To appear at CDC 2007
New Orleans, LA

Asynchronous Distributed Searchlight Scheduling

Karl J. Obermeyer Anurag Ganguli Francesco Bullo

***Abstract*— This paper develops and compares two asynchronous distributed scheduling algorithms for multiple controlled searchlights in nonconvex polygonal environments. A searchlight is a ray emitted by source location that (i) cannot penetrate the boundary of the environment and (ii) undergoes controlled slewing about its source location. Evaders move inside the environment along continuous trajectories and are detected precisely when they are on the searchlight ray at some time instant. The objective is for the searchlights to detect any evader in finite time and to do so using only local sensing and limited communication among them. The first algorithm we develop, called the Distributed One Way Sweep Strategy (DOWSS), is a distributed version of an algorithm described originally in 1990 by Sugihara et al [1]; this algorithm may be slow in "sweeping" the environment because only one searchlight slews at a time. Second, we develop a second algorithm, called the Parallel Tree Sweep Strategy (PTSS), in which searchlights sweep concurrently under the assumption that they are placed in appropriate locations; for this algorithm we establish linear completion time.**

I. Introduction

Consider a group of robotic agents guarding a nonconvex polygonal environment, e.g., a floor plan. For simplicity, we model the agents as point masses. Each agent is equipped with a single unidirectional sweeping sensor called a *searchlight* (imagine a ray of light such as a laser range finder emanating from each agent). A searchlight aims only in one direction at a time and cannot penetrate the boundary of the environment, but its direction can be changed continuously by the agent. A point is detected by a searchlight at some instant if and only if the point lies on the ray. An evader is any point which can move continuously with unbounded speed. The *Searchlight Scheduling Problem* is as follows.

> Find a schedule to slew a set of stationary searchlights such that any evader in an environment will necessarily be detected in finite time.

A searchlight problem instance consists of an environment and a set of stationary guard positions. A graphical description of our objective is given in Fig. 1.

To our knowledge the searchlight scheduling problem was first introduced in the inspiring paper by Sugihara, Suzuki and Yamashita in [1], which considers simple polygonal environments and stationary searchlights. The work in [2] extends [1] by considering guards with multiple searchlights (they call a guard possessing k searchlights a k-*searcher*) and polygonal environments containing holes. Some papers involving mobile searchlights, sometimes calling them *flashlights* or *beam detectors*, are [3], [4], [5], and [6]. Closely related is the Art Gallery Problem, namely the problem of finding a minimum set of locations from which the entire polygon is visible. Many variations on the Art Gallery Problem are wonderfully surveyed in [7], [8], and [9].

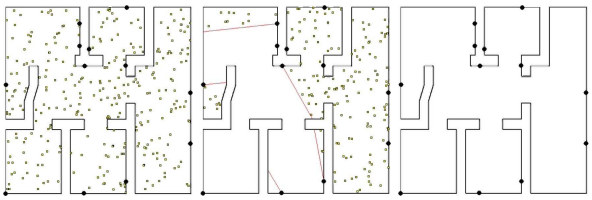

Fig. 1. Simulation results of the PTSS algorithm described in Section IV-B, executed by agents (black dots) in a polygon shaped like a typical floor plan. Left to right, moving evaders (small yellow squares) disappear as they are detected by searchlights (red). The cleared region grows until it encompasses the entire environment.

Assume now that each member of the group of guards is equipped with omnidirectional line-of-sight sensors. By a line-of-sight sensor, we mean any device or combination of devices that can be used to determine, in its line-of-sight, (i) the position or state of another guard, and (ii) the distance to the boundary of the environment. By omnidirectional, we mean that the field-of-vision for the sensor is 2π radians. There exist distributed algorithms to deploy asynchronous mobile robots with such omnidirectional sensors into nonconvex environments, and they are guaranteed to converge to fixed positions from which the entire environment is visible, e.g., [10] and [11]. At least one algorithm exists which guarantees the ancillary benefit of the final guard positions having a connected visibility graph ([11]). Once a set of guards seeing the entire environment has been established, it may be desired to continuously sweep the environment with searchlights so that any evader will be detected in finite time.

The main contribution of this paper is the development of two asynchronous distributed algorithms to solve the searchlight scheduling problem. Correctness and time completion bounds for nonconvex polygonal environments are discussed. The first algorithm, called the DOWSS (Distributed One Way Sweep Strategy, Section IV-A, is a distributed version of a known algorithm described originally in [1], but it can be very slow in clearing the entire environment because only one searchlight may slew at a time. On-line processing time required by agents during execution of DOWSS is relatively low, so that the expedience with which an environment can be cleared is essentially limited by the maximum angular speed searchlights may be slewn at. In an effort to reduce the time to clear the environment, we develop a second algorithm,

This work has been supported in part by AFOSR MURI Award F49620-02-1-0325, NSF Award CMS-0626457, and a DoD SMART fellowship.

