Journal of Autonomous Agents and Multi-Agent Systems manuscript No.
(will be inserted by the editor)

Capturing an Area-Covering Robot

Roi Yehoshua · Noa Agmon

Received: date / Accepted: date

Abstract Area coverage is a fundamental task in robotics, where one or more robots are required to visit all points in a target area at least once. In many real-world scenarios, the need arises for protecting one's territory from being covered by a robot, e.g., when we need to defend a building from being surveyed by an adversarial force. Therefore, this paper discusses the problem of defending a given area from being covered by a robot. In this problem, the defender needs to choose the locations of k stationary guards in the target area, each one having some probability of capturing the robot, in a way that maximizes the probability of stopping the covering robot. We consider two types of covering robots: one that has an a-priori map of the environment, including the locations of the guards; and the other has no prior knowledge of the environment, and thus has to use real-time sensor measurements in order to detect the guards and plan its path according to their discovered locations. We show that in both cases the defender can exploit the target area's topology, and specifically the vulnerability points in the area (i.e., places that must be visited by the robot more than once), in order to increase its chances of capturing the covering robot. We also show that although in general finding an optimal strategy for a defender with zero-knowledge on the robot's coverage strategy is $\mathcal{NP}$-Hard, for certain values of k an optimal strategy can be found in polynomial time. For other cases we suggest heuristics that can signif-

The research was funded in part by ISF grant 1337/15.

Roi Yehoshua
Department of Computer Science
Bar-Ilan University, Israel
E-mail: yehoshr1@cs.biu.ac.il

Noa Agmon
Department of Computer Science
Bar-Ilan University, Israel
E-mail: agmon@cs.biu.ac.il

icantly outperform the random baseline strategy. We provide both theoretical and empirical evaluation of our suggested algorithms.

Keywords Mobile robot coverage · area protection · adversarial coverage · motion and path planning · robotics in hazardous fields

1 Introduction

Coverage path planning is one of the fundamental problems in robotics. The goal of coverage path planning is to find a sequence of world locations which allows the robot to visit every part of the target area while optimizing some criteria, usually minimizing travel cost, while avoiding obstacles. This problem has many real-world applications, from automatic floor cleaning [7] and grass-mowing [1], to field demining [21] and surveillance by unmanned aerial vehicles (UAVs) [13, 20].

In a recently introduced version of the problem, *adversarial coverage* [24–28], the robot has to cover the given terrain without being stopped by an adversary. Each point in the area is associated with a probability of the robot being stopped at that point. The objective of the robot is to cover the entire target area (including the threat points) as quickly as possible while minimizing the probability that it will be stopped before completing the coverage. This problem is a generalization of the original problem of coverage in neutral environments (without adversarial presence), where risks do not exist, thus are not accounted for, and the only goal is to minimize coverage time [3, 10, 15]. Adversarial coverage can be applied in a wide range of fields, from performing coverage missions in hazardous environments such as nuclear power plants or the surface of Mars, to surveillance of enemy forces in the battle field and field demining.

In previous studies of the adversarial coverage problem, a simplistic adversarial model, in which the locations of the threat points in the environment are randomly chosen, was assumed. In this paper we build a more sophisticated model of the adversary, in which it can choose the best locations of the threat points, such that the probability of stopping the covering robot is maximized. Hence, we are now looking at the problem from the perspective of the adversary. In the eyes of the adversary, the threat points are considered as guards that protect its territory from being covered by an unwanted intruder. We will refer here to the adversary as the defender, since the covering robot is now the one that we are trying to defend against. Hence, this work addresses the *coverage defending problem*, the task of placing a group of guards in the environment as to maximize the probability of capturing a covering robot. Examples of practical applications of this problem include a strategic placement of surveillance cameras in a building as to protect it from being explored by an unwanted intruder, and deployment of anti-drones weapons that protect a territory from being surveyed by hostile UAVs.

We examine the impact of the defender's knowledge of the robot's coverage path on its choice of the guards' locations, and provide solutions for

defenders having no knowledge, partial knowledge and full knowledge of the robot's coverage strategy. We show that for a full-knowledge defender there is a simple algorithm that provides the optimal strategy for placing the guards, whereas finding an optimal strategy for a zero-knowledge defender is $\mathcal{NP}$-Hard. Nevertheless, we propose an algorithm for a zero-knowledge defender that outperforms the baseline random strategy, and can provide the optimal solution under certain conditions. We also discuss some cases in which the defender has partial knowledge of the robot's coverage path, e.g., when it knows where the path begins.

