, the problem they study is restricted since the source of the ray is stationary, whereas in this paper we consider lines where both extremities can be moved. The contribution most related to the problem we study in this paper was presented by Efrat et al. [32]. Therein the authors study a method where multiple robots, each with an unlimited range sensor, are arranged in a single movable polygonal chain operating in a simply-connected polygon with n vertices. The algorithm to compute motion strategies in [32] runs in $O(n^3)$ and improved algorithms developed in [33] run in $O(n \log n)$ In this paper, on the contrary, we consider the case where robots have limited sensing ranges and multiple polygonal lines are used to sweep the environment. We further allow the number of lines to vary as the search mission unfolds. An interesting approach that also assumes limited sensing range is presented in [34] where the authors present a system where multiple robots trap faster intruders, once they detect them, by forming a surrounding chain. In these kinds of capture or direct pursuit problems, similar to differential games [35] like the lion and man problem, time obviously play a crucial role. In this paper we do not consider any notion of time, and as pointed out in [6] this is one of the important open research questions with regard to robotic search in complex environments and large teams. A close approach to ours in terms of considering limited range sensor is found in [36]. Therein the authors present a distributed algorithm to clear an environment with a team of robots with limited range and limited field of view sensors. The key contribution is the allocation of robots to the frontier between cleared and contaminated areas that considers the geometry of locally visible areas in order to find good locations for newly arriving robots at a frontier. One key distinction is that the robots do not discover the overall topology of the environment and cannot plan strategies that minimize the number of robots needed to clear the environment. The bound provided for the number of robots that may be required for the centralized version of the algorithm in [36] is simply given by the number of viewpoints that it requires to be occupied.

III. Line-Clear: Definitions and Mathematical Preliminaries

In this section we define the search model. The reader is assumed to be familiar with basic computational geometry concepts and is referred to [1] for a comprehensive introduction to this topic. Our assumptions are similar to those used in other search and pursuit-evasion problems with guarantees, i.e., we search for an unknown number of targets that are omniscient, arbitrarily fast, and actively try to avoid being detected. Targets are omniscient in the sense that they have full knowledge of the environment and of the locations of all searchers. The presence of these targets is modeled with the concept of contamination. In the following we will first define the environment and then an abstract model for the sensing capabilities of the robot team, based on so-called *sweep lines*. We then define contamination, how it is cleared by sweep lines, and formally introduce the Line-Clear problem. Figures 2 and 3 illustrate some of the symbols and concepts we define in the following. The reader is invited to refer to this figure as the various definitions are provided.

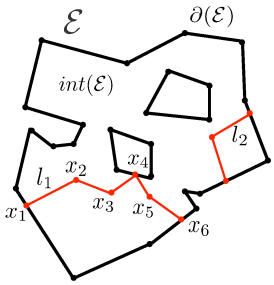

Fig. 2. Illustrations of the environment and sweep lines. The meaning of the various symbols used in the figure is given in the remainder of this section.

The environment is a bounded, closed, connected set $\mathcal{E} \subset \mathbb{R}^2$ with a boundary $\partial\mathcal{E}$ consisting of $k \geq 1$ simple polygons[1], one of which is the exterior polygon that contains all other $k-1$ polygons. In Fig. 2, $\mathcal{E}$ is bounded by three polygons, i.e., it has two holes. Let $\partial(\mathcal{E})$ be its boundary and $int(\mathcal{E})$ be its interior. In the following, given $x_i, x_j \in \mathbb{R}^2$, let $[x_i, x_j]$ be the closed segment between x_i and x_j and let (x_i, x_j) be the open segment between the same points. The ability of a team of robots to detect targets with their sensors is modeled by *sweep lines*, defined as follows:

Definition 1 (Sweep Line)**.** *A sweep line l is an ordered list of $n > 1$ points in $\mathbb{R}^2$ written $l = [x_1, \ldots, x_n]$ subject to the following constraints:*

1) $x_1, x_n \in \partial\mathcal{E}$*;*
2) *all open segments $(x_k, x_{k+1}), 1 \leq k \leq n-1$ do not intersect with each other, and $(x_k, x_{k+1}) \subset \mathcal{E}$.*[2]

The set of all sweep lines with n points is indicated as $S(n)$.

We now formalize the movement and coordination of multiple sweep lines by defining *moving sweep lines* and a *sweep schedule*. Figure 3 illustrates the idea of moving sweep lines that are coordinated in a sweep schedule to clear an environment. Subfigure 1 shows an environment partitioned into clear (gray) and contaminated regions (white). As a sweep line moves the cleared region grows (subfigure 2). The sweep schedule then replaces a single moving line with two sweep lines (subfigure 3) that can move separately. Our final objective is to determine a *sweep schedule* that will remove all contamination for $\mathcal{E}$.

[1]We recall that a simple polygon is a region bounded by a single polygonal chain that does not intersects itself.

[2]Note that this definition allows segments to share endpoints.

Definition 2 (Moving Sweep Line)**.** *A moving sweep line is a function $l : \mathbb{R}^+ \to S(n)$ with $l(t) = [x_1(t), \ldots, x_n(t)]$ such that $x_i(t)$ is continuous in t $\forall i \in \{1, \ldots, n\}$.*

Definition 3 (Sweep Schedule)**.** *A sweep schedule is a function $\tau : [0, T_f] \to \mathcal{P}(\mathcal{S})$, where $\mathcal{P}(\mathcal{S})$ is the powerset of all sweep lines with any number of segments.*

Note that the specific value of T_f (final time) in the above definition is immaterial to the remaining discussion. The definition of sweep schedules above is rather unconstrained as it can map into any set of sweep lines. For example, a set of k moving sweep lines $l_1(t), l_2(t), \ldots, l_k(t)$ defines the sweep schedule $\tau(t) = l_1(t) \cup \ldots \cup l_k(t)$. But we can also add and remove moving sweep lines to a sweep schedule. Constraints on sweep schedules will arise later through the requirement that they clear an environment, e.g., continuity is necessary to avoid *discrete jumps* that prevent clearing. For this we will have to define contamination and how it is cleared by a sweep schedule. Intuitively a sweep schedule should be thought of as maintaining and expanding a cleared area with sweep lines, controlling their addition, motion, splitting, and removal (as shown in Fig. 3).

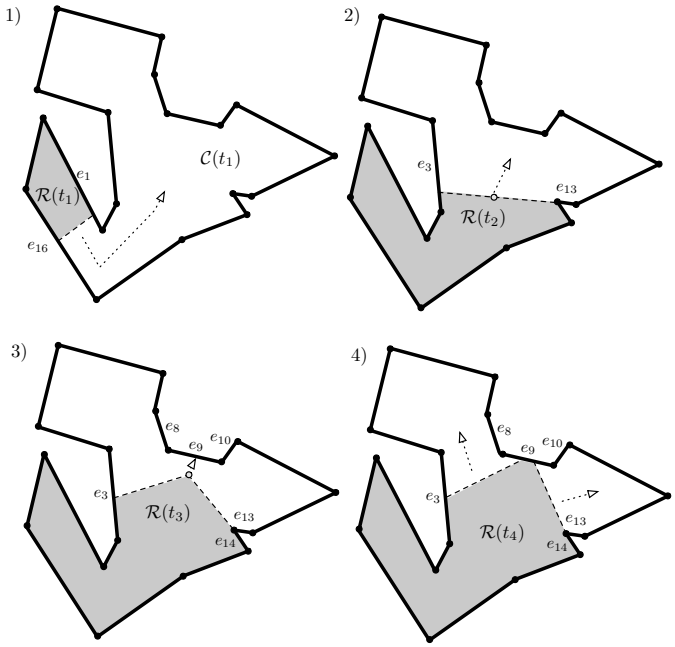

Fig. 3. Four snapshots of a sweep schedule at times t_1, t_2, t_3, and t_4 showing how a sweep line (dashed line) moves forward, extends the cleared area $\mathcal{R}(t)$, is replaced by a sweep line with a midpoint at time t_3 which then splits on e_9 and is replaced by two sweep lines between e_9 and e_3 and e_9 and e_{13} which can now move independently. The boundary is drawn with thick lines.

In order to relate a sweep schedule to contamination we define coverage sets for sweep lines which follow from the interpretation of sweep lines as chains of line segments.

Definition 4 (Coverage sets)**.** *Let $l = [x_1, \ldots, x_n]$ be a sweep line. Its* coverage set *$\bar{P}(l) \subset \mathcal{E}$ is the part of $\mathcal{E}$ covered by its $n-1$ segments. If $S = \{l_1, \ldots, l_k\}$ is a finite set of sweep lines, then its coverage set is $\bar{P}(S) = \bigcup_{i=1}^{k} \bar{P}(l_i)$.*

Referring to Figure 2, $\bar{P}(l_1)$ is given by its five segments, whereas $\bar{P}(\{l_1, l_2\})$ is given by all the segments shown in the figure. Moreover, as $\tau(t)$ is a set of sweep lines at time t (definition 3), it is possible to consider $\bar{P}(\tau(t))$, i.e., the coverage set of the sweep lines defined by the sweeping schedule at time t.)

We can now define contamination and disallow it from crossing the coverage set of a sweep schedule.

Definition 5 (Clear and contaminated points)**.** *For each time t a point in $\mathcal{E}$ is either* clear *or* contaminated*. $\mathcal{R}(t)$ is the set of all clear points at time t and $\mathcal{C}(t)$ is the set of contaminated points.*

A point $p \in \mathcal{E}$ is *contaminated* if a target may be located at p. Conversely, $p \in \mathcal{E}$ is *cleared* if it is known that no target is located at p. Consistent with the hypothesis that targets are omniscient and travel at unbounded speeds, as soon as a chance of recontamination arises, targets immediately take action and previously cleared points can become recontaminated. In other words the boundary of the set of cleared points must be continually guarded with sweep lines to prevent recontamination. The following definitions formalize this idea.

Definition 6 (Contamination path)**.** *Let x be a point in $\mathcal{E}$ and $\tau(t)$ the set of sweep lines at time t for schedule τ. A contamination path for x at time t is a path*[3] *between x and a point $y \in \mathcal{C}(t)$ that does not intersect $\bar{P}(\tau(t))$.*

A contamination path for a point models the existence of a path between the point and the contaminated region that does not go through the coverage set of a sweep schedule, i.e., the area guarded robots. As common in the literature, if there is a contamination path, then the point becomes contaminated itself. Analogously, a point that is in the coverage set is considered clear until it is recontaminated. Therefore, as illustrated in figure 3, the boundary between $\mathcal{R}(t)$ and $\mathcal{C}(t)$ is given by $\bar{P}(S)$ where S is the set of sweep lines at time t. The next definition formalizes this.