Karl J. Obermeyer and Francesco Bullo are with the Department of Mechanical Engineering, University of California at Santa Barbara, Santa Barbara, CA 93106, USA, karl@engr.ucsb.edu, bullo@engineering.ucsb.edu

Anurag Ganguli is with the Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, and with the Department of Mechanical and Environmental Engineering, University of California at Santa Barbara, Santa Barbara, CA 93106, USA, aganguli@uiuc.edu

called the PTSS (Parallel Tree Sweep Strategy, Section IV-B), which sweeps searchlights in parallel if guards are placed in appropriate locations. These locations are related to an environment partition with certain properties. That we analyze the time it takes to clear an environment, given a bound on the angular slewing velocity, is a unique feature among all papers involving searchlights to date. Finally, we discuss how DOWSS and PTSS can be combined with deployment and extended for environments with holes.

We begin with some technical definitions, statement of assumptions, and brief description of the known centralized algorithm called the one way sweep strategy (appears, e.g., in [1], [2], [4]). We then develop a partially asynchronous model, a distributed one way sweep strategy, and our new algorithm the parallel tree sweep strategy. Proofs of all results and a more algorithm details can be found in the report [12].

II. Preliminaries

A. Notation

We begin by introducing some basic notation. We let $\mathbb{R}$, $\mathbb{S}^1$, and $\mathbb{N}$ represent the set of real numbers, the circle, and natural numbers, respectively. Given two points $x, y \in \mathbb{R}^2$, we let $[x, y]$ signify the *closed segment* between x and y. Similarly, $]x, y[$ is the *open segment* between x and y, $[x, y[$ represents the set $]x, y[\cup\{x\}$ and $]x, y]$ is the set $]x, y[\cup\{y\}$. Also, we shall use P to refer to tuples of elements in $\mathbb{R}^2$ of the form $(p^{[0]}, \ldots, p^{[N-1]})$ (these will be the locations of the agents), where N denotes the total number of agents.

We now turn our attention to the environment we are interested in and to the concepts of visibility. Let Q be a simple polygonal environment, possibly nonconvex. By simple, we mean that Q does not contain any hole and the boundary does not intersect itself. Throughout this paper, n will refer to the number of edges of Q and r the number of reflex vertices. A point $q \in Q$ is *visible from* $p \in Q$ if $[p, q] \subset Q$. The *visibility set* $\mathcal{V}(p) \subset Q$ from a point $p \in Q$ is the set of points in Q visible from p. A *visibility gap* of a point p with respect to some region $R \subset Q$ is defined as any line segment $[a, b]$ such that $]a, b[\subset \operatorname{int}(R)$, $[a, b] \subset \partial\mathcal{V}(p)$, and it is maximal in the sense that $a, b \in \partial R$ (intuitively, visibility gaps block off portions of R not visible from p). The visibility graph $\mathcal{G}_{\text{vis}}$ of a set of agents P in environment Q is the undirected graph with P as the set of vertices and an edge between two agents if and only if they are visible to each other.

We now introduce some notation specific to the searchlight problem. An instance of the searchlight problem can be written as a pair (Q, P), where Q is an environment and P is a set of agent locations. For convenience, we will refer to the searchlight of the ith agent as $s^{[i]}$ (which is located at $p^{[i]} \in \mathbb{R}^2$), and $S = \{s^{[0]}, \ldots, s^{[N-1]}\}$ will be the set of all searchlights. $\theta^{[i]}$ will also denote the angle of the ith searchlight in radians from the positive horizontal axis. So, if we say, e.g., aim $s^{[i]}$ at point x, what we really mean is set $\theta^{[i]}$ equal to an angle such that the ith searchlight is aimed at x. Searchlights do not block visibility of other searchlights.

Definition 2.1 (Contamination and clarity): A point $x \in Q$ is *contaminated* if an undetected evader can be located at x, otherwise x is *clear*. A region is said to be *contaminated* if it contains a contaminated point, otherwise it is *clear*.

B. Problem description and assumptions

We now introduce the problem of interest. The *Distributed Searchlight Scheduling Problem* is to

> Design a distributed algorithm for a network of autonomous robotic agents in fixed positions, who will coordinate the slewing of their searchlights so that any evader in an environment will necessarily be detected in finite time. Furthermore, these agents are to operate using only information from local sensing and limited communication.