For a zero-knowledge defender, we propose defending strategies against two types of covering robots: an offline covering robot that possesses a map of the environment, including the specific locations of the guards; and an online covering robot that has no a-priori map of the environment, and thus has to use real-time sensor measurements in order to detect the guards. We show that in the latter case the defender can take advantage of the robot's limited sensing to affect the order in which it visits the various parts of the environment, thus making it visit places that are protected by guards more frequently.

Finally, we evaluate the defender's strategies in an extensive set of experiments, testing it against various types of covering robots (with different sensing capabilities) and in various types of environments. The results show that the defender's suggested strategy performs significantly better than randomly placing the guards in the environment. They also show that the defender's strategy is robust against robots with different sensing capabilities, i.e., adding better sensing capabilities to the robot has little influence on the ability of the defender to stop it.

The paper is organized as follows. Section 2 presents background and related work. Section 3 defines the coverage defending problem. Sections 4 and 5 present strategies for a full-knowledge defender and a zero-knowledge defender, respectively, and analyze them theoretically. Section 6 presents experimental results evaluating these strategies. Section 7 concludes with a summary and brief overview of future work.

2 Related Work

The problem of area coverage has been discussed extensively in the robotic literature. A survey of various coverage algorithms is provided in Choset [5], and a more recent one is provided by Galceran and Carreras [11]. These surveys distinguish between offline coverage algorithms, in which the robot has a priori knowledge of the entire environment, and online or sensor-based coverage algorithms, in which no such information is required. They further distinguish between approximate cellular decomposition, where the free space is approximately covered by a grid of equally-shaped cells, and exact decomposition, where the free space is exactly partitioned. Here we assume approximate cellular decomposition, and we treat both offline and online coverage.

A related problem to coverage is the patrolling problem, where a team of robots is required to visit a target area repeatedly in order to monitor some change in state of that area, typically to detect an intrusion to the area (see [23] for a survey on multi-robot patrolling algorithms). In contrast, here the guards are stationary and they may catch the covering robot with some probability less than or equal to 1. A new problem addressing patrolling security game where a defender is supported by a spatially uncertain alarm system was presented in [2]. In their work, a single defender that moves simultaneously with the attacker uses the detection signals from the alarm system to make a decision on its next move. Conversely, in our case the stationary defenders (guards) are used for both detecting and capturing the attacker.

In the art gallery problem, the minimum number of guards, who together can observe the whole gallery, needs to be determined [8, 18]. This problem is known to be $\mathcal{NP}$-Hard [18]. Conversely, here the defender does not necessarily possess enough guards to cover the entire area. In addition, in the art gallery problem it is assumed that each guard has a probability one of detecting an intruder within its sensing range, while in our case each guard has some probability $0 < p \leq 1$ of capturing the intruder.

Pursuit-evasion is another family of problems in which one one or more pursuers try to capture one or more evaders who, in turn, try to avoid capture [4, 6]. Typically, the objective in this problem is to decide how many pursuers are needed, and how they should move on a given graph, in order to capture all the evaders with either a minimum sum of their travel distances or minimum task-completion time. Conversely, in our problem the evader (the covering robot) needs to visit all the vertices in the graph in the shortest possible time, while the pursuers (the guards) are not moving.

Pita et al. [22] present a game-theoretic approach for choosing random checkpoints on the roadways entering the airport to combat various potential adversaries. The main focus of their study was to show how the security schedules of the police can be randomized in order to avoid the vulnerability that comes with predictability. In contrast to their work, here the covering robot may encounter more than one guard (checkpoint) during its coverage mission, and also it must visit all the places in the target area rather than penetrate into a specific location.

The offline single-robot adversarial coverage problem was formally defined in [28]. There we suggested two heuristic algorithms for solving the problem: STAC, a spanning-tree based coverage algorithm, and GAC, which follows a greedy approach. We have shown that while STAC tends to achieve higher expected coverage, GAC produces shorter coverage paths with lower accumulated risk. The online single-robot version of the problem was presented in [25]. Both of these studies assumed a simplistic adversarial model, in which the threats are randomly scattered across the environment.

In [24] we have built a more sophisticated adversarial model, in which the adversary (referred to as the defender in this paper) can choose the best locations of the threat points (guards), such that the probability of stopping the covering robot is maximized. By analyzing the graph representing the

target area, we have shown how the vulnerability points in the area (i.e., places that must be visited multiple times by the robot) can be exploited by the adversary in order to increase its chances of catching the covering robot. In that work we assumed that all the guards have the same probability of capturing the robot, and that the covering robot has full knowledge of the environment prior to its movement (i.e., offline coverage).

