1

Distributed coverage of nonconvex environments

Anurag Ganguli[1], Jorge Cortés[2], and Francesco Bullo[3]

[1] Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, IL 61801, USA, and the Department of Mechanical Engineering, University of California, Santa Barbara, CA 93106, USA, aganguli@uiuc.edu
[2] Department of Applied Mathematics and Statistics, University of California, Santa Cruz, CA 95064, USA, jcortes@ucsc.edu
[3] Department of Mechanical Engineering, University of California, Santa Barbara, CA 93106, USA, bullo@engineering.ucsb.edu

1.1 Introduction

Sensor networks and multi-agent robotic systems have been receiving increasing attention in recent times. This is due in no small part to the remarkable advances made in recent years in the development of small, agile, relatively inexpensive sensor nodes with mobile and networking capabilities. These sensor nodes are envisioned to be the basic components of complex networks intended to perform a wide variety of tasks. These include search and rescue, exploration, environmental monitoring, location-aware computing, and the maintaining of structures. The potential advantages of employing arrays of robotic sensors are numerous. For instance, certain tasks are difficult, if not impossible, when performed by a single agent. Further, a group of agents inherently provides robustness to failures of single agents or communication links.

The existence of such motion-enabled sensing devices and the anticipated development of still more advanced versions raise compelling questions. A particularly important issue is whether large numbers of such small autonomous devices will be successfully deployed as a search team to cooperatively carry out a prescribed task reliably, robustly and adaptively, without a centralized controller and with limited communications among its members.

Motivated by these future scenarios, this paper focuses on algorithms for **visually-guided agents,** i.e., mobile robotic agents with line-of-sight sensing and communication capabilities, to solve a distributed version of the **Art Gallery Problem**. In the remainder of the introduction, we describe the problem in its original context, broadly highlight the characteristics of visually-guided agents and reformulate the original problem with respect to visually-guided agents.

Art Gallery and Illumination Problems

The classic Art Gallery Problem, was introduced by Klee and first analyzed by Chvátal, see [1, 2]. This combinatorial and geometric problem is stated as follows:

> Imagine placing guards inside an art gallery in the shape of a nonconvex polygon with n vertices: how many guards are required and where should they be placed in order for each point in the gallery to be visible by at least one guard?

The Art Gallery Theorem [1] states that $\lfloor n/3 \rfloor$ guards are sufficient and sometimes necessary to guard any polygon with n vertices. An elegant "triangulation + coloring" proof was proposed by Fisk [3]. The proof is constructive, i.e., it includes an efficient placement algorithm; an illustration is provided in Figure 1.1.

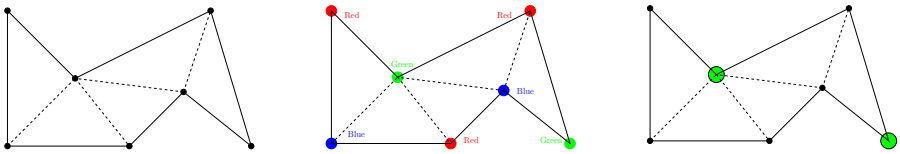


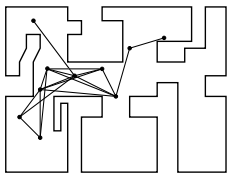
Fig. 1.1. Fisk's Algorithm: 1: triangulate the polygon (see dashed lines). 2: three-color the vertices so that each triangle has all three colors (possible because the "dual graph" is a tree). 3: select the color with smallest cardinality and place guards at the corresponding vertices (see the two guards in right picture).

Fisk's solution is, however, **centralized**, that is, it assumes that a central processor has global knowledge of the environment and that guards can be placed in desired locations without accounting for sensor-based and/or communication-based deployment.

Networks of visually-guided agents

Taking the Art Gallery Problem as a starting point, we consider a novel scenario where the guards are robotic agents in a simple nonconvex environment and are equipped with "line-of-sight" sensing and communication capabilities. In other words, our version of the Art Gallery Problem is different from its classic counterpart by the use of distributed feedback and communication protocols, rather than open-loop centralized computation.

We consider agents moving in a nonconvex planar or spatial environment, and make the following assumptions: (A1) Each agent is equipped with an "omnidirectional sensor." By this we mean a device or combination of devices (omnidirectional cameras, range and proximity sensors)



Inter-agent communication graph

that sense distance to the environment boundaries and to other agents within unobstructed lines of sight; (A2) The agents do not know the entire environment and their positions in it; (A3) Depending on the problem at hand, the guards are also allowed to exchange information with agents within line-of-sight through an asynchronous communication channel with delays and packet losses. This communication graph is depicted on the side; (A4) The agents are assumed to evolve asynchronously, i.e., a different sensing/communication/control schedule is allowed for each agent; (A5) For simplicity's sake, we model these agents as point masses with first-order dynamics. Assumptions (A1) through (A5) characterize what we refer to as **visually-guided agents**.