Definition 7 (Clearing and recontamination)**.** *A point x is clear at time t if $x \in \bar{P}(\tau(t))$. A point x is contaminated at time t if there is a contamination path at time t. A point x is recontaminated if it is clear at time t and contaminated at time $t' > t$.*

The purpose of sweep lines is to represent the ability of a team of robots to guard a region of space with one or more robots, such that no intruder can cross without being detected. The cost of a sweep line then represents the number of robots that a particular sweep line requires.

Definition 8 (Cost of Sweep Lines)**.** *Let l be a sweep line. Its cost $c(l)$ is a non-decreasing integer function of its length, where the length of a sweep line is the sum of the lengths of its segments. That is to say that if l_1 is a sweep line longer than l_2, then $c(l_1) \geq c(l_2)$. For a finite set of sweeplines S, its cost is the sum of the costs of its individual sweeplines. For a sweep schedule τ define the cost $c(\tau) = \max_{t \in [0,T]}\{c(\tau(t))\}$.*

[3]A path between two points x_1 and x_2 is defined as a continuous function $f : [0, 1] \to \mathcal{E}$ with $f(0) = x_1$ and $f(1) = x_2$.

The cost $c(\tau)$ represents the minimum number of robots needed to ensure that every sweep line in the sweep schedule can be covered with sensors. The precise number depends on the sensors used to cover the sweep line and this generic formulation allows to tackle the problem without committing to a specific sensor type. Our only assumption is that the cost is a non-decreasing function of the sweep line length. This formalizes the intuition that as a sweep line gets longer one needs at best the same, but potentially more, robots to cover it.

The Line-Clear problem asks for sweep schedules that clear an initially contaminated $\mathcal{E}$ at the lowest possible cost. We also introduce two additional properties of sweep schedules, namely connectedness and monotonicity, and allow variants of the Line-Clear problem that are restricted to sweep schedules with these properties.

Definition 9 (Connected and monotone sweep schedules)**.** *Let τ be a sweep schedule that clears $\mathcal{E}$. If $\mathcal{R}(t)$ is a connected set $\forall t \in [0, T_f]$, then τ is connected. If $t \leq t' \Rightarrow \mathcal{R}(t) \subseteq \mathcal{R}(t')$, then τ is monotone.*

Definition 10 (The Line-Clear problem)**.** *The Line-Clear problem is to find an optimal sweep schedule*

$$\tau_{opt} := \arg\min_{\tau} c(\tau).$$

such that $\mathcal{R}(T_f) = \mathcal{E}$. The connected Line-Clear problem additionally requires τ_{opt} to be connected, and its optimal solution is indicated as $\tau_{con,opt}$. We define the Line-Clear number of $\mathcal{E}$ $lc(\mathcal{E}) := c(\tau_{opt})$ and the connected Line-Clear number as $clc(\mathcal{E}) := c(\tau_{con,opt})$.

IV. Computational Complexity

We show that Line-Clear is NP-hard by establishing a correspondence between Line-Clear and edge-searching. Edge-searching is one of the more researched pursuit-evasion problems on graphs [10] and is defined as follows. Given a graph G, an intruder moves along its edges while a team of searchers tries to capture it. Searchers can perform three operations: 1) position themselves on a vertex, 2) move along an edge, and 3) move away from a vertex. The intruder is captured when it visits a vertex where a searcher is positioned, or when it is located on an edge along which a searcher is moving. The intruder is assumed to be omniscient whereas searchers just know the structure of the graph G. The goal of the searchers is to determine a strategy S, i.e., a sequence of the three operations described above, that guarantees capture of the intruder. This problem can also be modeled with contamination that is cleared by a strategy, with contamination residing in the edges of the graph. The search number of a graph, $s(G)$, is the minimum number of searchers required to capture an intruder in G. For a given integer K and graph G, determining whether $s(G) \leq K$ was shown to be NP-complete [13], [14]. It is also known that the problem is NP-complete even if one restricts G to be a planar graph with vertex degree at most three [37]. Let G^3 be such a planar graph with maximum vertex degree at most three. For this graph we will now construct an instance of the Line-Clear problem and relate the Line-Clear number $lc(\mathcal{E})$ to $s(G^3)$.

For a given G^3 we construct the associated Line-Clear environment $\mathcal{E}^3$ by combining three different elementary components called *intersection*, *corridor*, and *dead-end*. Every vertex of degree 3 is associated to an intersection in $\mathcal{E}^3$, every vertex of degree 2 to a corridor, and every vertex of degree 1 to a dead-end. These three elementary components are defined as shown in Fig. 4. These components rely on two parameters, d and l, that are constrained as follows. For a sweep line l_1 of length d going through a corridor we require that $c(l_1) = 1$. This ensures that corridors are narrow enough to be cleared at cost 1. For a sweep line l_2 of length l, we require that $c(l_2) > s(G^3)$. This ensures sufficient separation between the vertices and their areas $A(v)$ shown in Figure 4, since any sweep line intersecting $A(v_1)$ and $A(v_2)$ for two vertices v_1 and v_2 would have cost larger than $s(G^3)$. Now, given a planar embedding of G^3 we can replace every vertex with its associated component and where necessary extend their length beyond l. For every edge between two vertices in G^3 we now have a corridor in $\mathcal{E}^3$ with length at least $2 \cdot l$. Given an edge $e = (v, v')$ we write $A(e) \subset \mathcal{E}^3$ for the area of the corridor and $A(v) \subset \mathcal{E}^3$ for the area of the vertex v between corridors, exactly as shown in Fig. 4. It is straightforward to verify that from a planar embedding of G^3 we can use the above components to construct $\mathcal{E}^3$ without intersections. In such case we say that $\mathcal{E}^3$ is the environment associated to G^3.

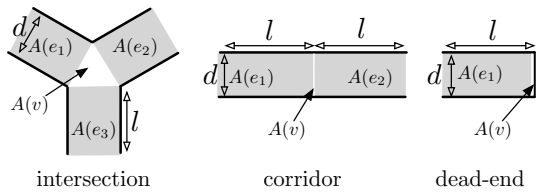

Fig. 4. The three components for the construction of $\mathcal{E}^3$ from G^3. Areas associated to edges are marked in grey and areas associated to vertices are marked in white. For simplicity we show straight corridors but turns are allowed given that their width does not exceed d.

The following theorem formalizes the equivalence between a sweeping schedule in $\mathcal{E}^3$ and a strategy in G^3. Its proof, clarifying the remaining details in the construction outlined above, is given in the appendix.

Theorem 1. *Let G^3 be a planar graph whose vertex degree is at most 3 and let $\mathcal{E}^3$ be its associated environment. Then $s(G^3) = lc(\mathcal{E}^3)$.*

The above theorem immediately leads to the following corollary considering how an instance of edge-searching on G^3 can be reduced to an instance of Line-Clear on $\mathcal{E}^3$.

Corollary 1. *Recalling that edge-searching is NP-complete on graphs of type G^3 the NP-hardness of Line-Clear follows.*

Note that the converse, a reduction of Line-Clear to instances of edge-searching, is far more challenging. Attempts at using edge-searching algorithms to develop algorithms for Line-Clear, although not described here, have proven to be thus far unproductive. Why this is the case will become clearer

as we proceed to solve the Line-Clear problem in a more restricted setting in the next sections.

V. Reduction to a Combinatorial Problem

Having established the NP-hardness of the Line-Clear problem for the general case of multiply-connected environments, we switch to considering the computation of monotone sweep schedules for the connected Line-Clear problem for the simpler case of simply-connected environments, i.e., environments without holes and one exterior polygon describing the boundary. The key insight to solve this special case is in observing that one only needs to keep track of the sequence in which polygon edges on the boundary of the environment are cleared. This allows us to ignore the continuous nature of the original problem and focus on a simpler set of critical moments that determine the cost of sweep schedules. To demonstrate this we will first have to introduce the concept of *choice sets* to keep track of the remaining contaminated edges on the boundary of the environment. We then show that any sweep schedule has to satisfy lower bounds that are expressed in terms of choice sets. This shows that choice sets can capture the cost relevant structure of the problem. Consequently, the remaining sections will be focused around choice sets.

Let $\mathcal{E}_s$ be a simply-connected environment for the connected Line-Clear problem and let $\{v_1, \dots, v_n\}$ be the set of vertices defining its outer polygonal boundary, with edges written $e_i = [v_i, v_{i+1}]$ (with $e_n = [v_n, v_1]$). We refer to an index i as an *obstacle index*. For notational convenience, indices for vertices and edges are modulo n, i.e., we identify $n+1$ with 1, $i+n$ with i, and so on. We call an edge e_i cleared if *any* point on the segment $[v_i, v_{i+1}]$ is cleared.

Now, consider the progression of a sweep schedule that expands a connected $\mathcal{R}(t)$ without recontamination, as shown in Figure 5 for some time t. Note that each connected contaminated component (two such components are shown in Figure 5) has on its boundary a sequence of contaminated edges with consecutive indices. A sweep schedule that clears $\mathcal{E}_s$ will eventually have to choose another contaminated edge to be cleared when expanding $\mathcal{R}(t)$. In Figure 5 this choice could be an edge from either connected component, i.e., an edge e_i with index $i \in \{4,5,6,7\}$ or $i \in \{11,12\}$. The following definition of choice sets generalizes this observation to any sweep schedule at any time t.

Definition 11 (Choice Set)**.** *For any $k = 1, \dots, n$ and $i = 1, \dots, n$ let*

$$T_k^i := \{i, i+1, \dots, i+k-1\} \subseteq \{1, \dots, n\}$$

and call it a choice set*. For $k = 0$ let $T_0^i = T_0 = \emptyset$ and call them* empty choice set*.*

In essence T_k^i is a sequence of k integers starting at i and *wrapping around* n. For example, for $n = 6$, $i = 5, k = 4$, we obtain $T_k^i = \{5, 6, 1, 2\}$. The important part is that we can now associate any connected contaminated component of $\mathcal{E}_s$ at some time t with a choice set T_k^i. For convenience, we write $\mathcal{C}(T_k^i)$ for the connected contaminated component represented by T_k^i, as shown in Figure 5. Note that T_k^i can represent connected contaminated components of slightly varying shapes as long as it contains all the contaminated edges, a sweep line with endpoints on e_{i-1} and e_{i+k} and potentially a subset of the cleared edges e_{i-1} and e_{i+k}.