This paper extends the work initially presented in [24], while making the following new contributions:

1. We extend the defender model to cases where its guards have non-uniform probability of capturing the covering robot, thus taking into account the ability of each guard to detect the robot.
2. We analyze different models of the covering robot, i.e., offline coverage (in which the robot is given the map of the environment in advance) vs. online coverage (in which the covering robot has no knowledge of the environment prior to the coverage). We show that in the online case, the defender can take advantage of the robot's limited knowledge on the environment in order to increase its probability of capturing the robot.
3. We evaluate the extended models in a new variety of simulated environments and settings.

3 Coverage Defending Problem Formulation

We represent the target area by an undirected connected graph $G = (V, E)$ with $v_i \in V$ vertices, $e_{i,j} \in E$ edges, and $|V| = n$. G corresponds to the topological map for the coverage mission and is assumed to be known a priori to the defender.

In the robotic coverage literature, a typical representation of the environment is that of a grid map (e.g., [10, 19, 29]). In this case, the graph $G = (V, E)$ is the graph induced by the grid cells, i.e., each grid cell that is not occupied by an obstacle is represented by a vertex in V, and vertices that represent adjacent free cells in the grid are connected by an edge in E. When the environment is represented as a grid, we assume that the robot can move only in the four basic directions (up/down, left/right), but not diagonally.

To make our theoretical analysis as general as possible, we define the defender strategy for any graph representation of the environment, but use grid maps in the experiments.

3.1 Covering Robot Model

In the offline case, we assume that the robot has the map describing the target area (i.e., the graph G). We also assume that the robot is able to localize itself to a specific node in the graph using an appropriate localization system.

Some of the coverage algorithms known in the literature assume that the robot must return to its starting point when the coverage ends, facilitating

its collection and storage (e.g., Spiral-STC [9]), while others do not hold this assumption (e.g., the wavefront algorithm [29]). Here, the success of the robot will be determined solely by its ability to complete the coverage, thus we do not deal with the question of what happens to the robot after it completes its coverage task (e.g, if it needs to return to its initial location or find a safe escape route).

The goal of the robot is to find a path in G that visits every vertex of V at least once and has the minimum total risk, i.e., encounters cells that are protected by guards the least number of times. Finding the optimal coverage plan for the robot is an $\mathcal{NP}$-Hard problem [28].

In the online case, we define an *observation function* $\mathcal{O}(v)$ that returns the set of vertices in the graph that can be observed by the robot from a given vertex v. Even though our approach can be applied to any observation function, for simplicity we assume the robot is equipped with an omnidirectional sensor with limited range r (e.g., a panoramic camera). If the robot's sensors provide less than a 360° field of view, we assume that after arriving at a new location, the robot turns around in order to take a 360° view of its surroundings.

Thus, when the environment is represented as a regular grid of equal square cells, the observation function is defined as follows: the robot in a cell c observes a cell c' if the line segment connecting the centers of c and c' is in the circle of radius r, centered in c, and does not cross any occupied cell. Figure 1 shows an example for $\mathcal{O}(c)$, when the robot is located at cell $c = (5, 5)$ and has a sensor with radius $r = 3$.

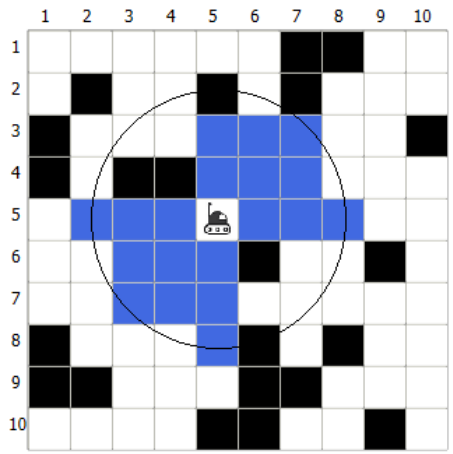

Fig. 1: The set of cells that can be detected by the robot's sensor from cell (5, 5) within the range $r = 3$. Obstacles are represented by black cells, and visible cells are colored blue.