Illuminating art galleries via incremental partition and deployment

Combining the discussion in the earlier subsections, we obtain the following version of the Art Gallery Problem: starting from arbitrary positions, how should the agents move (and what should they communicate) in order to reach final positions such that each point of the environment is visible to at least one agent. This is what we refer to as the **distributed art-gallery deployment problem.** Remarkably, the difficulty of this problem is inherently due to the communication and sensing constraints: the agents are not given a map of the environment and no central entity controls them.

The proposed algorithms allow for sensor-based, distributed, asynchronous execution and guaranteed visibility is achieved when the number of agents is at least $\lfloor n/2 \rfloor$. The algorithm is organized in three steps:

[Geometric Structure]: first, we show that any simple nonconvex polygon can be partitioned into star-shaped polygons in an incremental distributed way. This induces a graph, the *vertex-induced tree*, as follows: every star-shaped polygon in the partition is a node and edges between nodes exist only when the corresponding polygons are contiguous;

[Distributed Information Processing]: second, we design appropriate distributed algorithms to manage the geographic information obtained by the network of agents. This entails deciding what information needs to be stored by what agent and how it needs to be transmitted and updated;

[Local Navigation and Global Exploration]: third and final, we devise navigation algorithms for two purposes: (i) to traverse edges of the vertex-induced tree, i.e., to move individual agents between contiguous polygons, and (ii) to explore and deploy a group of agents over the nodes of the vertex-induced tree.

This combination of "geometric structure + information management + navigation algorithms" is the key idea that allows individual agents to explore

and traverse the nonconvex polygon only based on local sensing and communication. We refer to solutions of this form as **incremental partition and deployment algorithms**.

The rest of the paper is organized as follows. In Section 1.2, we present the literature related to our current work. Section 1.3 contains preliminaries and notation. In Section 1.4, we present the algorithm details. Finally, we conclude and talk about future directions of research in Section 1.5.

1.2 Related Work

The content of this paper is related to the works on map building and exploration, deployment of robotic networks, illumination and geometric optimization problems, and distributed algorithms. In the following, we cite the works that are relevant, by subject or by the tools therein, to either the problem or the approach in this paper or both.

Map building and exploration

The robotics literature is abound in works on map building and exploration of unknown environments. However, the most relevant to the problem at hand include topological exploration of graph-like environments by single and multiple robots. In [4], a single robot with a marker explores such an environment via a depth-first linear time algorithm. While at a node of the graph, the robot has the ability to identify the neighboring nodes, order them in a consistent way, remember the last node visited and drop a marker to designate that a given node has already been explored. Topological exploration with multiple robots is the subject of [5]. Multiple robots, each equipped with a marker, explore the map independently. They communicate with robots located at the same node. The robots start at the same node, plan partition of work and rendezvous schedule (by exchanging messages), explore a portion of the environment and return to a predetermined location where they merge their maps. The process is repeated till the maps with each of the robots is isomorphic with the the world map. Multi-robot exploration of an unknown environment while reducing the odometry error has also been studied [6]. Here, exploration proceeds via constructing partitions of the environment into triangles or quadrilaterals, depending on whether the diameter of the environment is large compared to the range of the sensor, and then moving along the dual graph of the partition.

Deployment of robotic networks

Some related works on deployment include [7], where an incremental heuristic for deployment is proposed, [8] where distributed algorithms for coverage control based on Voronoi partitions are designed, and [9], in which the relevance of

random walk on graphs is discussed (the environment and its graphical representation are assumed known a priori, and general strategies are evaluated via Monte Carlo simulation). Coordinated deployment of multiple heterogeneous robots has also been studied in [10]. Deployment locations are user-specified after an initial map of the unknown environment has been built.

Illumination problems and geometric optimization

Illumination and art gallery problems are classic topics, e.g., see [11, 12, 13]. Coverage algorithms (for systems with binary, limited-range sensors) are surveyed in [14]. Next-best-view problems are discussed in [15]. Geometric optimization is a vast and exciting avenue of current research, see for example [16, 17]. Here, by geometric optimization, we mean an optimization problem induced by a collection of geometric objects. For example, in facility location problems service sites are spatially allocated to fulfill a specified request [18, 19]. These approaches mainly rely on centralized computation for a known static environment and are not applicable in a distributed, asynchronous, adaptive setting.

Distributed algorithms

The study of distributed algorithms is concerned with providing mathematical models, devising precise specifications for their behavior, and formally proving their correctness and complexity. Via an automata-theoretic approach, the reference [20] treats distributed consensus, resource allocation, communication, and data consistency problems. Numerical distributed asynchronous algorithms as networking algorithms, rate and flow control, and gradient descent flows are discussed in [21]. All these references do not typically address algorithms over ad-hoc dynamically changing networks. The recent work [22] proposes a model of distributed robotic network.

In addition, the proposed work is related to visibility-based pursuit-evasion problems, see [23, 24], although these works focus on single agents and not on distributed policies for groups of agents.

The sensing and communication abilities of each agent is attuned to the coordination problems at hand. The study of vision as a sensor in coordination problems is in its infancy; beside our work described below, only few preliminary references are available [24, 25]. Vision and, more generally, sensor-based coordination is instead a key interaction modality for animal networks.