Let $\mathcal{T}(t)$ be all the choice sets associated with the connected components of $\mathcal{C}(t)$. Without loss of generality let exactly one edge be cleared at each time $t_1 < t_2 < \dots < t_n$. Consequently, the different sets of choice sets can be enumerated as $\mathcal{T}(t_0) = \{T_n^1\}, \mathcal{T}(t_1), \dots, \mathcal{T}(t_n) = \{T_0\}$, starting at t_0 with all edges contaminated and ending at t_n with all edges cleared. Note that this is uniquely determined by the sequence in which edges are cleared which we shall write as $o_1, \dots, o_n$, with each $o_j \in \{1, \dots, n\}$ being the index for edge e_{o_j}.

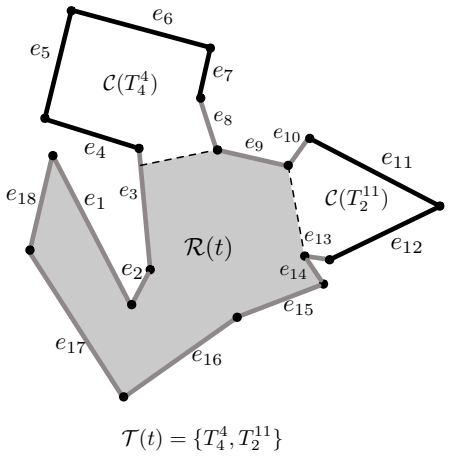

Fig. 5. An illustration of how choice sets capture the progress made by the sweep schedule with each choice set representing a connected contaminated area. The grey part is cleared and the white part inside the polygon is contaminated.

Our goal is now to show that all sweep schedules that make the same choices for $o_1, \dots, o_n$ (and therefore have the same choices sets) also have the same lower bound for their cost. We do this by expressing a lower bound for the sweep schedule in terms of $\mathcal{T}(t_1), \dots, \mathcal{T}(t_{n-1})$.

Let $d(\cdot, \cdot)$ be the shortest path in $\mathcal{E}_s$ between two points or edges[4]. Since $\mathcal{E}_s$ is bounded by a polygon, there also exists a sweep line between these two points or edges with length $d(\cdot, \cdot)$. For a choice set T_k^i we define (recall definition 8)

$$b(T_k^i) := c(d(e_{i-1}, e_{i+k})).$$

$b(T_k^i)$ is called the blocking cost for T_k^i, since it reflects the minimal costs for blocking recontamination from entering $\mathcal{R}(t)$ when $T_k^i \in \mathcal{T}(t)$.

Now, as the sweep schedule progresses it will expand $\mathcal{R}(t)$ until it clears a first edge from T_k^i. Figure 6 illustrates this process. Let e_o be this first edge with $o \in T_k^i$. Clearing e_o requires moving the sweep line that bounds $\mathcal{C}(T_k^i)$ onwards toward e_o at some additional cost. A lower bound for this cost is obtained by considering the lowest cost for two sweep

[4]This path can be computed efficiently combining ideas from [38, Chapter 6.2.4] with [39] (see Section VII.)

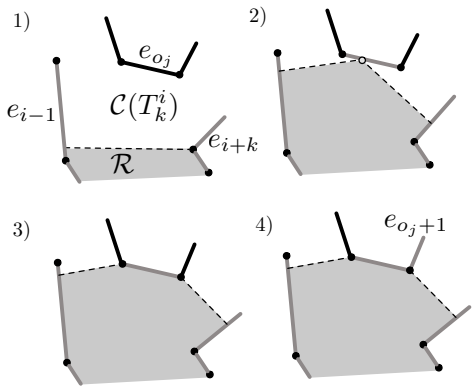

Fig. 6. An illustration of a sweep line from part 1) splitting in part 2) and moving to the lowest blocking cost in part 3). Part 4) shows another split that occurs at no additional cost for which the left side has a zero length sweep line, clearing e_{o_j+1}. Cleared edges are grey.

lines, one between e_{i-1} and e_o and one between e_{i+k} and e_o, which have the same endpoint on e_o. We call this lower bound the cost for choosing o out of T_k^i and define it as:

$$c(o|T_k^i) := \min_{p \in e_o} \{c(d(p, e_{i-1})) + c(d(p, e_{i+k}))\}.$$

It is now straightforward to show that at any point in time t in between $t_j < t < t_{j+1}$ we have

$$c(\tau(t)) \geq \sum_{T \in \mathcal{T}(t)} b(T)$$

and that at time t_j when clearing edge e_{o_j} with $o_j \in T_k^i \in \mathcal{T}(t_{j-1})$ we have

$$c(\tau(t_j)) \geq c(o_j|T_k^i) + \sum_{T_k^i \in \mathcal{T}(t_{j-1}) \setminus T_k^i} b(T).$$

In addition to providing a lower bound the above also gives us a construction for a minimal cost sweep schedule with fixed choices for the sequence $o_1, \ldots, o_n$. For this it suffices to observe that there exist blocking sweep lines with cost $b(T)$ for every choice set T, that there exist splitting sweep lines with cost $c(o|T_k^i)$, and that one can move between these (for this one requires simply-connectedness). In this sense the lower bounds are tight, i.e., achievable with a sweep schedule.

The above considerations reduce the problem of computing a sweep schedule to the problem of determining which sequence $o_1, \ldots, o_n$ leads to a solution for the Line-Clear problem, since for any sequence we can construct its minimal cost sweep schedule. Clearly, a brute force approach testing all $n!$ possible sequences is not practical and in the next section we will demonstrate how to use choice sets for a more efficient solution.

VI. Finding Optimal Sweep Schedules for Simply-Connected Environments

In this section we show how to compute sequences $o_1, \ldots, o_n$ that lead to optimal cost sweep schedules and hence solve the connected and monotone Line-Clear problem. We first develop a representation of the search space for these sequences, and then turn to efficiently determining their cost. The basic idea is to use the choice sets from Section V to build a sequence $o_1, o_2, \ldots, o_n$ recursively.

The initial choice set for an environment with n vertices is $\{1, \ldots, n\}$. From this set we choose some o_1 as the first obstacle index. The next choice, for o_2, will lead to a separation of the remaining choices $\{1, \ldots, n\} \setminus \{o_1, o_2\}$ into two sets, the *right side* and *left side*. On the left side are all indices o such that $o_1 < o < o_2$ whereas on the right side are all indices o such that $o_2 < o < o_1$.[5] These sides are represented by choice sets and their connected contaminated components. There will be one choice set in $\mathcal{T}(t_2)$ if o_1 and o_2 are consecutive and two if they are not. In one of these connected components we can make a further choice of obstacle index for o_3 and so on. Figure 7 illustrates this idea. On the left subfigure, a line is setup between obstacles e_3 and e_8. Therefore two contaminated regions are placed to the left and the right of the initial line. The edges on the boundary of the left area are given by the choice set T_4^4, whereas the edges of the right side are given by T_{12}^9. On the right subfigure, the initial lines are setup along edges e_3 and e_4 and there is therefore just one contaminated area whose edges are given by the choice set T_{16}^5.

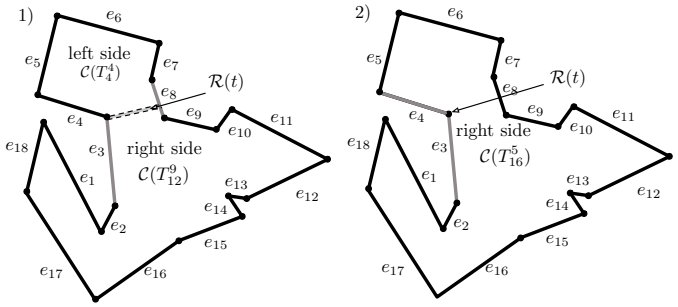

Fig. 7. An illustration of the first two choices, o_1 and o_2. On the left we have $o_1 = 3$ and $o_2 = 8$ resulting in a cleared line on the shortest path between e_3 and e_8 and $\mathcal{T}(t_2) = \{T_4^4, T_{12}^9\}$. On the right a consecutive choice of $o_1 = 3$ and $o_2 = 4$ leads to an empty choice set on the left side and $\mathcal{T}(t_2) = \{T_{16}^5\}$ on the right side.

The key to an efficient solution is to exploit the recursive structure of choice sets that emerges because of these splits into left and right sides. The next subsections characterize this structure and provide the associated algorithms to efficiently solve subproblems that arise.

A. Obstacle Sequences in Choice Sets

For a given choice set T_k^i and its associated contaminated area $\mathcal{C}(T_k^i)$ we are now going to construct a "local" sequence of obstacle indices only from T_k^i which corresponds to a "local" sweep schedule that clears $\mathcal{C}(T_k^i)$. Let $\mathcal{O}(T_k^i)$ be the set of all sequences of obstacle indices from T_k^i. Among these $k!$ sequences we would like to determine the *best* one, i.e., the one leading to the lowest cost sweep schedule. To this end, we define the cost of an obstacle index sequence based on the blocking and clearing costs of choice sets. Let $O \in \mathcal{O}(T_k^i)$

[5]Recall that we consider a circular ordering.

be a shorthand for one sequence, written $O = o_1, o_2, \ldots, o_k$. As mentioned earlier the choice for o_1 splits the contaminated region $\mathcal{C}(T_k^i)$ into (at most) two regions, called the right side and the left side. Both of these are associated with choice sets that are subsets of T_k^i. These must be cleared as well by defining their own sweep schedules with associated costs. This continues until all indices are chosen. The cost for the s-th choice, i.e., for o_s, is determined as follows. Write T^s for the choice set out of which we choose o_s. Moreover, let $T^{r,s}$ and $T^{l,s}$ be the left and right choice sets into which T^s will split when clearing e_{o_s}. The cost at step s of O, written $c_s(O)$, is then defined as follows.

Definition 12 (Sequence Costs)**.** *Let $O \in \mathcal{O}(T_k^i)$ be a sequence of obstacle indices. For $s = 0, \ldots, k$ we recursively define the sequence costs as follows:*

$$c_0(O) := 0 \quad \textit{and} \quad b_0(O) := b(T_k^i)$$
$$c_s(O) := b_{s-1}(O) - b(T^s) + c(o_s|T^s) \quad s \geq 1$$
$$b_s(O) := b_{s-1}(O) - b(T^s) + b(T^{l,s}) + b(T^{r,s}) \quad s \geq 1$$

The rationale behind the definition is as follows. The term $b_0(O)$ is the initial blocking cost to prevent recontamination from $\mathcal{C}(T_k^i)$. The term $b_s(O)$ describes the cost to prevent recontamination after having executed step s and cleared e_{o_s}. It is computed from the previous blocking cost at $s-1$ by removing $b(T^s)$, since T^s is split, and adding the costs for the resulting left $b(T^{l,s})$ and right side $b(T^{r,s})$. The total cost to execute s is then the cost to prevent recontamination from the previous step $s-1$ except for the $b(T^s)$ robots blocking T^s which can be reused for clearing e_{o_s} at the cost of $c(o_s|T^s)$ robots. The sequence is seeded at $c_0(O) = 0$ for convenience. The steps above can also be seen in the context of Figure 6 with subfigure 1 showing $b_0(O)$, subfigure 2 showing $c_1(O)$, and subfigure 3 showing the updated $b_1(O)$. Also note that subfigure 4 shows a particular case in which there is no noticeable movement or change in cost when clearing e_{o_j+1}, since the left side is empty and $b(T^s) = b(T^{r,s}) = c(o_j + 1|T^s)$, effectively clearing e_{o_j+1} at no additional cost.