3.2 Defender Model

The objective of the defender is to protect its territory from being covered by the robot. The defender can place $k < n$ stationary guards at different locations of the area. Each guard i has some probability p_i $(0 < p_i \leq 1)$ of intercepting the robot, when the robot enters into its location. This probability depends on the ability of the guard to detect and/or catch the robot, depending on the type of guards that will be employed. For example, if the area is protected by security cameras, then p_i represents the ability of each camera to detect the robot, depending on several factors such as the quality of the camera, illumination conditions in the area, etc. On the other hand, if the defender uses traps in the floor, then p_i represents the ability of each trap to physically stop the robot, depending on factors such as the size of the trap, its visibility, the sensing and maneuvering capabilities of the robot, etc.

Figure 2 shows an example world map represented as a grid of size 20×20. Obstacles are represented by black cells, unguarded cells are colored white, and cells that are protected by guards are represented by 5 different shades of the same color. [1] Darker shades represent higher values of p_i, i.e., locations that are protected by guards with higher probability of capturing the robot.

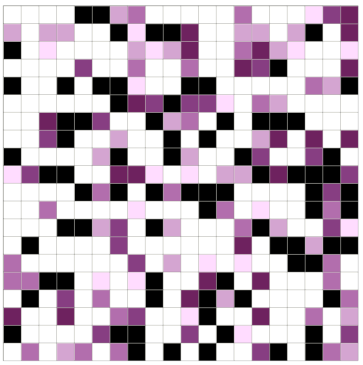

Fig. 2: An example map of the world. Darker shades represent locations that contain guards with higher probabilities of capturing the robot.

The goal of the defender is to assign the k guards to k locations in the environment such that the probability of capturing the covering robot is maximized. Let us formally define this objective function. First, we denote the coverage path followed by the robot by $A = (w_1, w_2, ..., w_m)$, where $w_1, ..., w_m$ are vertices of the graph G. Note that $m \geq n$, i.e., the number of vertices in the coverage path might be greater than the number of vertices in G, since the robot is allowed to repeat its steps. Let us denote the probability that the robot is captured at vertex v_i by p_i: $p_i = 0$ if no guard is assigned to v_i,

[1] appearing as magenta in the paper's online version and gray in its printed version.

otherwise it is equal to the assigned guard's probability of capturing the robot. We also denote the number of times the robot visits the vertex v_i along its coverage path by t_i ($t_i \geq 1$).

Now, we define the event S_A as the event that the robot is able to complete its coverage path A, without being captured by any of the guards. Assuming that the events of the robot being stopped at any vertex of A are independent, the probability of S_A is the product of all the probabilities that the robot is not stopped at any of the vertices in A, i.e.,

$$P(S_A) = \prod_{i=1}^{n} (1 - p_i)^{t_i} \tag{1}$$

Thus, the objective of the defender is to find an assignment for the given k guards that will minimize the probability $P(S_A)$.

We distinguish between defenders having no knowledge and full knowledge of the robot's coverage path. A full-knowledge defender knows the exact coverage path A of the robot. Thus, its objective is to choose k vertices in V at which to place guards, such that $P(S_A)$ is minimized for a specific coverage path A. On the other hand, a zero-knowledge defender has no information on the coverage path of the robot. Thus, it can only assume that the covering robot follows an optimal covering strategy, i.e., that it tries to visit every point in the target area the least possible number of times, especially the ones that are protected by guards. Hence, the objective of a zero-knowledge defender is to choose k vertices in V at which to place guards, such that $P(S_A)$ is minimized for an optimal coverage path A. Section 4 presents a strategy for a full-knowledge defender, while Section 5 deals with a zero-knowledge defender.

Table 1 summarizes the different types of defenders and robots discussed in the paper, and provides a reference to the algorithms that deal with each case.

Defender	**Robot**	**Description**	**Algorithm**
Full-knowledge	Any type	Defender knows the robot's path, thus robot's coverage strategy is irrelevant.	1
Zero-knowledge	Offline	Defender does not know the robot's path. The robot has the area's map.	4
Partial-knowledge	Offline	Defender only knows where the robot's path begins. The robot has the area's map.	4
Partial-knowledge	Online	Defender does not know the robot's path, but knows its sensing capabilities. The robot has no map of the area.	5

Table 1: Summary of defender and covering robot types

4 Full-Knowledge Defender

For a full-knowledge defender, there is a simple algorithm that generates an optimal strategy with maximum probability of stopping the robot (see Algorithm 1). The idea is to place the given k guards at vertices that are most frequently visited along the known coverage path A, sorted by their probability of capturing the robot.