1.3 Preliminaries and notation

We begin by introducing some basic notation. If p is a point in the polygon Q, we let $V(p)$ denote the set of visible points from p. A set S is *star shaped* if there exists $p \in S$ such that $S \subset V(p)$; if S is star shaped, we let $\ker(S)$ be its *kernel*, i.e., the set of points $k \in S$ such that $S \subset V(k)$. Finally, a *diagonal* of

a polygon Q is a segment inside Q connecting two vertices of Q (and therefore splitting Q into two polygons). A vertex of a polygon Q is *nonconvex* when the internal angle is strictly greater than π.

We consider a group of robotic agents modeled as point masses, moving in a simple nonconvex polygonal environment, Q. Each agent has a unique identifier UID, say i. Let p_i refer to the position of agent i. Each agent is equipped with an omnidirectional line-of-sight range sensor. Thus, the agent can sense its star-shaped visibility set $V(p_i)$. It can communicate with any other agent within line-of-sight and less than a certain distance r. The quantity r can be adjusted by the agent but is upper bounded, say by $R > 0$.

Each agent has access to some memory $\mathcal{M}_i$. By memory, we refer to all the necessary information that is not accessible via local sensing and communication. An agent i can broadcast its UID together with its memory contents to all agents inside its communication region. Such a broadcast is denoted by BROADCAST$(i, \mathcal{M}_i)$. It can also receive broadcasts from other agents. We also assume that there is a bounded time delay, $\delta > 0$, between a broadcast and the corresponding reception.

Every agent i repeatedly performs the following sequence of actions beginning at a time instant, say T_l^i:

(i) send repeated BROADCAST$(i, \mathcal{M}_i)$ after δ time intervals, until it starts moving;
(ii) LISTEN for a time interval equal to at least 2δ before processing the information;
(iii) PROCESS the necessary information. Also continue to LISTEN during this interval;
(iv) MOVE to a desired point.

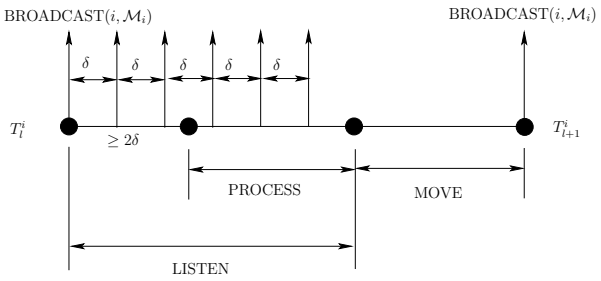


Fig. 1.2. Sequence of actions for agent i beginning at time T_l^i. Instantaneous BROADCAST$(i, \mathcal{M}_i)$ events are represented by vertical pulses. The MOVE interval might be empty if the agent does not move.The subsequent instant T_{l+1}^i is the time when the agent stops performing the MOVE action and it is not predetermined.

Agent i, in the MOVE state, is capable of moving at any time t according to the following discrete-time control system:

$$p_i(t + \Delta t) = p_i(t) + u_i,$$

where the control is bounded in magnitude by 1. The control action depends on time, on the memory $\mathcal{M}_i(t)$, and on the information obtained from communication and sensing. The subsequent wake-up instant T^i_{l+1} is the time when the agent stops performing the MOVE action and it is not predetermined. This model of visually-guided agents is similar in spirit to the **partially asynchronous network model** described in [21].

Given this model, the goal is to design a provably correct discrete-time algorithm which ensures that the agents converge to locations such that each point of the environment is visible to at least one agent. This is the **distributed art-gallery deployment problem** for visually-guided agents.

1.4 Distributed Art Gallery Deployment Problem

In this section we detail the **incremental partition and deployment algorithms** described in the introduction. We begin by describing a partition of a given simply connected nonconvex environment into star-shaped polygons and the graph that such a partition induces.

1.4.1 The vertex-induced partition and tree

Given a nonconvex polygon Q without holes and a vertex s of it, we compute a list $\{P_1, \ldots, P_m\}$ of star-shaped polygons composing a partition of Q and a list $\{k_1, \ldots, k_m\}$ of kernel points for each star-shaped polygon $\{P_1, \ldots, P_m\}$. The computation of these quantities is discussed in the following algorithm and is illustrated in Figure 1.3.

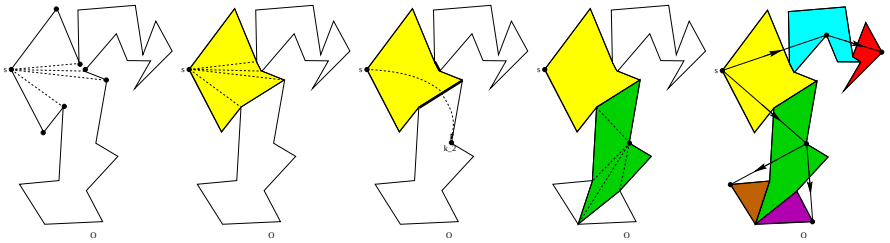


Fig. 1.3. Computation of the vertex-induced partition and tree in 5 steps.