We next introduce the notion of *critical steps* that will aid us in the construction of a low cost sequence O.

Definition 13 (Critical Step)**.** *For a given sequence O of length k and for all $s \leq k$, define $c_s^{max}(O) := \max_{j \leq s}\{c_j(O)\}$. A critical step of O is a step s that satisfies:*

$$c_{s'}^{max}(O) \leq c_s^{max}(O) \Rightarrow b_{s'}(O) \geq b_s(O) \tag{1}$$

for all steps $s' \in \{1, \ldots, k\}$.

In other words, a critical step is a step that has the lowest blocking cost b_s amongst all other steps that have the same or lower total cost to reach, given by c_s^{max}. Conversely, a critical step s has the lowest total cost c_s^{max} with which we can reach a blocking cost of b_s and is hence the cheapest way to reach this blocking cost. Note that to reach step s all previous steps need to be executed, hence the definition of c^{max}. Let us write $S(O) := \{s_1, \ldots, s_{k'}\}$ for all critical steps of O in order. Note that the first critical step is always $s_1 = 0$ with $b_0(O)$ and $c_0 = 0$ and the last critical step is always $s_{k'} = |T_k^i| = k$ with $b_k(O) = 0$ with the highest total cost to reach. For each critical step $s_j, j = 1, \ldots, k'$ we now define:

$$\rho_{s_j}(O) = c_{s_j}^{max}(O) - b_{s_{j-1}}(O).$$

For all non-critical steps $s \in (s_{j-1}, s_j) \subset \mathbb{N}$ we define $\rho_s(O) := \rho_{s_j}(O)$. The following lemma establishes some facts about sequences of critical steps that will be useful later on.

Lemma 1. *The following statements hold:*

1) *$b_{s_j}(O)$ is a decreasing sequence with $j = 1, \ldots, k'$;*
2) *$\rho_s(O)$ is a non-decreasing sequence with $s = 1, \ldots, k$;*
3) *$c_{s_j}^{max}(O) = \max_{s \in (s_{j-1}, s_j]} c_s(O)$.*

Proof: See appendix.

We now turn to the problem of choosing the best index out of T_k^i. Write $o \in T_k^i$ for the chosen obstacle index for splitting T_k^i into T^l and T^r. Suppose that we are already given fixed sequences to clear $\mathcal{C}(T^l)$, written as $O^l \in \mathcal{O}(T^l)$, and $\mathcal{C}(T^r)$, written as $O^r \in \mathcal{O}(T^r)$. Algorithm 1 shows how to construct a sequence $o_1, \ldots, o_k = O^a \in \mathcal{O}(T_k^i)$ given fixed choices for $o_1 = o$, O^l, and O^r. Therein the obstacle indices on the left and right sides are simply ordered with their respective ρ_s increasing and then added to O^a in that order (line 4-8).

Algorithm 1 $Min\_Obstacle\_Sequence(i, k, o, O^l, O^r)$

```
1:  o_1 ← o
2:  s^l ← 1, s^r ← 1, s ← 2.
3:  while s ≤ k do
4:      if ρ_{s^l}(O^l) < ρ_{s^r}(O^r) then
5:          o_s ← O^r_{s^l}, s^l ← s^l + 1
6:      else
7:          o_s ← O^r_{s^r}, s^r ← s^r + 1
8:      end if
9:      s ← s + 1
10: end while
11: return {o_1, ..., o_k}
```

The following theorem proves that O^a has not only the lowest possible cost of all sequences starting at o and having O^l and O^r as subsequences, but also that no other sequence can have an intermediate step that is better (i.e., a step with a lower block cost that is reached at lower total cost). In other words, O^a is the best possible combination of O^l and O^r.

Theorem 2. *Let $O \in \mathcal{O}(T_k^i)$ with $o_1 = o$ and let $O^l \in \mathcal{O}(T^l)$ and $O^r \in \mathcal{O}(T^r)$ be the obstacle sequences from the left and right choice set that are subsequences of O. Now, let $O^a \in \mathcal{O}(T_k^i)$ be the obstacle sequence constructed by Alg. 1 with input k, i, o, O^l, O^r. Then for all critical steps $s \in S(O), s' \in S(O^a)$:*

$$c_s^{max}(O) < c_{s'}^{max}(O^a) \Longrightarrow b_s(O) > b_{s'}(O^a).$$

Proof: See Appendix.

The next lemma is concerned with sequences $O \in \mathcal{O}(T_k^i)$ that will never be useful subsequences for a larger sequence. We call these *inferior sequences*, defined as follows:

Definition 14 (Inferior Obstacle Sequence)**.** *A sequence $O \in \mathcal{O}(T_k^i)$ is called* inferior *if $\exists \tilde{O} \in \mathcal{O}(T_k^i)$ s.t.:*

$$b_s(O) \leq b_{s'}(\tilde{O}) \implies \rho_s(O) \geq \rho_{s'}(\tilde{O}) \\ \wedge\ c_s^{max}(O) \geq c_{s'}^{max}(\tilde{O})$$

It follows directly from the definition and substitutions in the equations in the proof of Theorem 2 that inferior sequences, when considered as a left or right subsequence (O^l or O^r), cannot yield a non-inferior larger sequence O^a. As a consequence, in a recursive construction, we do not have to include inferior sequences when computing optimal sequences. Algorithm 2 shows how to test whether an obstacle sequence is inferior to another. It simply scans through the sequences and tests if O is inferior by applying Definition 14, i.e., it returns false if it finds a step that has a better or equal blocking cost reachable at lower total cost (returning false on line 6) or a lower blocking cost reached at the same total cost (returning false on line 13). If it cannot find such a step, then O is inferior.

Algorithm 2 $is\_inferior(O, \tilde{O})$

```
1:  b_min ← b_1(O), b̃_min ← b_1(Õ)
2:  c_max ← c_1(O), c̃_max ← c_1(Õ)
3:  while j ≤ k AND s ≤ k do
4:    if c_max < c̃_max then
5:      if b_min ≤ b̃_min then
6:        return false
7:      end if
8:      c_max ← max{c_max, c_s(Õ)}
9:      b_min ← min{b_min, b_s(O)}
10:     s ← s + 1
11:   else
12:     if c_max = c̃_max AND b_min < b̃_min then
13:       return false
14:     end if
15:     c̃_max ← max{c̃_max, c_j(Õ)}
16:     b̃_min ← min{b̃_min, b_j(Õ)}
17:     j ← j + 1
18:   end if
19: end while
20: return true
```

Algorithm 3 shows how to exploit this for finding all obstacle sequences for a choice set T_k^i and a choice $o \in T_k^i$ adding the resulting obstacle sequences to a set $\tilde{\mathcal{O}}(T_k^i) \subset \mathcal{O}(T_k^i)$, unless it is inferior. This requires that previously all relevant sequences on the left and right sides ($\tilde{\mathcal{O}}(T^l)$ and $\tilde{\mathcal{O}}(T^r)$) have already been constructed. Calling Algorithm 3 for every choice $o \in T_k^i$ would build up the complete $\tilde{\mathcal{O}}(T_k^i)$, an auxiliary structure that collects all relevant obstacle sequences for T_k^i. The set $\tilde{\mathcal{O}}(T_k^i)$ contains at most one sequence for every combination of sequences from the left and right sides due to Theorem 2 (instead of all possible $\binom{k-1}{|O^r|}$ combinations) and only non-inferior sequences, which makes it significantly smaller than the set of all sequences $\mathcal{O}(T_k^i)$.

Algorithm 3 can be applied to all choice sets with the outer loop going from $k = 1$ to $k = n$, an inner loop going from $i = 1$ to $i = n$, and an innermost loop with all $o \in T_k^i$ as shown in Algorithm 4. This ensures that all necessary sequences on every possible left and right side are built before they are required. Finally, the best obstacle sequence is selected among all choice sets with $k = n$. Theorem 2 and the fact that inferior sequences cannot construct optimal sequences guarantee that Algorithm 4 will find the optimal obstacle sequence. The corresponding sweep schedule is hence an optimal sweep schedule and a solution for the connected Line-Clear problem in simply-connected environments.

Algorithm 3 $Combine\_Obstacle\_Sequences(i, k, o)$

```
1:  T^l ← T^{i+1}_{o-i}, T^r ← T^{o+1}_{i+k-o-1}
2:  o_1 ← o
3:  for all (O^l, O^r) ∈ Õ(T^l) × Õ(T^r) do
4:    O ← Min_Obstacle_Sequence(i, k, o, O^l, O^r)
5:    inferior ← false
6:    for all Õ ∈ Õ(T^i_k) do
7:      if is_inferior(O, Õ) then
8:        inferior → true, break
9:      else if is_inferior(Õ, O) then
10:       Õ(T^i_k) ← Õ(T^i_k) \ {Õ}
11:     end if
12:   end for
13:   if ¬inferior then
14:     Õ(T^i_k) ← Õ(T^i_k) ∪ {O}
15:   end if
16: end for
17: return
```

Algorithm 4 $Connected\_Line\_Clear\_Number(\mathcal{E}_s)$

```
1:  Define b(·) and c(·|·) using E_s
2:  for all k = 1, ..., n do
3:    for all i = 1, ..., n do
4:      for all o ∈ T^i_k do
5:        Combine_Obstacle_Sequences(i, k, o)
6:      end for
7:    end for
8:  end for
9:  min ← ∞
10: for all i = 1, ..., n do
11:   O_min ← argmin_{O∈Õ(T^i_n)} {c^max_n(O)}
12:   if c^max_n(O_min) < min then
13:     min ← c^max_n(O_min)
14:   end if
15: end for
16: return min
```

B. Complexity

The complexity of Algorithm 4 is difficult to determine. The non-trivial parts are the additions to the sets $\tilde{\mathcal{O}}$ and the cartesian product of all sequences on the left and right (Algorithm 3 line 3). The result from Theorem 2 allows to ignore all combinations of left and right subsequence but one. Yet this can still lead to exponential growth of $\tilde{\mathcal{O}}(T_k^i)$ with increasing k due to the cartesian product in line 3 of Algorithm 3.