Algorithm 1 FullKnowledgeDefender(G, A, S)

input: $G = (V, E)$ – the graph representing the environment, A – the coverage path, S – the set of guards

1: Compute for every vertex in V the number of times it is visited by A
2: Sort the vertices in V in a decreasing order of their visits number
3: Denote by $v_1, ..., v_n$ the vertices after the sort
4: Sort the guards in S in a decreasing order of their probability of stopping the robot
5: Denote by $g_1, ..., g_k$ the guards after the sort
6: **for** $i \leftarrow 1$ **to** k **do**
7: Assign guard g_i to vertex v_i

Clearly, the allocation of guards as defined by Algorithm 1 maximizes the probabilities p_i with the highest exponents in (1), thus minimizing the probability of the robot completing the coverage $P(S_A)$. The runtime complexity of the algorithm is $O(n\log n)$, where n is the number of vertices in G.

5 Zero-Knowledge Defender

For a zero-knowledge defender, that has no information on the robot's coverage path, the problem of finding an optimal strategy that maximizes its chances of capturing the robot becomes $\mathcal{NP}$-Hard, as proven by the next theorem.

Definition 1 Zero-Knowledge Defender Coverage Problem (ZKDCP): Given a graph representation of the environment $G = (V, E)$, choose k vertices of V at which to position guards, such that the probability of stopping any robot covering the environment is maximized.

Theorem 1 *The ZKDCP problem is $\mathcal{NP}$-Hard.*

Proof. To prove the $\mathcal{NP}$-hardness of the problem, we will use a reduction from the Hamiltonian path problem, which is known to be $\mathcal{NP}$-complete [12].

Given an instance of the Hamiltonian path problem on a graph $G = (V, E)$, we construct an instance of the zero-knowledge defender coverage problem on the same graph G with $k = 1$. We will prove that there exists a Hamiltonian path in G, if and only if the optimal strategy for the defender is to place its single guard at a random vertex of G.

First direction – if there exists a Hamiltonian path in G, then there is a coverage path of G that visits each vertex exactly once (which is also the

optimal one). In this case, the optimal strategy for the defender is to place its single guard randomly at one of the vertices, since it has no benefit from placing the guard at any specific vertex (non-optimal coverage paths may visit any other vertex more frequently than that vertex).

Second direction – if G is non-Hamiltonian, then there is at least one vertex of G that must be visited multiple times by any coverage path of G. In this case, the optimal strategy for the defender is to place its single guard at the vertex that must be visited the greatest number of times, which is clearly different from just choosing a random vertex of G.

Therefore, we can find if there exists a Hamiltonian path in a given graph G, by checking if the optimal strategy for a zero-knowledge defender is to place a single guard randomly at one of the vertices of G or not. Thus, the zero-knowledge defender coverage problem is $\mathcal{NP}$-hard. □

Although the general problem is $\mathcal{NP}$-hard, we will show in Section 5.3.1 that for certain values of k and in certain types of environments, an optimal defender strategy can be found in polynomial time.

5.1 Outline of the defender's strategy

We now describe the general strategy for a zero-knowledge defender. More detailed algorithms are presented in Section 5.3 for a defender acting against an offline covering robot, and in Section 5.4 for a defender acting against an online covering robot.

In general, the defender's strategy consists of the following main steps:

1. Place guards at vertices of the graph that must be visited by the covering robot more than once, while giving precedence to vertices that must be visited more frequently.
2. Place guards at groups of vertices, where at least one of the vertices must be visited by the covering robot more than once, while giving precedence to groups that must be visited more frequently.
3. If there are any more guards left to place after taking the first two steps, use heuristics to choose additional locations in which to place the remaining guards.

5.2 Analysis of the representative graph

The defender's strategy is based on exploiting certain properties of the graph representing the environment. In this section we describe these properties and analyze them theoretically. First, we use the following definitions and theorems from graph theory [14].

Definition 2 An **articulation point** (cut vertex) in a connected graph G is a vertex whose removal would break the graph into two or more connected components.

Definition 3 A connected graph is **biconnected** if it has no articulation points.

Definition 4 A **block** (a biconnected component) is a maximal biconnected subgraph, i.e. a subgraph with as many edges as possible and no articulation points.

Definition 5 A **vertex cut** in a connected graph G is a set of vertices whose removal renders G disconnected.

Definition 6 A **block-cut tree** of a graph G is a tree in which each node represents either a block or an articulation point of G. A node representing an articulation point is connected to all nodes representing blocks that contain that point.