`Vertex-Induced Partition and Tree Algorithm`

1: set $k_1 = s$, and collect all vertices of Q visible from k_1 (see Fig. 1.3(a))

2: let P_1 be the polygon determined by these vertices (by definition $k_1 \in \ker(P_1)$) (see Fig. 1.3(b))
3: identify the edges of P_1 that are diagonals of Q; call them **gaps**. For all gaps, place a new point, say k_2, across the gap at a new vertex of Q such that k_2 sees the gap (see Fig. 1.3(c))
4: repeat last three steps for new point k_2, until all gaps have been crossed (see Fig. 1.3(d))
5: define edges starting from s going to all kernel points and crossing all edges (see Fig. 1.3(e))

We refer to the list $\{P_1, \ldots, P_m\}$ computed in the algorithm as the **vertex-induced partition**. The algorithm computes not only the partition and a list of kernel points, but also a collection of edges connecting the kernel points. In other words, we also computed a directed graph, the **vertex-induced tree**, denoted by $\mathcal{G}_Q(s)$: the nodes of this directed graph are $\{k_1, \ldots, k_m\}$ and an edge exists between any two vertices k_i, k_j if and only if $P_i \cap P_j$ is a diagonal of Q. Note that $k_1 = s$; we refer to this node as the root of $\mathcal{G}_Q(s)$. We now state some important properties of the vertex-induced tree.

Proposition 1. *Given a polygon Q without holes and a vertex s, the following statements hold:*

(i) the directed graph $\mathcal{G}_Q(s)$ is a rooted tree;
(ii) the maximum number of nodes in the vertex-induced tree is less than or equal to $\lfloor \frac{n}{2} \rfloor$, where n is the number of vertices in Q.

Proof. The fact that $\mathcal{G}_Q(s)$ is a tree is a consequence of the fact that Q has no holes. Since s is designated as the root, $\mathcal{G}_Q(s)$ is a rooted tree. This proves statement (i). To prove statement (ii), notice the set of nodes of $\mathcal{G}_Q(s)$ belong to the vertices of Q. Also, by construction no two adjacent vertices of Q can both belong to the node set $\{k_1, \cdots, k_m\}$. Since the number of vertices of Q is n, it follows that number of nodes of $\mathcal{G}_Q(s)$ is less than or equal to $\lfloor \frac{n}{2} \rfloor$.

It is clear from the construction of the vertex-induced tree that, if we design a distributed algorithm to place agents on each node of the tree, then we will have solved the distributed art-gallery deployment problem.

Remark 1. If we can deploy the agents over the kernel points, then we will have solved the art-gallery deployment problem requiring $\lfloor n/2 \rfloor$ agents in the worst case, which is in general more than the $\lfloor n/3 \rfloor$ number required if the entire environment were known a priori. This is not surprising considering the weaker assumption of no global knowledge that we make while posing the problem.

Local node-to-node navigation algorithms

Note that by virtue of the constructions in the previous section, we have converted the original problem into a graph "navigation and deployment" problem. We now describe algorithms to plan paths between neighboring nodes of the vertex-induced tree. In a rooted tree, every neighbor of a node is either a child or the parent. Therefore, we present two simple informal descriptions.

```
Move-to-Child Algorithm
1: compute the mid-point of the gap between the node and the child
2: go to the mid-point
3: compute the nearest vertex from which the entire gap is visible and which
   is across the gap
4: go to that vertex
```

```
Move-to-Parent Algorithm
1: compute the mid-point of the gap between the node and the parent
2: go to the mid-point of the gap
3: from the mid-point, go to the vertex representing the parent node
```

Figure 1.4 shows paths between parents and children as computed by the previous two algorithms. It is easy to see that navigation is very simple if sufficient information is available to the agents. We address this aspect in the next subsection.

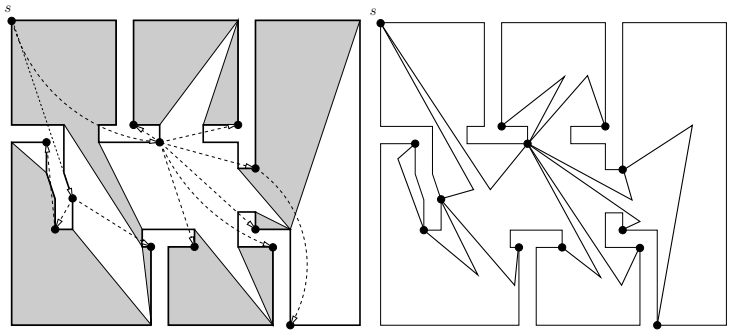


Fig. 1.4. Left figure: a vertex-induced tree and partition in a prototypical floor-plan. Right figure: the planned paths between neighboring nodes.

1.4.2 Distributed information processing

From the previous discussion we know that the following information must be available to an agent to properly navigate from node to node. If the node

is executing the `Move-to-Child Algorithm`, then it needs to know what gap to visit, i.e., what child to visit. If the node is executing the `Move-to-Parent Algorithm`, then it needs to know where the parent node is located.