The key questions remaining is how much further the consideration of only non-inferior sequences reduces the complexity or, in other words, how many sequences in $\mathcal{O}(T_k^i)$ can be non-inferior and hence become part of $\tilde{\mathcal{O}}(T_k^i)$. From Definition 14 it follows that if for all $\tilde{O}$ there is some s, s' so that:

$$b_s(O) \leq b_{s'}(\tilde{O}) \wedge (\rho_s(O) \geq \rho_{s'}(\tilde{O}) \vee c_s^{max}(O) \geq c_{s'}^{max}(\tilde{O}))$$

then O is non-inferior. Hence O has to have at least one step that has a low blocking cost that cannot be achieved by another obstacle sequence either earlier in the ρ-ordering or with lower cost c^{max}. If all the term above are equal we could consider one of the two sequence as a duplicate and remove it, but this does not make a significant difference. The problem is that the criterion above may still allow us to generate an exponential number of sequences by setting appropriate values for ρ, c^{max}, b in which each sequence has some advantage over another. Given these values the next open question is whether one can construct an environment $\mathcal{E}_s$. If it is, then our algorithm has exponential complexity in n and we would conjecture that connected Line-Clear in simply-connected environments is NP-hard. If not, then our algorithm has polynomial time complexity. Discussing how one may generate an exponential number of appropriate values for ρ, c^{max}, b that is required to generate an exponential number of non-inferior sequences and the construction of a corresponding environment is beyond the scope of this paper and an interesting question for further work. In practice, however, one is unlikely to encounter such an environment. A single narrow section with a low blocking cost b will already wipe out many alternative obstacle sequences. In practice, we hence expect polynomial runtime and we provide some support for this conjecture in the next section.

In the next section we also discuss further practical improvements, an implementation for finding shortest block and split lines, and experiments on random environments that shed some light on the number of non-inferior sequences and their growth in larger environments. In addition we compare two simpler heuristic variants to our algorithm with respect to their computation time and resulting cost of sweep schedules.

VII. Implementation and Experiments

A. Implementation Details

To solve the connected Line-Clear problem in simply-connected environments we need to compute $c(o|T_k^i)$ and $b(T_k^i)$. These two costs are the connection between the geometry of the environment and the combinatorial perspective of the problem established in Section V. Accordingly, we combine ideas from shortest-path roadmaps with edge to edge visibility considered. For our purposes two edges are visible if there exists a straight line l between two points on either edge that is a subset of $l \subset \mathcal{E}$. The key observation exploited in [39] is that two edges are visible either when one of the edges is visible from an endpoint of the other or when another vertex, which is not part of either edge, sees both edges (illustrated in Fig. 8). It hence suffices to look at the visibility polygons at every vertex since two edges will have a line going through some vertex.

To compute the shortest path between any two obstacle edges in our polygon environment classic algorithms like [40] can be used on the full-visibility graph augmented with edges towards all reflex vertices (see also [38, Chapter 6.2.4]). Within this graph there are two types of vertices, those corresponding to a vertex of an obstacle edge and those corresponding to the obstacle edge. To search this structure one can use A* with edge weights corresponding to the distance between reflexive vertices and reflexive vertices to obstacle edges [41]. A valid path starts at an obstacle edge, possibly continues across multiple reflexive vertices, and ends at an obstacle edge. This determines all $b(T_k^i)$ in $O(nE)$, with E being the number of edges in the roadmap, by computing the shortest edge-edge path for each edge to all others (note that there are n^2 different T_k^i).

To compute $c(o|T_k^i)$ we leverage the computation above by looking up the shortest path between edges e_{i-1} and e_o as well as e_{i+k} and e_o. Let $[a_1, b_1]$ and $[a_2, b_2]$ be the last line segment of these paths which have endpoints b_1 and b_2 on e_o. The point p on e_o that realizes $c(o|T_k^i)$ has the same angle for $[p, x_r]$ and $[p, x_l]$, as shown in Fig. 9. Now, the previous x_r and x_l may not anymore be the endpoints of the shortest path from p to the respective obstacle edges, if the line segment 'peeled' off the reflexive vertex, as shown in Fig. 9 for x_r. The number of times this can happen is bounded by the number of reflexive vertices visible from the obstacle edge e_o.

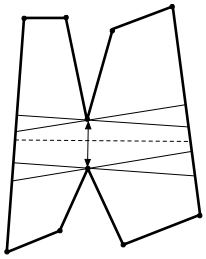

Fig. 8. Thick lines show obstacle edges. Any visibility line between two edges that is not at an endpoint, shown as a dashed line, can be moved in either direction until it touches a vertex of another edge. Hence visibility between any two edges implies a shortest visibility line at a vertex.

B. Polynomial Growth Experiments

The primary concern for the complexity of Algorithm 4 is the growth of $|\tilde{O}(T_k^i)|$, in particular that it should not grow exponentially. To experimentally estimate its growth trend we implemented the algorithms and tested them with random polygons of varying sizes ranging from $n = 5$ to $n = 400$ vertices. Results are shown in Fig. 10 and hint that the number of obstacle sequences that have to be considered grows at a similar rate than the number of choice sets (which is exactly n^2).

Overall this suggests that obtaining $O(n^3)$ complexity, under reasonable conditions, is within reach, since Alg. 3 can potentially be improved to require $O(n)$ steps. Note that the related Graph-Clear problem is $O(n^2)$ on trees (which correspond to simply-connected environments) and in a sense

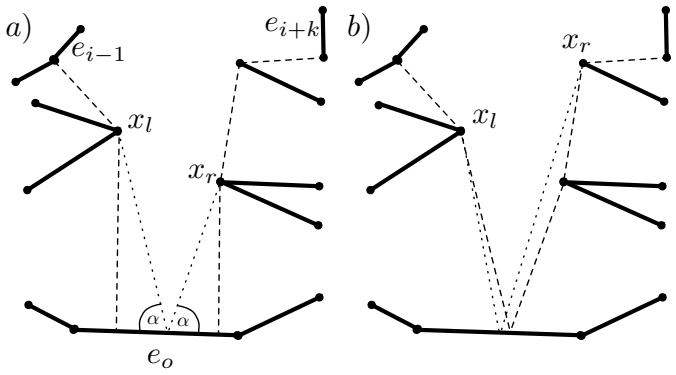

Fig. 9. This figure illustrates how to compute the best split point on an obstacle edge e_o for $c(o|T_k^i)$. In part a) the shortest lines between e_o and the two other obstacle edges are shown as dashed lines. The best split point for the final segment of these two lines is where the angle to the endpoints, x_r and x_l, is identical, shown as α. Now, the new shortest line from this point may not go through these endpoints in which case the procedure has to be repeated for these endpoints.

the Line-Clear problem has to deal with all possible trees that can be embedded into $\mathcal{E}_s$ (with the choices at each choice set defining the structure of the tree).

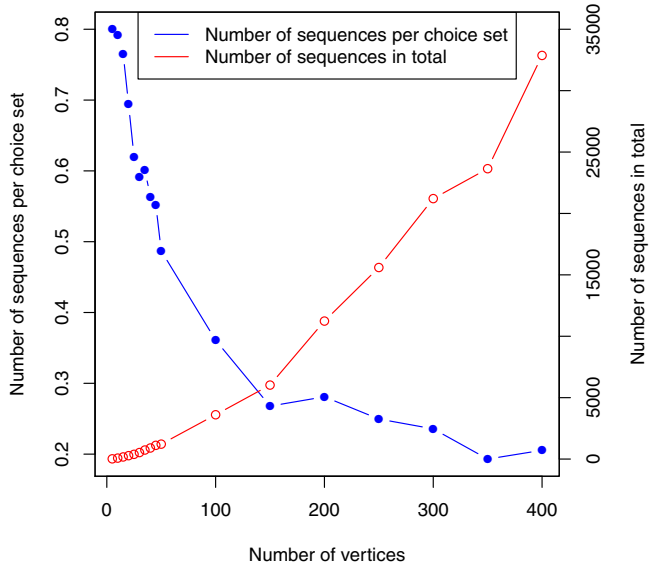

Fig. 10. Total number of sequences computed across all choice sets. For each data point 10 random polygons were constructed and the plot shows the resulting mean.

C. Further Improvements and Heuristic Comparisons

Additional practical enhancements to Algorithm 4 can avoid computing the sequences for all choice sets. One simple improvement to line 4 is to compute the sequences only if the shortest line that realizes the cost $b(T_k^i)$ does not pass through a reflexive vertex of a contaminated obstacles $o_c \in T_k^i$. If it does, then a split on o_c could have executed earlier thereby avoiding ever reaching a state in which T_k^i appears. On tests with polygons with 10 to 400 vertices, we observed that adding this we can avoid between 11% (for environments with 10 vertices) and 62% (for environments with 400 vertices) of all choice sets.

To derive some useful "yardsticks" to compare the performance of our algorithms with heuristic approaches solving the same problem, we derive simpler variants of the algorithm. For example, instead of considering all possible $o \in T_k^i$ in lines 4 to 6 in Algorithm 4, we can introduce a heuristic that chooses either the best next split $o_{split} = \arg\min_{o \in T_k^i}\{c(o|T_k^i\}$ or the best next choices for minimizing the blocking cost $o_{block} = \arg\min_{o \in T_k^i}\{b(T^l) + b(T^r)\}$. This approach reduces the number of sequences in $\tilde{\mathcal{O}}(T_k^i)$ to one for each choice set, and then we only have to assemble a single sequence in Algorithm 3. This leads to a single call to Algorithm 1 taking $\mathcal{O}(k)$ time. Hence the overall complexity for these simpler variants is $\mathcal{O}(n^4)$. We shall call *minimum split* the heuristic choosing the best split, and *minimum block* the one that selects the choice for minimizing the best block.

To assess the performance of our optimal algorithm, we contrast its runtime and solution quality against the heuristics we just described and display the results in Fig. 11 and Fig. 12.