What is precisely meant by local sensing and limited communication will become clear in later sections. We make the following *standing assumptions* about every searchlight instance in this paper:

(i) The environment is a simple polygon with finitely many reflex vertices.

(ii) Every point in the environment is visible from some agent and there are a finite number $N \in \mathbb{N}$ of agents.

(iii) For every connected component of $\mathcal{G}_{\text{vis}}$, there is at least one agent located on the boundary of the environment.

C. One Way Sweep Strategy (OWSS)

This section describes informally the centralized recursive One Way Sweep Strategy (OWSS hereinafter) originally introduced in [1]. The reader is referred to [1] for a detailed description. Centralized OWSS also appears in [4] and [2]. OWSS is a method for clearing a subregion of a simple 2D region Q determined by the rays of searchlights. The subregions of interest are the so-called *semiconvex subregions* of Q *supported* by a set of searchlights at a given time and are defined as follows:

Definition 2.2 (Semiconvex subregion): Q is always a semiconvex subregion of Q supported by $\emptyset$. Furthermore, any $R \subset Q$ is a semiconvex subregion of Q supported by a set of searchlights S_{sup} if both of the following hold:

(i) It is enclosed by a segment of ∂Q and the rays of some of the searchlights in S_{sup}.

(ii) The interior of R is not visible from any searchlight in S_{sup}.

To clear an environment $\mathcal{E}$, that is a semiconvex subregion supported by $\emptyset$, we may begin by selecting an arbitrary searchlight on the boundary. The first searchlight selected to clear an environment is called the *root*. As the root slews it blocks off various semiconvex subregions which must be cleared by the help of other agents. The helpers in turn may require help clearing various semiconvex subregions, and helpers of helpers may require help, etc., so that a recursion tree is produced.

III. Asynchronous Network of Agents with Searchlights

In this section we lay down the sensing and communication model for the agents with searchlights. Each agent is able to sense the relative position of any point in its visibility set as well as identify visibility gaps on the boundary of

its visibility set. The agents' communication graph $\mathcal{G}_{\rm comm}$ is assumed connected. An agent can slew its searchlight continuously in any direction and turn it on or off.

Each of the N agents has a unique identifier (UID), say i, and a portion of memory dedicated to outgoing messages with contents denoted by $\mathcal{M}^{[i]}$. Agent i can broadcast its UID together with $\mathcal{M}^{[i]}$ to all agents within its communication region (defined differently in each algorithm). We assume a bounded time delay, $\delta > 0$, between a broadcast and the corresponding reception.

Each agent repeatedly performs the following sequence of actions between any two wake-up instants:

(i) SPEAK, that is, send a BROADCAST repeatedly at δ intervals, until it starts slewing;
(ii) LISTEN for a time interval at least δ;
(iii) PROCESS and LISTEN after receiving a valid message;
(iv) SLEW to an angle decided during PROCESS.

See Fig. 2 for a schematic illustration of the schedule.

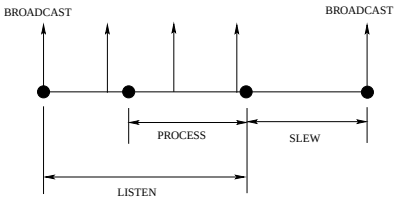

Fig. 2. Sequence of actions performed by an agent i in between two wake-up instants. Note that a BROADCAST is an instantaneous event taking place where there is a vertical pulse, where as the PROCESS, LISTEN and SLEW actions take place over an interval. The SLEW interval may be empty if the agent does not sweep.

Any agent i performing the SLEW action does so according to the discrete-time control system $\theta^{[i]}(t + \Delta t) = \theta^{[i]}(t) + u^{[i]}$, where the control is bounded in magnitude by $s_{\max}$. The control action depends on time, values of variables stored in local memory, and the information obtained from communication and sensing. The subsequent wake-up instant is the time when the agent stops performing SLEW and is not predetermined. This network model is identical to that used for distributed deployment in [10] and [11].

IV. Distributed Algorithms

Here we design distributed algorithms for a network of agents as described above, where no agent has global knowledge of the environment or locations of all other agents.