This geographic information is gathered and managed by the agents via the following state transition laws and communication protocols. At this time, we make full use of the computation, communication and sensing abilities of visually-guided agents mentioned in Section 1.3.

(i) The memory content $\mathcal{M}$ of each agent is a quadruple of points in Q labeled $(p_{\text{parent}}, p_{\text{last}}, g_1, g_2)$. All four values are initialized to the initial location of the agent. During any broadcast, these values are sent over together with the agent's UID.

During run time, $\mathcal{M}$ is updated to acquire and maintain the following meaning: p_{parent} is the parent kernel point to the current agent's position, p_{last} is the last *way point*[4] visited by the agent, and (g_1, g_2) is the diagonal shared between the current cell and the parent cell, i.e., the gap toward the parent node. This is accomplished as follows:

(ii) After an agent moves from a kernel point k_i to a child kernel point k_j through a gap described by two vertices v', v'', its memory $\mathcal{M}$ is updated as follows: $p_{\text{parent}} := k_i$, $p_{\text{last}} = k_j$ and $(g_1, g_2) := (v', v'')$.

(iii) After an agent moves from a kernel point k_j to the parent kernel point k_i, its memory $\mathcal{M}$ is updated as follows: first, $p_{\text{last}} := w$, where w is the way point on the path between k_j and k_i, and second, the agent acquires updated values of $\{p_{\text{parent}}, g_1, g_2\}$ by listening to the incoming message with the highest UID.

Remark 2. At any time, at any occupied node, p_{parent} corresponding to the agent with the highest UID refers to the location of the parent of the current node. Also, (g_1, g_2) refers to the gap between the current node and the parent node. To see this, we argue as follows: Given any node k_i of $\mathcal{G}_Q(s)$ that is occupied by one or more agents, let l be the highest UID among all agents. Then, we claim that the last node visited by l is the parent of k_i. We prove this by contradiction. Let the last node visited by l be a child of k_i. To visit that child, it must have first visited k_i. Then, by the `Depth-First Navigation Algorithm`, it must have moved from k_i because of the presence of an agent with a UID greater than l. Therefore, the maximum UID at k_i must be greater than l which is a contradiction. Hence, the last node visited by l is the parent of k_i. Now according to (ii) above, the quantity p_{parent} for l refers to the parent of k_i. Also, (g_1, g_2) refers to the gap between k_i and its parent.

No common reference frame

In the description of the memory update laws, we have used a global reference frame to refer to the contents of $\mathcal{M}$. However, this assumption can be easily

[4] A way point is a mid-point of the gap between two nodes (Figure 1.4 right)

relaxed by storing the variables in $\mathcal{M}$ in a different way. For example, instead of storing the location of the parent node as a point p_{parent} relative to some global frame, the location of the parent can be stored as an integer d_{parent}, as shown in Figure 1.5. Note that such a representation does not depend on the orientation of the reference frame of an agent. The location of the gap

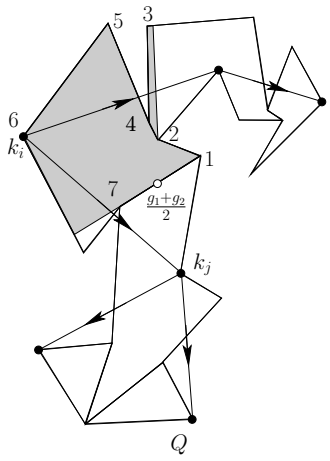


Fig. 1.5. Illustration of how the relative location of the parent of a node can be stored without the use of a common reference frame. The polygon is the environment Q. The graph with the directed edges is the vertex-induced tree in Figure 1.3. The node k_i is the parent of k_j and the point $\frac{g_1+g_2}{2}$ denoted by the white disc refers to the mid-point of the gap between k_j and k_i. The shaded region is the set of all points visible from $\frac{g_1+g_2}{2}$ on the side of the diagonal (g_1, g_2) not containing k_j. The vertices of Q in this visibility set are enumerated $(1, \cdots, 7)$ in counter-clockwise order, the vertex 1 being one of the vertices $\{g_1, g_2\}$, say g_1, and with g_2 being the last vertex in the ordering. The location of the parent can now be stored as $d_{\text{parent}} = 6$.

(g_1, g_2) can be stored in a similar fashion. The point p_{last} can be stored with respect to the local reference frame. We do not store p_{parent} and (g_1, g_2) in terms of local coordinates since these variables may be used as updates by other collocated robots. This would necessitate that the robots be aware of the relative orientations of their local coordinate frames or, equivalently, be equipped with compasses. By storing p_{parent} and (g_1, g_2) according to the scheme in Figure 1.5, the use of compasses is eliminated.

Remark 3. If the number of vertices of the environment visible from any point of the environment is bounded, then the amount of memory required to store p_{parent} and (g_1, g_2) is also bounded. Also if the diameter of the environment is bounded, then the memory required to store p_{last} is bounded. Thus, under the aforesaid assumptions, the memory $\mathcal{M}$ is constant irrespective of the complexity of the environment.