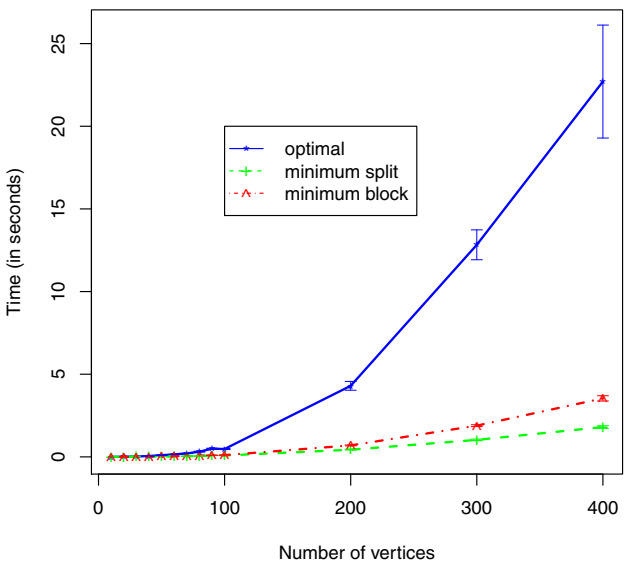

Fig. 11. A comparison of actual runtimes on a regular consumer laptop (2013 Macbook Pro) between the optimal algorithm and two simplified variants choosing either a *minimum split* or *minimum block* in each choice set. For each data point 10 random polygons were constructed and the plot shows the resulting mean and standard error.

Fig. 11 shows that, as expected, the two heuristics are faster than the optimal algorithm because they consider a significantly smaller search space. However, Fig. 12 confirms that there is value in using the optimal algorithm, as the strategies it produces are clearly less costly. Combined, the three algorithms we presented offer a range of different approaches to solving instances of the Line-Clear problem.

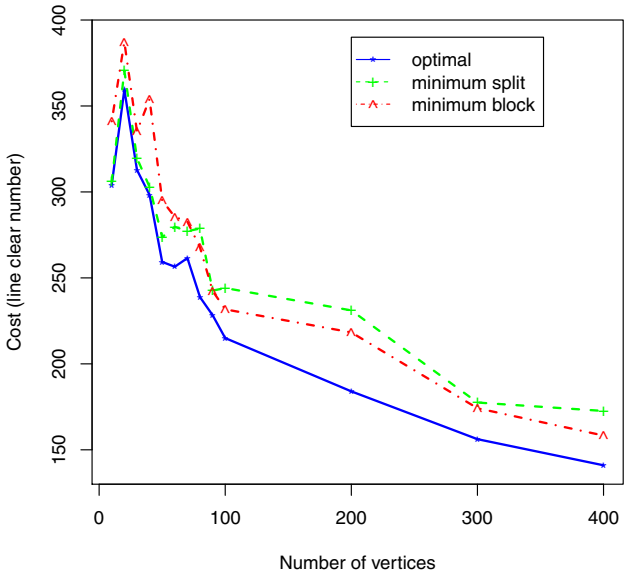

Fig. 12. A comparison of cost of strategies resulting from the optimal algorithm and two simplified variants choosing either a minimum split or minimum block in each choice set. For each data point 10 random polygons were constructed and the plot shows the resulting mean.

VIII. Discussion and Conclusion

We presented and formalized the problem of searching for worst-case intruders in a bounded two dimensional environment, coined Line-Clear. Through a reduction from a graph-based pursuit-evasion problem known to be NP-hard, we showed that Line-Clear is also NP-hard.

We then considered the connected Line-Clear problem in simply-connected environments and provided a combinatorial perspective for this restricted setting that is related to the original problem via a minimal sweep schedule construction. Exploiting this combinatorial perspective we showed how to compute optimal sweep schedules based on the sequence in which they clear all obstacles in the environment. Questions regarding its complexity remain open and we identified conditions under which the algorithm runs in polynomial time. Practical considerations regarding the covering of sweep lines with sensors were addressed in prior work, in addition to applying the algorithm for simply-connected environments to multiply-connected environments [4], [42].

Overall, the problem of searching for targets in 2D, as one would expect, is considerably more challenging than searching on a graph. While graph-searching algorithms have been applied to robotics problems [17], [20], [22], [43] the problem of constructing suitable graphs has remained challenging. The work presented here fills this gap and in a way the structures here can be thought of as finding the optimal combinatorial representation of an environment. In addition, our methods also exploit techniques from graph-searching, particularly with the trade-off between the cost to block recontamination and to further expand a cleared area.

From a theoretical perspective there are a number of open problems that can now be addressed using our formal problem definition. Especially the question of whether recontamination matters is of interest. Visibility-based pursuit-evasion with unlimited range sensors is not monotone but many graph-based models are. We conjecture that connected Line-Clear is in fact monotone and a combination of the proof strategies developed in [44] with a geometric analysis may lead to a proof of this conjecture. In addition, the question regarding the number of inferior obstacle sequences is left open, particularly whether 2D environments can be constructed in which the number of non-inferior sequences grows exponentially.

From a practical perspective there are more questions that need to be addressed. One of the first is the consideration of time-optimal strategies, with some preliminary work presented in [5]. Second one should consider a distributed variant, similar to [36] and [3], that utilizes the theoretical insights developed here and investigates the trade-off between the time it takes to clear a known and an unknown environment. Thirdly, one can consider probabilistic aspects of the search problem and integrate these into Line-Clear. In particular, the relaxation of the target model from worst-case to probabilistic models, as already successfully done for graphs in [45], bears some promise. In the context of visibility-based pursuit-evasion this has been attempted in [46]. Further probabilistic considerations can be made with regard to maximizing the expected time to capture, similar to what was done in [43] for graphs. Probabilistic sensors could also be considered building on the work in [21] and [47] which consider probabilistic detection for graph-based and visibility-based models respectively.

The basic ideas of this paper may also be extended to 3D environments, although the computation of *smallest sweep planes* (the 3D analogue of shortest sweep lines) and how to cover them with sensors is more difficult. But the order in which to move such planes may have a similar structure.

Appendix

Proof of Theorem 1: We first show how to construct a sweep schedule for $\mathcal{E}^3$ from an edge-searching strategy for G^3. Let $S(G^3)$ be an optimal edge-searching strategy for G^3, i.e., an edge-searching strategy for G^3 using $s(G^3)$ searchers. We now show how to construct a schedule τ to clear $\mathcal{E}^3$ with cost $s(G^3)$. As shown in [14], recontamination does not improve edge-searching strategies, and hence, without loss of generality, we can assume that $S(G^3)$ does not recontaminate edges. To prevent recontamination, if a vertex v has at least one edge contaminated and one cleared there must be a searcher located in v.

A sweep schedule τ for $\mathcal{E}^3$ can be constructed by moving sweep lines through the corridors of $\mathcal{E}^3$ based on how $S(G^3)$ clears G^3. Let us consider a vertex v with degree 3 and show how edges incident to v are cleared. During the execution of $S(G^3)$ the edges of v are either cleared by a searcher moving *in* or *out* of v. An edge (v_1, v) is cleared moving *in* if the searcher moves from v_1 to v, whereas it is cleared moving *out* if the searcher moves from v to v_1. Let e_1, e_2, and e_3 be the first, second, and third edge of v that are cleared in

$S(G^3)$. Note that since we ruled out recontamination, each edge is cleared only once and the order is then well defined.

If $e_1 = (v, v_1)$ is cleared by an *out* step, then there must be at least two searchers on v prior to this step. Two searchers are needed because one will move along e_1 whereas the other will stay in v to prevent recontamination because edges e_2 and e_3 are still contaminated. The associated sweep schedule τ in $\mathcal{E}^3$ will be as follows. Two sweep lines l_1, l_2 are added to τ to cover $A(e_1) \cap A(v)$, i.e., $\bar{P}(l_1) = \bar{P}(l_2) = A(e_1) \cap A(v)$. Sweep line l_1 is then moved through $A(e_1)$ to clear it, stopping at $A(e_1) \cap A(v_1)$. If e_1 is cleared by an *in* step, then a sweep line from its neighbor vertex is moving through $A(e_1)$ and stops at $A(e_1) \cap A(v)$. Fig. 13 illustrates these two cases. Note that because of the assumptions made about d, one searcher suffices to clear the corridor, and one searcher can prevent recontamination, i.e., $c(l_1) = c(l_2) = 1$.

Similarly, if e_2 is cleared by an *out* step, there must be two searchers in v to prevent recontamination from e_3. In τ we have one sweep line l with $\bar{P}(l) = A(e_1) \cap A(v)$. We move l at cost at most 2 to split on $A(v) \cap (A(e_2) \cup A(e_3))$ into two sweep lines l_1, l_2, each at cost 1, with $\bar{P}(l_1) = A(e_2) \cap A(v)$ and $\bar{P}(l_2) = A(e_3) \cap A(v)$. Sweep line l_1 is then moved through $A(e_2)$ whereas l_2 prevents recontamination. If e_2 is cleared by an *in* step, then the new sweep line merges with the existing sweep line in τ and moves towards $A(v) \cap A(e_3)$.

For e_3, *in* and *out*, either the sweep line at $A(v) \cap A(e_3)$ is moved outward or a new sweep line is coming in through e_3. Fig. 13 illustrates the above. The cases for vertices with degree one and two are analogue and omitted for brevity. It is now immediate to verify that if two edges of v are cleared, then so is $A(v)$ and if an edge e is clear then so is $A(e)$. Similarly, the number of agents is the same as the cost for the sweep lines. Hence τ clears $\mathcal{E}^3$ at cost $s(G^3)$.

Next we show how to construct an edge-searching strategy from a given sweep schedule. Let τ_{opt} be an optimal sweep schedule for $\mathcal{E}^3$. We can construct an edge-searching strategy $S_{\tau_{opt}}(G^3)$ by reversing the idea described above. Note that because $s(G^3)$ is strictly speaking not known yet (since we only have an optimal sweep schedule), the construction of $\mathcal{E}^3$ for this direction uses $R(l) > lc(\mathcal{E}^3)$ instead of $R(l) > s(G^3)$. For this, we order all corridors $A(e)$ by the time at which they are cleared last.[6] Now, construct an edge-searching strategy $S_{\tau_{opt}}(G^3)$ which clears all edges e in the same order as $A(e)$, adding and removing searchers to guard vertices wherever necessary.[7]

It is easy to verify that the cost of $S_{\tau_{opt}}(G^3)$ is not more than than $lc(\mathcal{E}^3)$ by using the fact that $R(l) > s(G^3)$ (or $R(l) > lc(\mathcal{E}^3)$). This forces every sweep line to be contained within their corridors as they move between the areas associated to vertices. Together with the above construction of a sweep schedule from a strategy this proves the claim. □

Proof of Lemma 1 By definition $c^{max}_{s_j}(O)$ is non-decreasing. Hence, two critical steps with $s_{j'}$ and s_j with $j' < j$ satisfy

[6]Note that we need the *last* since it is not proven that Line-Clear is monotone.