A. Distributed One Way Sweep Strategy (DOWSS)

Once one understands OWSS as in Section II-C, especially its recursive nature, performing one way sweep of an environment in a distributed fashion is fairly straightforward. We give here an informal description and supply a pseudocode in Table I (a more detailed pseudocode can be found in the companion tech. report [12]). In our discussion root/parent/child will refer to the relative location of agents in the simulated one way sweep recursion tree. In this tree, each node corresponds to a one way slewing action by some agent. A single agent may correspond to more than one node, but only one node at a time. To begin DOWSS, some agent (the root[1]), say i, can aim as far clockwise as possible and then begin slewing until it encounters a visibility gap. Paused at a visibility gap, agent i broadcasts a call for help to the network. For convenience, call the semiconvex subregion which i needs help clearing R. All agents not busy in the set of supporting searchlights $S_{\rm sup}$ (indeed at the zeroth level of recursion only the root is in $S_{\rm sup}$), who also know they can see a portion of $\mathrm{int}(R)$ but are not in $\mathrm{int}(R)$, volunteer themselves to help i. Agent i then chooses a child and the process continues recursively. In DOWSS as in Tab. I, an agent needing help always chooses the first child to volunteer, but some other criteria could be used, e.g., who sees the largest portion of R. Whenever a child is finished helping, i.e., clearing a semiconvex subregion, it reports to its parent so the parent knows they may continue slewing.

The only subtle part of DOWSS is getting agents to recognize, without global knowledge of the environment, that they see the interior of a particular semiconvex subregion which some potential parent needs help clearing. More precisely, suppose some agent s must decide whether to respond as a volunteer to agent i's help request to clear a semiconvex subregion R. Agent s must calculate if it actually satisfies $p^{[l]} \notin \mathrm{int}(R)$ and $\mathrm{int}(R) \cap \mathcal{V}(p^{[l]}) \neq \emptyset$. This is accomplished by agent i sending along with its help request an oriented polyline ψ (in SPEAK section of Tab. I). By an oriented polyline we mean that ψ consists of a set of points listed according to some orientation convention, e.g., so that if one were to walk along the points in the order listed, then the interior of R would always be to the right. The polyline encodes the portion of ∂R which is not part of $\partial\mathcal{E}$ and the orientation encodes which side of ψ is the interior of R. Notice that for this to work, all agents must have a common reference frame. Whenever the root broadcasts a polyline, it is just a line segment, but as recursion becomes deeper, an agent needing help may have to calculate a polyline consisting of a portion of its own beam and its parent's polyline. The polyline may even close on itself and create a convex polygon. Examples of these scenarios are illustrated by in Fig 3. We conclude our description of DOWSS with the following theorem.

Theorem 4.1 (Correctness of DOWSS): Given a simple polygonal environment $\mathcal{E}$ and agent positions $P = (p^{[0]}, \ldots, p^{[N-1]})$, let the following conditions hold:

(i) the standing assumptions are satisfied;
(ii) all agents $i \in \{0, \ldots, N-1\}$ have a common reference frame;
(iii) $p^{[0]} \in \partial\mathcal{E}$;
(iv) the agents operate under DOWSS.

Then $\mathcal{E}$ is cleared in finite time.

We now give an upper bound on the time it takes DOWSS to clear the environment assuming the searchlights slew at some constant angular velocity ω, and that communication and processing time are negligible.

Lemma 4.2 (DOWSS Time to Clear Environment): Let agents in a network executing DOWSS slew their searchlights with angular speed ω. Then the time required

[1]The root could be chosen by any leader election scheme, e.g., a predetermined or lowest UID.

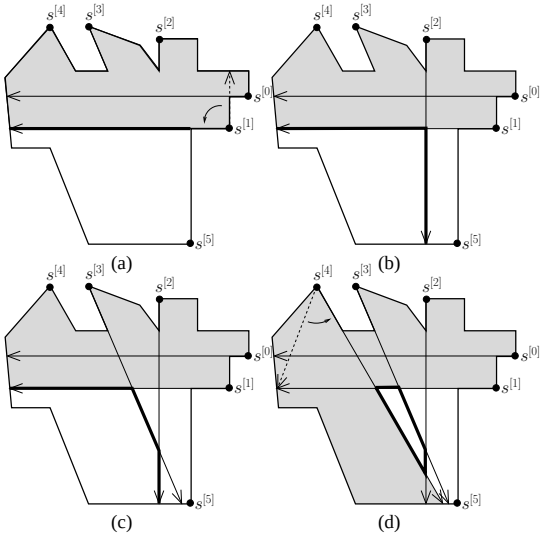

Fig. 3. An example execution of DOWSS. The configuration in (a) results from $s^{[0]}$ clearing the very top of the region with help of $s^{[2]}$, $s^{[3]}$, and $s^{[4]}$ followed by $s^{[1]}$ attempting to clear the semiconvex subregion below where $s^{[0]}$ is aimed. When $s^{[1]}$ gets stuck, it requests help by broadcasting the thick black polyline in (a), in this case just a line segment. $s^{[2]}$ then helps $s^{[1]}$ but gets stuck right off, so it broadcasts the thick black polyline shown in (b). Next $s^{[3]}$ helps $s^{[2]}$ but gets stuck and broadcast the polyline in (c). Similarly $s^{[4]}$ broadcasts the polyline in (d), in this case a convex polygon, which only $s^{[5]}$ can clear. In general, information passed between agents during any execution of DOWSS will be in the form of either an oriented line segment (a), a general oriented polyline (b and c), or a convex polygon (d).