`Depth-First Navigation Algorithm`
All agents are initially located at root s
During each PROCESS action, each agent executes:

```
 1: Find maximum UID received during the LISTEN action
 2: If maximum received UID is less than its own UID
 3:    then stay at current kernel point
 4: else
 5: If there are no children of the present kernel point
 6:    then Move-to-Parent Algorithm towards p_parent via {g_1, g_2}
 7: else
 8:    Order the children in a suitable way
 9:    If p_last in memory is the parent of the present node,
          then Move-to-Child Algorithm towards the first child in the ordering
10:    If the last node visited is a child that is not the last in the ordering,
          then Move-to-Child Algorithm towards next child in the ordering
11:    If (the last node visited is a child that is the last in the ordering) AND
       (current node is not the root),
          then Move-to-Parent Algorithm towards p_parent via {g_1, g_2}
12:    If (the last node visited is a child that is the last in the ordering) AND
       (current node is the root),
          then Move-to-Child Algorithm towards the first child in the ordering
```

Table 1.1. `Depth-First Navigation Algorithm`.

1.4.3 Global exploration and deployment algorithms

At this time, we have all the elements necessary to present a global navigation algorithm that leads the agents to deploy themselves over the nodes of the vertex-induced tree. We term this algorithm `Depth-First Navigation Algorithm`, see Table 1.1.

Note that the instruction `5:` through `11:` in `Depth-First Navigation Algorithm` essentially amount to a depth-first graph search. Alternatively, it is fairly easy to design randomized graph search algorithms, where the nodes select their motion among equally likely children/parent decisions.

The following Figures 1.6 and 1.7 show the results of the simulations of the depth-first search and randomized search algorithms respectively. The nodes of the vertex-induced tree of the environment in the simulations are precisely the locations where the agents in Figure 1.6 are located at the end of the simulation. In Figure 1.7, there are more agents than the number of nodes in the vertex-induced tree. Hence, the extra agents keep exploring the graph without coming to rest.

1.4.4 Convergence and run time analysis

In this section, we provide the results on convergence of the algorithm and we also characterize the time taken for the task to be completed. Given a

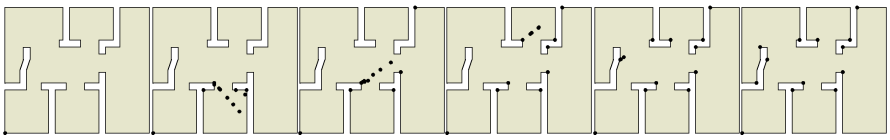


Fig. 1.6. From left to right, evolution of a network implementing depth-first search. The number of vertices of the environment is $n = 46$ and the number of agents is $N = 13 < \lfloor \frac{46}{3} \rfloor$. Each point of the environment is visible at the end of the simulation.

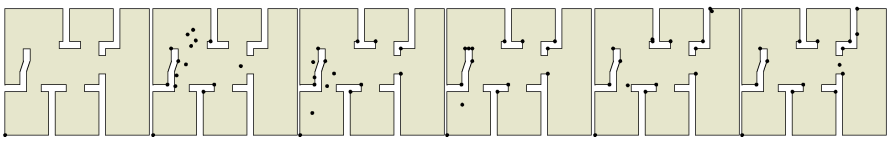


Fig. 1.7. From left to right, evolution of a network implementing randomized search. While the polygon is the same as above and therefore the vertex-induced tree still has only 13 nodes, the number of agents is 15; after each node of the tree is populated, the 2 extra agents continue to explore the vertex-induced tree.

polygon Q without holes and a vertex s, we define the following length: For each edge (k_i, k_j) of $\mathcal{G}_Q(s)$, let $d_{\text{edge}}(k_i, k_j)$ be the path length between k_i and k_j. The length of the vertex-induced tree $\mathcal{G}_Q(s)$ is defined by

$$\mathcal{L}_{\text{vit}}(\mathcal{G}_Q(s)) = \sum_{e \in \text{ edges of } \mathcal{G}_Q(s)} d_{\text{edge}}(e).$$

With these notions we can state the next result.

Theorem 1 (Convergence and Run Time Analysis). *Given a polygon without holes Q, assume that N visually-guided agents begin their motion from a vertex s of Q. Assume Q has n vertices and the vertex-induced tree $\mathcal{G}_Q(s)$ has m nodes. Assume also that there exists a bound $\lambda_{\max}$ on the* LISTEN *interval for any agent i. Then the following statements hold:*

(i) In finite time t^* there is at least one agent on $\min\{m, N\}$ nodes of $\mathcal{G}_Q(s)$.
(ii) If $N \geq \lfloor n/2 \rfloor$, then the art-gallery deployment problem is solved in finite time by the `Depth-First Navigation Algorithm`.
(iii) assuming unit speed for any agent, the time taken for task completion, t^*, obeys the following:

$$t^* = \mathcal{T}_{\text{motion}} + \mathcal{T}_{\text{nodes}},$$

where $\mathcal{T}_{\text{motion}} \leq 2\mathcal{L}_{\text{vit}}(\mathcal{G}_Q(s)) - \min\{d_{\text{edge}}(e) |\ e \in \text{ edges of } \mathcal{G}_Q(s)\}$ and $\mathcal{T}_{\text{nodes}} \leq 2(m-1)\lambda_{\max}$.