The block decomposition theorem [14] states that there is a unique block-cut tree for each graph. Figure 3 shows an example for a block-cut tree of a graph. The blocks are $b_1 = \langle 1, 2\rangle, b_2 = \langle 2, 3, 4\rangle, b_3 = \langle 2, 5, 6, 7\rangle, b_4 = \langle 7, 8, 9, 10, 11\rangle, b_5 = \langle 8, 12, 13, 14, 15\rangle, b_6 = \langle 10, 16\rangle, b_7 = \langle 10, 17, 18\rangle$, and the articulation points are $c_1 = 2, c_2 = 7, c_3 = 8, c_4 = 10$.

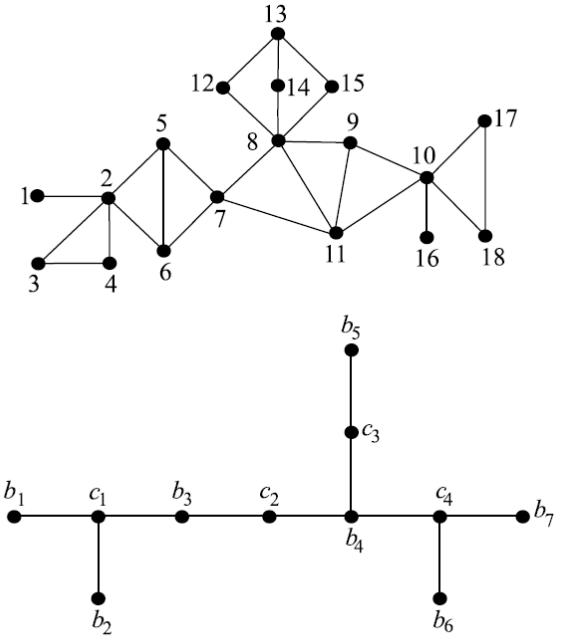

Fig. 3: A graph (upper figure) and its block-cut tree (lower figure).

We now extend the definition of an articulation point to a k-connected articulation point.

Definition 7 Connectivity of a vertex v is the number of connected components the graph would split into if v is removed from the graph.

Definition 8 A **k-connected articulation point** is an articulation point whose connectivity is k.

For example, the articulation point c_1 in Figure 3 is 3-connected, since removing it from the graph would split it into three connected components: $\langle 1 \rangle, \langle 3, 4 \rangle, \langle 5, 6..., 18 \rangle$, while the articulation point c_2 is 2-connected, since removing it from the graph would split it into only two connected components: $\langle 1, ..., 6 \rangle, \langle 8, ..., 18 \rangle$. By definition, each articulation point must be at least 2-connected.

In Section 5.3.1 we show the relationship between the connectivity of the articulation points and the number of times the covering robots must visit these points.

5.2.1 Finding the articulation points and their connectivity

Algorithm 2 describes how to find all the articulation points in the graph and their connectivity. It is an extension of the classical linear-time algorithm for computing biconnected components in a connected undirected graph by Hopcroft and Tarjan [17], which is based on Depth-First Search (DFS).

Before introducing the algorithm, we remind the reader that every edge (u, v) traversed during a DFS execution can be classified into one of several types, depending on whether we have visited v before in the DFS, and if so, the relationship between u and v. In particular, we will be interested in the following two types of edges:

Definition 9 The edge (u, v) is a **tree edge** if v is visited for the first time as we traverse the edge (u, v) during a DFS execution.

Definition 10 The edge (u, v) is a **back edge** if v is an ancestor of u when we traverse the edge (u, v) during a DFS execution.

Algorithm 2 keeps the following fields for each vertex v in the graph:

- $num[v]$: the DFS visit number of v (i.e., the first time that v is visited during the DFS execution).
- $low[v]$: the smallest value of $num[x]$, where x is a vertex of the graph that can be reached from v following a sequence of zero or more tree edges followed by at most one back edge.
- $level[v]$: the DFS tree level of v.
- $children[v]$: the number of children of v in the DFS tree.
- $connectivity[v]$: the connectivity of v.
- $blocks[v]$: the list of blocks that are rooted at v.

Figure 4 demonstrates how these fields are computed on a sample graph. For instance, $low[b] = 1$, since the lowest numbered vertex that can be reached from d is a (through the tree edge (b, d) and then the back edge (d, a)). On the other hand, $low[c] = 5$, since the lowest numbered vertex that can be reached from c is c itself.