TABLE I

ASYNCHRONOUS SCHEDULE FOR DOWSS (CF , 2, 3)

Name:	DOWSS
Goal:	Agents in the network coordinate their searchlight slewing to clear an environment $\mathcal{E}$.
Assumes:	Agents are stationary and have a completely connected communication topology with no packet loss. Sweeping is initialized by a root.

For time $t > 0$, each agent executes the following actions between any two wake up instants according to the schedule in Section III:

SPEAK

Broadcast either
- (i) a request for help,
- (ii) a message to engage a child, or
- (iii) a signal of task completion to a parent.

LISTEN

Listen for either
- (i) a help request from a potential parent,
- (ii) volunteers to help,
- (iii) engagement by parent, or
- (iv) current child reporting completion.

PROCESS
- (i) Use oriented polyline from potential parent with information from sensing to check if able to help, or
- (ii) if engaged, compute wayangles, visibility gaps and oriented polylines.

SLEW
- (i) Aim at start wayangle and switch searchlight on,
- (ii) slew to next wayangle, or
- (iii) slew to finish wayangle and switch searchlight off.

to clear an environment with r reflex vertices is no greater than $\frac{2\pi}{\omega}\frac{1-r^N}{1-r}$.

It is not known whether this bound is tight, but at least examples as in Fig. 4 can be constructed where DOWSS and OWSS run in $\mathcal{O}(r^2)$ ($\Rightarrow \mathcal{O}(n^2)$) time if guards are chosen malevolently. A key point is that DOWSS and OWSS do not specify (i) how to place guards given an environment, or (ii) how to optimally choose guards at each step given a set of guards. These are interesting unsolved problems in their own right which we do not explore in this paper.

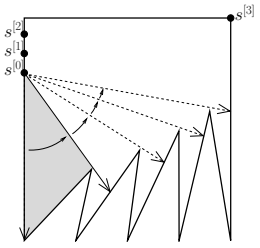

Fig. 4. An example from a class of searchlight instances for which malevolent guard choice (consecutive order of UIDs) in OWSS or DOWSS implies time to clear the environment is $\mathcal{O}(r^2)$ (and therefore $\mathcal{O}(n^2)$).

Another performance measure of a distributed algorithm is the size of the messages which must be communicated.

Lemma 4.3 (DOWSS Message Size): If the environment has n sides and r reflex vertices, the polyline (passed as a message between agents during DOWSS) consists of a list of no more than $r+1$ points in $\mathbb{R}^2$. Furthermore, since $r \leq n-3$, the list consists of no more than $n-2$ points in $\mathbb{R}^2$.

That DOWSS allows flexibility in guard positions (only standing assumptions required) may be an advantage if agents are immobile. However, DOWSS only allowing one searchlight slewn at a time is a clear disadvantage when time to clear the environment is to be minimized. This lead us to design the algorithm in the next section.

B. Positioning Guards for Parallel Sweeping

The DOWSS algorithm in the previous section is a distributed message-passing and local sensing scheme to perform scheduling given *a priori* the location of the searchlights. Given an arbitrary positioning, time to completion of DOWSS can be large; see Lemma 4.2 and Fig. 4.

The algorithm we design in this section, called the Parallel Tree Sweep Strategy (PTSS), provides a way of choosing searchlight locations and a corresponding schedule to achieve faster clearing times. PTSS works roughly like this: According to some technical criteria described below, the environment is partitioned into regions called cells with one agent located in each cell. Additionally, the network possesses a distributed representation of a rooted tree. By distributed representation we mean that every agent knows who its parent and children are. Using the tree, agents slew their searchlights in a way that expands the clear region from the root out to the leaves, thus clearing the entire environment. Since agents may operate in parallel, time to clear the environment is linear in the height of the tree and thus $\mathcal{O}(n)$. Guaranteed linear time to completion is a clear advantage over DOWSS which can be quadratic or worse

(see Lemma 4.2 and Fig. 4). Before describing PTSS more precisely, we need a few definitions.

Definition 4.4: (i) A set $\mathcal{S} \subset \mathbb{R}^2$ is *star-shaped* if there exists a point $p \in \mathcal{S}$ with the property that all points in $\mathcal{S}$ are visible from p. The set of all such points of a given star-shaped set $\mathcal{S}$ is called the *kernel* of $\mathcal{S}$ and is denoted by $\ker(\mathcal{S})$.