[7]An optimal edge-searching strategy can also be written as a sequence of only edge moves, since the placement and removal of searchers on vertices is implicitly given by the constraints from avoiding recontamination.

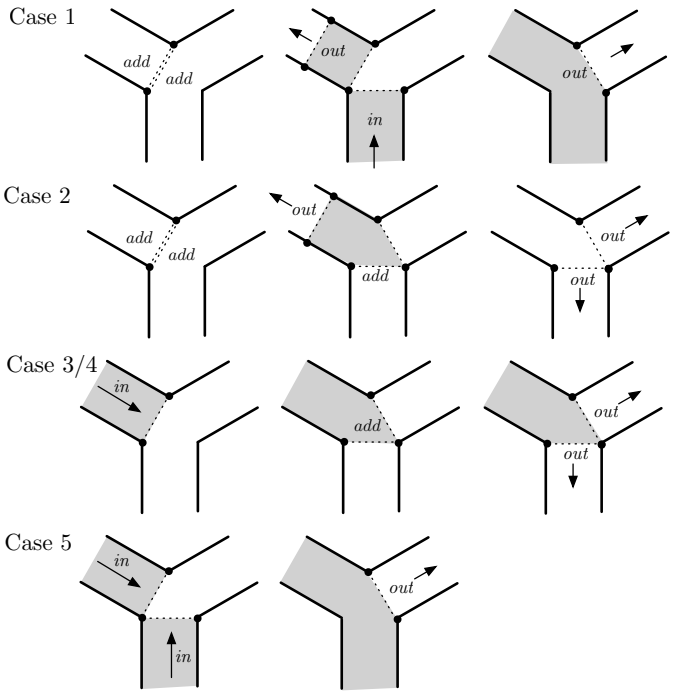

Fig. 13. An illustration of the relevant cases for constructing a sweep schedule from an edge-searching strategy on a graph G^3. Cleared areas are grey and sweep lines are dashed lines.

$c^{max}_{s_{j'}}(O) \leq c^{max}_{s_j}(O)$ and by definition of critical steps it follows that $b_{s_{j'}}(O) \geq b_{s_j}(O)$. Assuming the removal of duplicate critical steps we have $c^{max}_{s_{j'}}(O) < c^{max}_{s_j}(O)$ and therefore $b_{s_{j'}}(O) > b_{s_j}(O)$ (otherwise we get a contradiction to $s_{j'}$ being a critical step). Since $c^{max}_s(O)$ is non-decreasing and $b_{s_j}(O)$ is decreasing it follows that $\rho_s(O)$ is non-decreasing. Statement 3 also follows directly from $c^{max}_{s_{j'}}(O) < c^{max}_{s_j}(O)$, i.e., the step s that assumes the maximum must occur between s_{j-1} and s_j or at s_j. □

Proof of Theorem 2: First we will express the costs for O in terms of O^l and O^r. We write o_s, o^l_s and o^r_s for the elements in the sequences O, O^l and O^r, respectively, and use the superscripts l and r when referring to the left and right subsequences. Given any O, by definition we have $c_1(O) = c(o_1|T^i_k)$ and $b_1 = b(T^l) + b(T^r)$. For the remaining $o_s, s \geq 2$, we have a corresponding obstacle index either in O^l or O^r. To identify this index we write, for every $s = 2, \ldots, k$,

$$l(s) := \max\{0, \max\{s' \mid \exists s'' \leq s : o^l_{s'} = o_{s''}\}\},$$

for the last index in O^l whose obstacle corresponds to one of $\{o_1, \ldots, o_s\}$ (or 0 if there is no such index). For $r(s)$ the definition is analogous. For $s > 1$ we can now write:

$$c_s(O) = b_{l(s-1)}(O^l) + b_{r(s-1)}(O^r) + c(o_s|T^s) - b(T^s).$$

Written in this manner is becomes clear that $c(o_s|T^s) - b(T^s)$ is independent of the ordering of indices from O^l and O^r in O, i.e., it is simply the additional cost required to move a sweep line that is blocking T^s with cost $b(T^s)$ to split on o_s. The terms $b_{l(s-1)}(O^l)$ and $b_{r(s-1)}(O^r)$ are the parts with which the left side contributes to the cost when clearing an obstacle on the right side and vice versa. Here is where

the term ρ becomes relevant since it determines the trade-off between continuing on the left or right. The statement of the lemma can also be interpreted in colloquial terms as saying that no other obstacle sequence O can have a better critical step than O^a.

First let us show that we only need to consider sequences that change between O^l and O^r after a critical step in O^l and O^r. Consider s_j and s_{j+1}, two consecutive critical steps in O^l. All steps in between $s \in (s_j, s_{j+1})$ either satisfy:

1) $c_s^{max}(O^l) > c_{s_j}^{max}(O^l)$ and $b_s(O^l) \geq b_{s_j}(O^l)$ or
2) $c_s^{max}(O^l) = c_{s_{j+1}}^{max}(O^l)$ and $b_s(O^l) \geq b_{s_{j+1}}(O^l)$.

This means that they either have a higher total cost but no benefit in blocking, or they already have the same total cost as the next critical step which reduces the blocking cost by at least as much. Now, if we add a step o_s to O from the left side O^l and it is not critical and the next step o_{s+1} is on the right side O^r, then in case 1) the removal of all previous steps until $o^l_{s_j}$ (and in case 2) the addition of all steps until $o^l_{s_{j+1}}$) would not lead to a worse sequence.

Therefore, we can simplify the cost representation further by only considering the steps c_j which $o_{c_j} = o^l_{s_j}$ or $o_{a_j} = o^r_{s_j}$ for some critical step s_j in either O^l or O^r. Since $c^{max}_{l(c_j)}(O^l)$ and $c^{max}_{r(c_j)}(O^r)$ are non-decreasing sequences in j we get:

$$\max_{c_{j-1}<s\leq c_j} c_s(O) = \begin{cases} b_{r(c_{j-1})}(O^r) + c^{max}_{l(c_j)}(O^l), & \text{if } o_{c_j} \in O^l \\ b_{l(c_{j-1})}(O^l) + c^{max}_{r(c_j)}(O^r), & \text{otherwise.} \end{cases}$$

Also note that $b_{l(c_j)}(O^l)$ and $b_{r(c_j)}(O^r)$ are both non-increasing sequences in j. Now let O be any obstacle sequence with O^l and O^r as subsequences. Write c_j for the steps at which the critical steps on the right and left are added to O. Similarly, write c^a_j for the same for O^a. Let a_j be the first step so that $o_{c_j} \neq o^a_{c^a_j}$, i.e., the first difference in critical steps between O and O^a. Without loss of generality, let $o_{c_j} \in O^l$, and hence $o^a_{c^a_j} \in O^r$. Note that $c^a_{j-1} = c_{j-1}$ and $o^a_{c^a_{j-1}} = o_{c_{j-1}}$ and consider the following:

$$\begin{aligned}
\rho_{l(c_j)}(O^l) &\geq \rho_{r(c^a_j)}(O^r) \\
c^{max}_{l(c_j)}(O^l) - b_{l(c_{j-1})}(O^l) &\geq c^{max}_{r(c^a_j)}(O^r) - b_{r(c^a_{j-1})}(O^r) \\
c^{max}_{l(c_j)}(O^l) + b_{r(c^a_{j-1})}(O^r) &\geq c^{max}_{r(c^a_j)}(O^r) + b_{l(c_{j-1})}(O^l) \\
c^{max}_{l(c_j)}(O^l) + b_{r(c_{j-1})}(O^r) &\geq c^{max}_{r(c^a_j)}(O^r) + b_{l(c^a_{j-1})}(O^l) \\
\max_{c_{j-1}<s\leq c_j} c_s(O) &\geq \max_{c^a_{j-1}<s\leq c^a_j} c_s(O^a).
\end{aligned}$$

Let b be the index at which $o_b = o^a_{c^a_{j-1}+1}$ and $o_y = o^a_{c^a_j}$. Consider the sequence

$$\begin{aligned}
O' = \{&o_1, \ldots, o_{c_{j-1}}, o^a_{c^a_{j-1}+1}, \ldots, o^a_{c^a_j}, \\
&o_{c_{j-1}+1}, \ldots, o_{b-1}, o_{y+1}, \ldots, o_k\}
\end{aligned}$$

based on O but with the indices $o^a_{c^a_{j-1}+1}$ to $o^a_{c^a_j}$ moved to the position they have in O^a. Now since $\max_{c_{j-1}<s\leq c_j} c_s(O) \geq \max_{c^a_{j-1}<s\leq c^a_j} c_s(O^a)$ and $b_{l(c^a_j)}(O^l) < b_{l(c_{j-1})}(O^l)$ the only difference between O and O' is that the blocking cost on the left is reduced earlier without incuring a larger c^{max}. Hence all subsequent costs are either the same or lower, i.e., $\forall s, s'$ with $o_s = o'_{s'}$ we have:

$$b_{s'}(O') \leq b_s(O) \text{ and } c^{max}_{s'}(O') \leq c^{max}_s(O).$$

Repeating the above procedure for O', which is now identical up to obstacle $o^a_{c^a_j}$ with O^a we can move all steps into the order they are in O^a without incurring larger c^{max} or larger b costs.

At this point, the statement follows from a simple contradiction. Assume that there is an O with an $s \in S(O)$ and $s' \in S(O^a)$ so that

$$c^{max}_s(O) < c^{max}_{s'}(O^a) \text{ and } b_s(O) \leq b_{s'}(O^a).$$

Let $o^a_{s''} = o_s$ and by the above construction

$$c^{max}_s(O) \geq c^{max}_{s''}(O^a) \text{ and } b_s(O) \geq b_{s''}(O^a)$$

and hence

$$c^{max}_{s'}(O^a) > c^{max}_{s''}(O^a) \text{ and } b_{s'}(O^a) \geq b_{s''}(O^a)$$

which contradicts $s \in S(O^a)$. □

REFERENCES

[1] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf, *Computational geometry: algorithms and applications*. Springer, 2000.

[2] A. Kolling and S. Carpin, "Surveillance strategies for target detection with sweep lines," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2009, pp. 5821–5827.

[3] ——, "Multi-robot pursuit-evasion without maps," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2010, pp. 3045–3051.

[4] A. Kolling and A. Kleiner, "Multi-uav motion planning for guaranteed search," in *Proceedings of the Twelth International Joint Conference on Autonomous Agents and Multiagent Systems*, 2013, pp. 79–86.

[5] A. Kolling, A. Kleiner, and P. Rudol, "Fast guaranteed search with unmanned aerial vehicles," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013.