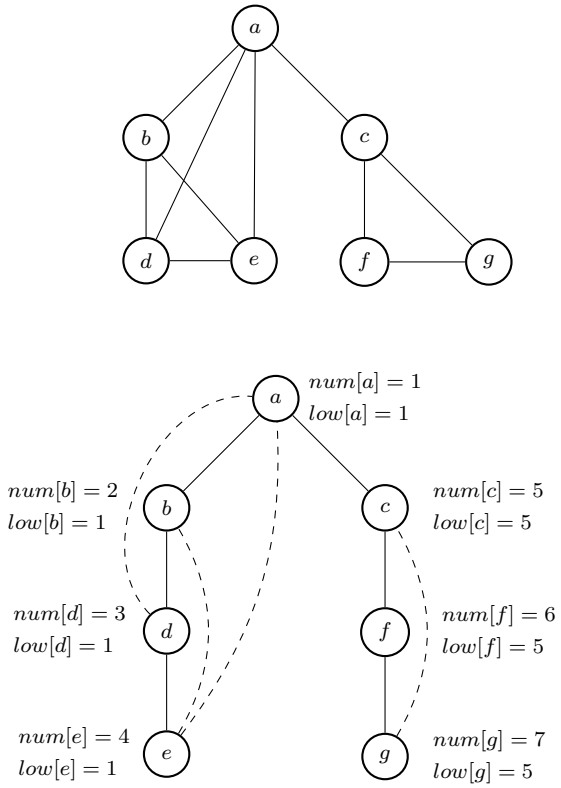

Fig. 4: Lower figure shows the DFS tree starting at a with num/low values for the input graph in the upper figure.

The algorithm for finding articulation points is based on the following two key facts:

1. The root of the DFS tree is an articulation point if and only if it has more than one child (since every path from the left subtree of the root to its right subtree must go through the root).
2. Any other vertex v is an articulation point if and only if v has some child w such that $low[w] \geq num[v]$, i.e., there is a child w of v that cannot reach a vertex visited before v. The number of such children determines the connectivity of v (i.e., the number of connected components the graph would break into if v is removed from the graph).

For example, in Figure 4, the root a is an articulation point because it has two children, and $connectivity[a] = 2$. Vertex c is also an articulation point because it has a child f with $low[f] \geq num[c]$, and its connectivity is 2. The other vertices in this graph are not articulation points.

In order to find the blocks of the graph, the edges of G are placed on a stack as they are traversed. When an articulation point is found, the corresponding edges of its block are all on top of the stack. Procedure CreateBlock is then used to pop the edges of this block from the stack. For that purpose, it uses the standard stack operations $\mathsf{Pop}(S)$ which removes the top element from the stack S, and $\mathsf{Top}(S)$ that returns the top element of S.

We have extended the classical algorithm with the following additions:

1. We compute the connectivity of each articulation point (lines 14, 20 and 22 in the algorithm).
2. We maintain for each articulation point a list of all the blocks that are attached to it in the block-cut tree (lines 16 and 24). This information will be needed in the guards assignment algorithm (Section 5.3).

Algorithm 2 FindArticulationPoints(v)

input: v – the current vertex
Global variables: S – the stack of visited edges, ArticulationPointList – the list of articulation points, *counter* – the current DFS visit number.
Global initialization: $counter = 0$, $level[s] = 0$, where s is the starting vertex.

```
1:  counter ← counter + 1
2:  num[v] ← counter
3:  low[v] ← num[v]
4:  for each neighbor w of v do
5:      if w was not visited yet then
6:          level[w] ← level[v] + 1
7:          Push(S, (v, w))
8:          FindArticulationPoints(w)          ▷ recursively perform DFS at children nodes
9:          low[v] ← min(low[v], low[w])
10:         if num[v] = 1 then                                  ▷ special case for root
11:             if children[v] ≥ 2 then
12:                 if v ∉ ArticulationPointList then
13:                     Add v to ArticulationPointList
14:                     connectivity[v] ← children[v]
15:             B ← CreateBlock(S, v, w)
16:             Add B to blocks[v]
17:         else if low[w] ≥ num[v] then          ▷ v is an articulation point separating w
18:             if v ∉ ArticulationPointList then
19:                 Add v to ArticulationPointList
20:                 connectivity[v] ← 2
21:             else
22:                 connectivity[v] ← connectivity[v] + 1
23:             B ← CreateBlock(S, v, w)
24:             Add B to blocks[v]
25:     else if level[w] < level[v] − 1 then                    ▷ (v, w) is a back edge
26:         low[v] ← min(low[v], num[w])
27:         Push(S, (v, w))
```

```
1: procedure CreateBlock(S, v, w)
   input: S – the stack of visited edges, v and w are vertices where w is a child of v
2:     Create a new block B
3:     while Top(S) ≠ (v, w) do                    ▷ Retrieve all edges in the component
4:         (u1, u2) ← Pop(S)
5:         Add (u1, u2) to B
6:     Add Pop(S) to B                                          ▷ Add (v, w) to B
7:     return B
```