(ii) Given a compact subset $\mathcal{E}$ of $\mathbb{R}^2$, a *partition* of $\mathcal{E}$ is is a collection of sets $\{\mathcal{P}^{[0]}, \ldots, \mathcal{P}^{[N-1]}\}$ such that $\cup_{i=0}^{N-1} \mathcal{P}^{[i]} = \mathcal{E}$ where $\mathcal{P}^{[i]}$'s are compact, simply connected subsets of $\mathcal{E}$ with disjoint interiors. $\{\mathcal{P}^{[0]}, \ldots, \mathcal{P}^{[N-1]}\}$ will be called *cells* of the partition.

For our purposes a *gap* (which visibility gap is a special case of) will refer to any segment $[q, q']$ with $q, q' \in \partial\mathcal{E}$ and $]q, q'[\in \mathring{\mathcal{E}}$. The cells of the partitions we consider will be separated by gaps.

Definition 4.5 (PTSS partition): Given a simple polygonal environment $\mathcal{E}$, a partition $\{\mathcal{P}^{[0]}, \ldots, \mathcal{P}^{[N-1]}\}$ is a *PTSS partition* if the following conditions are true:

(i) $\mathcal{P}^{[i]}$ is a star-shaped cell for all $i \in \{0, \ldots, N-1\}$;
(ii) the dual graph[2] of the partition is a tree;
(iii) a root, say $\mathcal{P}^{[0]}$, of the dual graph may be chosen so that $\ker(\mathcal{P}^{[0]}) \cap \partial\mathcal{E} \neq \emptyset$, and for any node other than the root, say $\mathcal{P}^{[k]}$ with parent $\mathcal{P}^{[j]}$, we have that $(\mathcal{P}^{[j]} \cap \mathcal{P}^{[k]}) \cap \ker(\mathcal{P}^{[k]}) \cap \partial\mathcal{E} \neq \emptyset$.

Definition 4.6: Given a PTSS partition $\{\mathcal{P}^{[0]}, \ldots, \mathcal{P}^{[N-1]}\}$ of $\mathcal{E}$ and a root cell $\mathcal{P}^{[0]}$ of the partition's dual graph satisfying the properties discussed in Definition 4.5, the corresponding (rooted) *PTSS tree* is defined as follows:

(i) the node set $(p^{[0]}, \ldots, p^{[N-1]})$ is such that $p^{[0]} \in \ker(\mathcal{P}^{[0]}) \cap \partial\mathcal{E}$ and for $k > 1$, $p^{[k]} \in (\mathcal{P}^{[j]} \cap \mathcal{P}^{[k]}) \cap \ker(\mathcal{P}^{[k]}) \cap \partial\mathcal{E}$, where $\mathcal{P}^{[j]}$ is the parent of $\mathcal{P}^{[k]}$ in the dual graph of the partition;
(ii) there exists an edge $(p^{[j]}, p^{[k]})$ if and only if there exists an edge $(\mathcal{P}^{[j]}, \mathcal{P}^{[k]})$ in the dual graph.

We now describe two examples of PTSS partitions seen in Fig. 5. The left configuration in Fig. 5 results from what we call a Reflex Vertex Straddling (RVS hereinafter) deployment. RVS deployment begins with all agents located at the root followed by one agent moving to the furthest end of each of the root's visibility gaps, thus becoming children of the root. Likewise, further agents are deployed from each child to take positions on the furthest end of the children's visibility gaps located across the gaps dividing the parent from the children. In this way, the root's cell in the PTSS partition is just its visibility set, but the cells of all successive agents consist of the portion of the agents' visibility sets lying across the gaps dividing their cells from their respective parents' cells. It is easy to see that in final positions resulting from an RVS deployment, agents see the entire environment.

Lemma 4.7: RVS deployment requires, in general, no more than $r+1 \leq n-2$ agents to see the entire environment from their final positions. In an orthogonal environment, no more than $\frac{n}{2} - 2$ agents are required.

[2]The dual graph of a partition is the graph with cells corresponding to nodes, and there is an edge between nodes if the corresponding cells share a curve of nonzero length.

See Fig. 1 for simulation results of PTSS executed by agents in an RVS configuration. The right configuration in Fig. 5 results from the deployment described in [11] in which an orthogonal environment is partitioned into convex quadrilaterals.

Lemma 4.8: The deployment described in [11] requires no more than $\frac{n}{2} - 2$ agents to see the entire (orthogonal) environment from their final positions.

Both the PTSS configurations in these examples may be generated via distributed deployment algorithms in which agents perform a depth-first, breadth-first, or randomized search on the PTSS tree constructed on-line. Please refer to [10] and [11] for a detailed description of these algorithms.