Proof. We first prove statement (i). Let us first see that at any time t, any agent is either at a node of $\mathcal{G}_Q(s)$ or on the path between two nodes. According to the `Depth-First Navigation Algorithm`, an agent always moves

according to either the `Move-to-Child Algorithm` or the `Move-to-Parent Algorithm`. By the memory update laws in Section 1.4.2, during any PROCESS interval, an agent at a node always has in its memory the location of the parent node and the gap between the current node and its parent. Therefore, an agent at a node always has enough information to compute the locations of the parent and the children and, thus, always is either at a node of $\mathcal{G}_Q(s)$ or on the path between two nodes.

Now, from step 2 of `Depth-First Navigation Algorithm`, an agent stays at a node unless there is an agent with a higher UID collocated at the same node. It also follows that once a node is occupied by an agent, it continues to be occupied by at least one agent for all future times. Therefore, the number of occupied nodes is non-decreasing. Since the number of nodes are finite, there exists a finite time τ_1 such that for all time $t \geq \tau_1$, exactly w nodes are occupied. Also, the highest UID at any occupied node is non-decreasing. Since the number of agents are finite, there exists a finite time τ_2 such that for all time $t \geq \tau_2$, the highest UID at all w occupied nodes is constant. Now, let $\tau \geq \max\{\tau_1, \tau_2\}$. Now, if $w \geq N$, then we are done. If $w < N$, then at any time $t \geq \tau$, there are $N - w$ agents that either belong to w occupied nodes or belong to the paths between two nodes of $\mathcal{G}_Q(s)$. Since the UID at any occupied node is constant, this implies that the $N - w$ agents are the ones with the lowest UIDs. From the `Depth-First Navigation Algorithm`, each of the $N - w$ agents perform a depth-first search on $\mathcal{G}_Q(s)$ spending at most $\lambda_{\max}$ time at any node. If $w \geq \min\{m, N\}$, then we are done. If $w < N \leq m$, there is at least one node that is unoccupied. Therefore, each of the $N - w$ agents will reach an unoccupied node of $\mathcal{G}_Q(s)$ in finite time. Thus, the number of occupied nodes increases which is a contradiction. Therefore, $w \geq N$. If on the other hand, $w < m \leq N$, then again there is at least one node that is unoccupied. By a similar argument as before, it follows that the number of unoccupied nodes increases.

Statement (ii) follows from statement (i) and from Proposition 1 (ii) which states that $m \leq \lfloor \frac{n}{2} \rfloor$.

To prove statement (iii), let us assume that k_l be the last node to be occupied at time t^*. Clearly, k_l has to be a leaf. Let the agent first occupying k_l be j. To travel from the root to any leaf via a depth-first search, an agent traverses each edge at most twice except for the edge incident to the leaf, which has to be traversed only once. Thus, agent j travels at most $\left(\sum_{e \in \text{ edges of } \mathcal{G}_Q(s)} 2d_{\text{edge}}(e)\right) - \min\{d_{\text{edge}}(e) |\ e \in \text{ edges of } \mathcal{G}_Q(s)\}$ distance. Since the agent is assumed to move with unit speed, the time taken to travel this distance, $\mathcal{T}_{\text{motion}}$, is $2\mathcal{L}_{\text{vit}}(\mathcal{G}_Q(s)) - \min\{d_{\text{edge}}(e) |\ e \in \text{ edges of } \mathcal{G}_Q(s)\}$. Also, while traveling from the root to k_l, agent j stops at each of the remaining $m - 1$ nodes at most twice. At each node, agent j spends at most $\lambda_{\max}$ time. Thus, the time spent at the nodes, $\mathcal{T}_{\text{node}}$, is $2(m-1)\lambda_{\max}$. The total time, $t*$ is equal to $\mathcal{T}_{\text{motion}} + \mathcal{T}_{\text{node}}$ and the result follows.

1.5 Conclusions

In this paper, we pose a distributed version of the classic Art Gallery Problem for mobile robotic agents. Under assumptions of line-of-sight communication and sensing on the agents, we design a provably correct `Depth-First Navigation Algorithm` that solves the problem given that the agents are initially collocated at a vertex of the environment. The algorithm is robust to arbitrary but bounded communication delays. Under the assumptions of bounded environment diameter and bounded number of vertices visible from any point in the environment, the memory required by the agents is constant irrespective of the environment complexity. An early version of this algorithm appeared in [26].

1.6 Acknowledgment

This material is based upon work supported in part by AFOSR through Award F49620-02-1-0325, by NSF through Award CMS-0626457, and by NSF through CAREER Award ECS-0546871. The authors thank Prof. Seth Hutchinson for his kind support.