[6] T. Chung, G. Hollinger, and V. Isler, "Search and pursuit-evasion in mobile robotics," *Autonomous Robots*, vol. 31, no. 4, pp. 299–316, 2011.

[7] A. Khan, B. Rinner, and A. Cavallaro, "Cooperative robots to observe moving targets: Review," *IEEE Transactions on Cybernetics*, to appear.

[8] Y. Liu and G. Nejat, "Robotic urban search and rescue: A survey from the control perspective," *Journal of Intelligent & Robotic Systems*, vol. 72, no. 2, pp. 147–165, 2013.

[9] N. Noori, A. Beveridge, and V. Isler, "Pursuit-evasion: A toolkit to make applications more accessible [tutorial]," *IEEE Robotics Automation Magazine*, vol. 23, no. 4, pp. 138–149, 2016.

[10] T. Parsons, "Pursuit-evasion in a graph," in *Theory and Applications of Graphs*, Y. Alavi and D. R. Lick, Eds. Springer Berlin / Heidelberg, 1978, vol. 642, pp. 426–441.

[11] B. Alspach, "Searching and sweeping graphs: a brief survey," *Le matematiche*, vol. 59, no. 1, 2, pp. 5–37, 2006.

[12] F. V. Fomin and D. M. Thilikos, "An annotated bibliography on guaranteed graph searching," *Theoretical Computer Science*, vol. 399, no. 3, pp. 236–245, 2008.

[13] N. Megiddo, S. L. Hakimi, M. R. Garey, D. S. Johnson, and C. H. Papadimitriou, "The complexity of searching a graph," *Journal of the ACM*, vol. 35, no. 1, pp. 18–44, 1988.

[14] A. S. LaPaugh, "Recontamination does not help to search a graph," *Journal of the ACM*, vol. 40, no. 2, pp. 224–245, 1993.

[15] B. Yang, D. Dyer, and B. Alspach, "Sweeping graphs with large clique number," *Lecture notes in computer science*, pp. 908–920, 2004.

[16] F. V. Fomin and D. M. Thilikos, "On the monotonicity of games generated by symmetric submodular functions," WG 2001: 177-188, 2001.

[17] G. Hollinger, A. Kehagias, and S. Singh, "GSST: Anytime guaranteed search," *Autonomous Robots*, vol. 29, no. 1, pp. 99–118, 2010.

[18] A. Kolling, A. Kleiner, M. Lewis, and K. Sycara, "Solving pursuit-evasion problems on height maps," in *ICRA2010 Workshop: Search and Pursuit/Evasion in the Physical World: Efficiency, Scalability, and Guarantees*, 2010.

[19] ——, "Pursuit-evasion in 2.5d based on team-visibility," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2010, pp. 4610 – 4616.

[20] A. Kleiner, A. Kolling, M. Lewis, and K. Sycara, "Hierarchical visibility for guaranteed search in large-scale outdoor terrain," *Autonomous Agents and Multi-Agent Systems*, pp. 1–36, 2011.

[21] A. Kolling and S. Carpin, "Probabilistic Graph-Clear," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2009, pp. 3508–3514.

[22] ——, "Pursuit-evasion on trees by robot teams," *IEEE Transactions on Robotics*, vol. 26, no. 1, pp. 32–47, 2010.

[23] L. Barrière, P. Flocchini, P. Fraigniaud, and N. Santoro, "Capture of an intruder by mobile agents," in *Proceedings of the Fourteenth Annual ACM Symposium on Parallel Algorithms and Architectures*. New York, NY, USA: ACM Press, 2002, pp. 200–209.

[24] D. Dereniowski, "Connected searching of weighted trees," *Mathematical Foundations of Computer Science 2010*, pp. 330–341, 2010.

[25] I. Suzuki and M. Yamashita, "Searching for a mobile intruder in a polygonal region," *SIAM Journal on Computing*, vol. 21, no. 5, pp. 863–888, 1992.

[26] S. M. LaValle, D. Lin, L. Guibas, J.-C. Latombe, and R. Motwani, "Finding an unpredictable target in a workspace with obstacles," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 1997, pp. 737–742.

[27] L. J. Guibas, J.-C. Latombe, S. M. LaValle, D. Lin, and R. Motwani, "A visibility-based pursuit-evasion problem," *International Journal of Computational Geometry and Applications*, vol. 9, pp. 471–494, 1999.

[28] S. Sachs, S. M. LaValle, and S. Rajko, "Visibility-based pursuit-evasion in an unknown planar environment." *The International Journal of Robotics Research*, vol. 23(1), pp. 3–26, 2004.

[29] N. M. Stiffler and J. M. O'Kane, "A complete algorithm for visibility-based pursuit-evasion with multiple pursuers," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2014, pp. 1660–1667.

[30] ——, "Pursuit-evasion with fixed beams," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2016, pp. 4251 – 4258.

[31] K. J. Obermeyer, A. Ganguli, and F. Bullo, "A complete algorithm for searchlight scheduling," *International Journal of Computational Geometry and Applications*, vol. 21, no. 1, pp. 101–130, 2011.

[32] A. Efrat, L. J. Guibas, S. Har-Peled, D. C. Lin, J. S. B. Mitchell, and T. M. Murali, "Sweeping simple polygons with a chain of guards," in *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, 2000, pp. 927–936.

[33] X. Tan, "Sweeping simple polygons with the minimum number of chain guards," *Information processing letters*, vol. 102, no. 2-3, pp. 66–71, 2007.

[34] S. Bopardikar, F. Bullo, and J. P. Hespanha, "On discrete-time pursuit-evasion games with sensing limitations," *IEEE Transactions on Robotics*, vol. 24, no. 6, pp. 1429–1439, 2008.

[35] R. Isaacs, *Differential Games*. Wiley, New York, NY, 1965.

[36] J. W. Durham, A. Franchi, and F. Bullo, "Distributed pursuit-evasion without mapping or global localization via local frontiers," *Autonomous Robots*, vol. 32, no. 1, pp. 81–95, 2012.

[37] B. Monien and I. H. Sudborough, "Min cut is NP-complete for edge weighted trees," *Theoretical Computer Science*, vol. 58, no. 1-3, pp. 209–229, 1988.

[38] S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge University Press, 2006, available at http://planning.cs.uiuc.edu/.

[39] S. K. Wismath, "Computing the full visibility graph of a set of line segments," *Information processing letters*, vol. 42, no. 5, pp. 257–261, 1992.

[40] T. Lozano-Pérez and M. Wesley, "An algorithm for planning collision-free paths among polyherdral obstacles," *Communications of the ACM*, vol. 22, no. 10, pp. 560–570, 1979.

[41] G. Gallo and S. Pallottino, "Shortest path algorithms," *Annals of Operations Research*, vol. 13, no. 1, pp. 1–79, 1988.

[42] A. Kleiner and A. Kolling, "Guaranteed search with large teams of unmanned aerial vehicles," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2013, pp. 2977 – 2983.

[43] G. Hollinger, S. Singh, J. Djugash, and A. Kehagias, "Efficient multi-robot search for a moving target," *The International Journal of Robotics Research*, vol. 28, no. 1, pp. 201–219, February 2009.

[44] D. Bienstock and P. Seymour, "Monotonicity in graph searching," *Journal of Algorithms*, vol. 12, no. 2, pp. 239–245, 1991.

[45] M. Adler, H. Räcke, N. Sivadasan, C. Sohler, and B. Vöcking, "Randomized pursuit-evasion in graphs," *Combinatorics Probability and Computing*, vol. 12, no. 3, pp. 225–244, 2003.

[46] B. Tovar and S. M. LaValle, "Visibility-based pursuit-evasion with bounded speed," in *Proceedings of the Workshop on Algorithmic Foundations of Robotics*, 2006, pp. 475–489.

[47] N. M. Stiffler, A. Kolling, and J. M. O'Kane, "Persistent pursuit-evasion: The case of the preoccupied pursuer," in *Robotics and Automation, 2017 IEEE International Conference on*. IEEE, 2017, pp. 5027–5034.

Andreas Kolling is a scientist and technology lead at iRobot. He received a Ph.D. degree in electrical engineering and computer science in 2009 from the University of California, Merced, and a M.S. degree in computer science in 2006 and B.S. degree in mathematics in 2004 from Jacobs University, Bremen, Germany. From 2013 to 2016 he was an assistant professor in the Department of Automatic Control and Systems Engineering at The University of Sheffield, UK, where he led the multi-robot systems lab. From 2010 to 2012 he was a postdoctoral research fellow at the Robotics Institute at Carnegie Mellon University and at the University of Pittsburgh. His research interests include planning, multi-robot systems, and human-robot interaction. He has served as a general co-chair for DARS in 2016, as associate editor for ICRA and IROS since 2014, and as guest editor for Autonomous Robots for the special issue 'Distributed Robots: From Fundamentals to Applications'.

Alexander Kleiner received a M.Sc. degree (with distinction) in computer science at the Stafford University, UK, in 2000, a Ph.D degree (magna cum laude) in computer science at the University of Freiburg, Germany in 2008, and a docent degree (habilitation) at Linkoeping University, Sweden, in 2013. He was a postdoctoral fellow at Carnegie Mellon University in 2010 and at La Sapienza University, Rome, Italy in 2011. From 2011 to 2014 he was associate professor at Linkoeping University, Sweden, where he led the research group on collaborative robotics.From 2014 to 2017 he worked as a scientist and technology lead at iRobot, and now works as President of AI for FaceMap LLC. His research interests include collaborative robotics, navigation planning, and machine learning. From 2006 to 2014 he served as a member of the executive committee of RoboCup, and since 2008 as member of the IEEE Technical Committee on Safety Security and Rescue Robotics. He served as General Chair of the SSRR in 2013 and program chair in 2012. From 2012 to 2017 he served as an associate editor for IROS and ICRA.

Stefano Carpin is Professor of engineering at the University of California, Merced. He received Laurea (M.Sc.) and Ph.D. degrees in electrical engineering and computer science from the University of Padova, Italy in 1999 and 2003, respectively. From 2003 to 2006 he held faculty positions with Jacobs University Bremen, Germany. Since 2007 he has been with the School of Engineering at UC Merced, where he established and leads the robotics laboratory. His research interests include mobile and cooperative robotics for service tasks, and robot algorithms. He is an associate editor for the IEEE Transactions on Automation Science and Engineering and for the IEEE Robotics and Automation Letters. From 2006 to 2009 he was an elected executive member of the RoboCup federation. Under his supervision, teams participating in the RoboCup Virtual Robots Rescue competition won second place in 2006 and 2008, and first place in 2009.