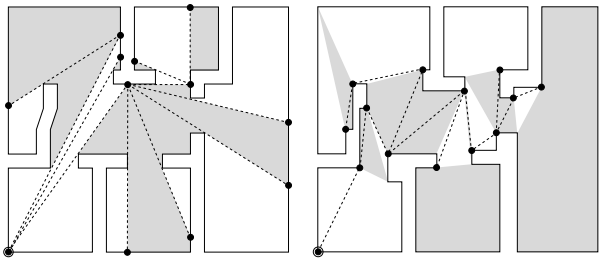

Fig. 5. Left are agent positions resulting from a Reflex Vertex Straddling (RVS) deployment. Right are agent positions resulting from the deployment described in [11] in which an orthogonal environment is partitioned into convex quadrilaterals. The PTSS partitions are shown by coloring the cells alternating grey and white (caution: grey does not depict clarity here). Dotted lines show edges of the PTSS tree where the circled agent is the root.

We now turn our attention to the pseudocode in Tab. II (a more detailed pseudocode can be found in the companion tech. report [12]) and describe PTSS more precisely. Suppose some agents are positioned in an environment according to a PTSS partition and rooted tree. PTSS begins by the root pointing its searchlight along a wall and then slewing away from the wall, sweeping over its cell, pausing whenever it encounters a gap. At a gap, the root and its child at that gap execute the protocol described in Fig. 6 in order to expand the clear region across the gap. The root's children do the same with their children, and so on. In this way, the clear region expands from the root to the leaves at which time the entire environment has been cleared. We arrive at the following lemma and correctness result.

Lemma 4.9 (Expanding a Clear Region Across a Gap): Suppose an environment is endowed with a PTSS partition and tree, and that agent i is a parent of agent j (see Fig. 6). Then a clear region may always be expanded across the gap from $\mathcal{P}^{[i]}$ to $\mathcal{P}^{[j]}$ by $s^{[j]}$ first aiming across the gap and waiting for $s^{[i]}$ to slew over the gap. Both agents may then continue clearing the remainder of their respective cells concurrently.

Theorem 4.10 (Correctness of PTSS): Given a simple polygonal environment $\mathcal{E}$ and agent positions $P = (p^{[0]}, \ldots, p^{[N-1]})$, let the following conditions hold:

(i) the standing assumptions are satisfied;
(ii) all agents $i \in \{0, \ldots, N\}$ are positioned in a PTSS partition and rooted tree with agent 1 as the root;
(iii) the agents operate under PTSS.

Then $\mathcal{E}$ is cleared in finite time.

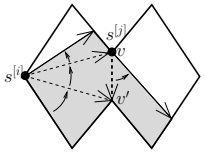

Fig. 6. Expanding a clear region (grey) across a gap (thick dashed segment [v, v']) from cell $\mathcal{P}^{[i]}$ to cell $\mathcal{P}^{[j]}$ may always be accomplished by the child ($s^{[j]}$) aiming across the gap and waiting for the parent ($s^{[i]}$) to slew over the gap. Both agents may then continue clearing the remainder of their respective cells.

TABLE II

ASYNCHRONOUS SCHEDULE FOR PTSS (CF FIG. 2, 6, 5)

Name:	PTSS
Goal:	Agents in the network coordinate their searchlight slewing to clear an environment $\mathcal{E}$.
Assumes:	Agents are statically positioned as nodes in a PTSS partition and tree, and each knows a priori the gaps of its cell and UIDs of the corresponding children and parent. Sweeping is initialized by the root.

For time $t > 0$, each agent executes the following actions between any two wake up instants according to the schedule in Section III:

SPEAK

Broadcast either

(i) a command for a child to aim across a gap,
(ii) a confirmation to a parent when aimed across gap, or
(iii) when finished slewing over a gap, a signal of completion to the child.

LISTEN

Listen for either

(i) instruction from a parent to aim across a gap,
(ii) confirmation from a child aimed across a gap, or
(iii) confirmation that parent has passed the gap.

PROCESS

When first engaged, compute wayangles where coordination with children will be necessary.

SLEW

(i) Aim at start wayangle and switch searchlight on,
(ii) slew to next wayangle, or
(iii) slew to finish wayangle and switch searchlight off.

Since multiple branches of the PTSS tree may be cleared concurrently, and using Lemmas 4.7 and 4.8, we have the next lemma (assuming processing and communication time are negligible, cf. Lemma 4.2).