References

1. V. Chvátal, "A combinatorial theorem in plane geometry," *Journal of Combinatorial Theory. Series B*, vol. 18, pp. 39–41, 1975.
2. R. Honsberger, *Mathematical Gems II: The Dolciani Mathematical Expositions*. Mathematical Association of America, 1976.
3. S. Fisk, "A short proof of Chvátal's watchman theorem," *Journal of Combinatorial Theory. Series B*, vol. 24, p. 374, 1978.
4. I. Rekleitis and V. Dujmović, "Efficient topological exploration," in *IEEE Int. Conf. on Robotics and Automation*, (Detroit, MI), pp. 676–681, May 1999.
5. G. Dudek, M. Jenkin, E. Milios, and D. Wilkes, "Topological exploration with multiple robots," in *International Symposium on Robotics and Applications*, (Anchorage, Alaska), May 1998.
6. I. M. Rekleitis, G. Dudek, and E. E. Milios, "Multi-robot exploration of an unknown environment, efficiently reducing the odometry error," in *International Joint Conference in Artifical Intelligence*, vol. 2, (Nagoya, Japan), pp. 1340–1346, Aug. 1997.
7. A. Howard, M. J. Matarić, and G. S. Sukhatme, "An incremental self-deployment algorithm for mobile sensor networks," *Autonomous Robots*, vol. 13, no. 2, pp. 113–126, 2002.
8. J. Cortés, S. Martínez, T. Karatas, and F. Bullo, "Coverage control for mobile sensing networks," *IEEE Transactions on Robotics and Automation*, vol. 20, no. 2, pp. 243–255, 2004.
9. J. Grace and J. Baillieul, "Stochastic algorithms for autonomous robotic surveillance," in *IEEE Conf. on Decision and Control and European Control Conference*, (Seville, Spain), pp. 2200–2205, Dec. 2005.

10. R. Simmons, D. Apfelbaum, D. Fox, R. Goldman, K. Haigh, D. Musliner, M. Pelican, and S. Thrun, "Coordinated deployment of multiple heterogenous robots," in *IEEE/RSJ Int. Conf. on Intelligent Robots & Systems*, (Takamatsu, Japan), pp. 2254–2260, 2000.
11. T. C. Shermer, "Recent results in art galleries," *IEEE Proceedings*, vol. 80, no. 9, pp. 1384–1399, 1992.
12. 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.
13. J. E. Goodman and J. O'Rourke, eds., *Handbook of Discrete and Computational Geometry*. Boca Raton, FL: CRC Press, 1997.
14. H. Choset, "Coverage for robotics - a survey of recent results," *Annals of Mathematics and Artificial Intelligence*, vol. 31, pp. 113–126, 2001.
15. H. H. González-Baños, A. Efrat, J.-C. Latombe, E. Mao, and T. M. Murali, "Planning robot motion strategies for efficient model construction," in *International Symposium on Robotics Research*, (Snowbird, UT), Oct. 1999.
16. J. S. B. Mitchell, "Shortest paths and networks," in *Handbook of Discrete and Computational Geometry* (J. E. Goodman and J. O'Rourke, eds.), ch. 24, pp. 445–466, Boca Raton, FL: CRC Press, 1997.
17. P. K. Agarwal and M. Sharir, "Efficient algorithms for geometric optimization," *ACM Computing Surveys*, vol. 30, no. 4, pp. 412–458, 1998.
18. A. Okabe, B. Boots, K. Sugihara, and S. N. Chiu, *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*. Wiley Series in Probability and Statistics, New York: John Wiley, 2 ed., 2000.
19. Z. Drezner, ed., *Facility Location: A Survey of Applications and Methods*. Springer Series in Operations Research, New York: Springer Verlag, 1995.
20. N. A. Lynch, *Distributed Algorithms*. San Mateo, CA: Morgan Kaufmann Publishers, 1997.
21. D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods*. Belmont, MA: Athena Scientific, 1997.
22. S. Martínez, F. Bullo, J. Cortés, and E. Frazzoli, "On synchronous robotic networks – Part I: Models, tasks, and complexity . Part II: Time complexity of rendezvous and deployment algorithms," *IEEE Transactions on Automatic Control*, vol. 53, no. 1, 2008. To appear.
23. L. Guilamo, B. Tovar, and S. M. LaValle, "Pursuit-evasion in an unkown environment using gap navigation trees," in *IEEE/RSJ Int. Conf. on Intelligent Robots & Systems*, (Sendai, Japan), pp. 3456–3462, Sept. 2004.
24. V. Isler, S. Kannan, and S. Khanna, "Randomized pursuit-evasion in a polygonal environment," *IEEE Transactions on Robotics*, vol. 5, no. 21, pp. 864–875, 2005.
25. N. Moshtagh, A. Jadbabaie, and K. Daniilidis, "Vison-based distributed coordination of multiagent systems," in *Robotics: Science and Systems* (S. Thrun, G. Sukhatme, S. Schaal, and O. Brock, eds.), pp. 41–48, Cambridge, MA: MIT Press, 2005.
26. 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.