Lemma 4.11 (PTSS Time to Clear Environment): Let the agents in a network executing PTSS slew their searchlights with angular speed ω. Then time required to clear an environment is

(i) linear in the height of the PTSS tree;
(ii) no greater than $\frac{2\pi}{\omega}(r+1) \leq \frac{2\pi}{\omega}(n-2)$ if agents are in final positions according to an RVS deployment;
(iii) no greater than $\frac{\pi}{\omega}(n-2)$ if agents are in final positions in an orthogonal polygon according to an RVS deployment or the deployment described in [11].

Looking at the SPEAK section of Tab. II, it is easy to see that message size is constant (cf. Lemma 4.3).

Lemma 4.12 (PTSS Message Size): Messages passed between agents executing PTSS have constant size.

Requiring guards to be situated in a PTSS tree is more restrictive than the mere standing assumptions required by DOWSS, but the time savings using PTSS over DOWSS can be considerable. Despite our two example schemes to construct a PTSS tree, it is not clear how to construct one which clears an environment in minimum time among all possible PTSS trees. It is also not clear how to optimally choose the root of the tree (point of deployment). However, if the environment layout is known a priori and one may choose the root location, then an exhaustive strategy may be adopted whereby all possible root choices are compared.

V. CONCLUSIONS

In this paper we have provided two solutions to the distributed searchlight scheduling problem. DOWSS requires that the guards satisfy the standing assumptions, has message size $\mathcal{O}(n)$, and sometimes requires time $\mathcal{O}(r^2)$ to clear an environment. PTSS requires that the agents be positioned according to a PTSS tree, has constant message size, and requires time linear in the height of the PTSS tree. We have given two procedures for constructing PTSS trees, one requiring no more than $r \leq n-3$ guards for a general polygonal environment, and two requiring no more than $\frac{n-2}{2}$ guards for an orthogonal environment. Guards slew through a total angle no greater than 2π, so the upper bounds on the time for PTSS to clear an environment with these partitions are $\frac{2\pi}{\omega}r \leq \frac{2\pi}{\omega}(n-3)$ and $\frac{\pi}{\omega}(n-2)$, respectively. Because PTSS allows searchlights to slew concurrently, it generally clears an environment much faster than DOWSS. However, a direct comparison is not appropriate since DOWSS does not specify how to choose guards whereas PTSS does.

REFERENCES

[1] K. Sugihara, I. Suzuki, and M. Yamashita, "The searchlight scheduling problem," *SIAM Journal on Computing*, vol. 19, no. 6, pp. 1024–1040, 1990.

[2] M. Yamashita, I. Suzuki, and T. Kameda, "Searching a polygonal region by a group of stationary k-searchers," *Information Processing Letters*, vol. 92, no. 1, pp. 1–8, 2004.

[3] B. P. Gerkey, S. Thrun, and G. Gordon, "Visibility-based pursuit-evasion with limited field of view," *International Journal of Robotics Research*, vol. 25, no. 4, pp. 299–315, 2006.

[4] M. Yamashita, H. Umemoto, I. Suzuki, and T. Kameda, "Searching for mobile intruders in a polygonal region by a group of mobile searchers," *Algorithmica*, vol. 31, pp. 208–236, 2001.

[5] B. Simov, G. Slutzki, and S. M. LaValle, "Pursuit-evasion using beam detection," in *IEEE Int. Conf. on Robotics and Automation*, 2000.

[6] J. H. Lee, S. M. Park, and K. Y. Chwa, "Simple algorithms for searching a polygon with flashlights," *Information Processing Letters*, vol. 81, no. 5, pp. 265–270, 2002.

[7] J. Urrutia, "Art gallery and illumination problems," in *Handbook of Computational Geometry* (J. R. Sack and J. Urrutia, eds.), pp. 973–1027, Amsterdam, the Netherlands: North-Holland, 2000.

[8] J. O'Rourke, *Art Gallery Theorems and Algorithms*. Oxford, UK: Oxford University Press, 1987.

[9] T. C. Shermer, "Recent results in art galleries," *IEEE Proceedings*, vol. 80, no. 9, pp. 1384–1399, 1992.

[10] A. Ganguli, J. Cortés, and F. Bullo, "Distributed deployment of asynchronous guards in art galleries," in *American Control Conference*, (Minneapolis, MN), pp. 1416–1421, June 2006.

[11] A. Ganguli, J. Cortés, and F. Bullo, "Visibility-based multi-agent deployment in orthogonal environments," in *American Control Conference*, (New York), pp. 3426–3431, July 2007.

[12] K. J. Obermeyer, A. Ganguli, and F. Bullo, "Asynchronous distributed searchlight scheduling." Available electronically at `http://arxiv.org/abs/cs/0701077`, Jan. 2007.