Algorithm 2 can be started with any vertex v of the graph, since by the block decomposition theorem there is a unique block-cut tree for the graph G. Choosing a different vertex as the root of the block-cut tree will only cause the tree to rotate, but will not modify its structure.

The correctness of Algorithm 2 with respect to finding the articulation points and blocks of the graph follows directly from the correctness of the original algorithm by Hopcroft and Tarjan [17]. We also prove that the connectivity of each articulation point is computed correctly by the algorithm.

Theorem 2 *(correctness) Algorithm 2 computes correctly the connectivity of each articulation point in the graph.*

Proof. If the root of the block-cut tree has more than one child then it is an articulation point, and the number of connected components its removal splits the graph into is equal to the number of its child nodes (line 14 in the algorithm). Any other vertex v is as an articulation point if and only if v has some child w such that $low[w] \geq num[v]$, i.e., there is a child w of v that cannot reach a vertex visited before v (line 17). The first time this condition is true for v, it is added to the articulation points list and its connectivity is set to 2 (line 20), since removing v from the graph would split it into at least two connected components: one that contains the vertices visited before v and another that contains w and its descendants in the DFS tree. In each subsequent time this condition becomes true for v, it means that we have found another child w' of v, such that v separates w' from the vertices visited before v, thus removing v from the graph would create another connected component in the graph (the subtree rooted in w'). Hence, the connectivity of v is increased by 1 (line 22). □

The runtime complexity of the original algorithm for finding the articulation points is $O(|V| + |E|)$. Since we need to add only $O(1)$ operations to compute the connectivity of each articulation point, the runtime complexity of Algorithm 2 remains linear in the size of the graph.

5.2.2 Finding the vertex cuts and their connectivity

Similarly, we extend the definition of a vertex cut to a k-connected vertex cut.

Definition 11 A **k-connected vertex cut** is a vertex cut whose removal from the graph would split it into k connected components.

For example, in Figure 3 the set of vertices $\langle 9, 11\rangle$ is a 2-connected vertex cut, since its removal from the graph would split it into two connected components: $\langle 1, ..., 8, 12, ..., 15\rangle$ and $\langle 10, 16, 17, 18\rangle$.

Algorithm 3 can be used to find all the vertex cuts of a given size in the graph and their connectivity.

Algorithm 3 FindVertexCuts(G, d)

input: G – the graph representing the environment, d – vertex cut size

```
1: Create a new list of vertex cuts L
2: for every subset S of nodes with size d in G do
3:     Find the connected components in G − S by running DFS
4:     n ← the number of connected components
5:     if n > 1 then
6:         Add the subset S to L
7:         connectivity[S] ← n
8: return L
```

The runtime complexity of FindVertexCuts is as follows. The number of subsets of d nodes in G is $\binom{|V|}{d}$. For each subset, we find the connected components in G without the subset by running DFS, whose runtime is $O(|V|+|E|)$. Hence, in the worst case the total runtime of the procedure is $O\big(\binom{|V|}{d} \cdot (|V| + |E|)\big)$.

Note that for a constant cut size d (which is not dependent on $|V|$), the runtime complexity of the procedure is polynomial in the graph size. On the other hand, if d depends on $|V|$ (for example, if $d = \frac{|V|}{2}$), then the runtime is exponential.

We also note here that for finding vertex cuts of size $k = 2$ (separating pairs) there is a more efficient algorithm with linear time complexity, which is based on finding the triconnected components of the graph [16]. However, its implementation is far more complicated, and for the relatively small environments that we have used in the experiments, there was no need to implement it.

Using the above definitions and algorithms from graph theory, we now present strategies for a zero-knowledge defender acting against an offline covering robot (Section 5.3) and against an online covering robot (Section 5.4).

5.3 Capturing an Offline Covering Robot

In this section we consider the case in which the defender has zero knowledge on the robot's coverage path (but has the map of the environment), while the robot has a map of the environment that includes the exact locations of the obstacles and the guards placed by the defender. We will often refer to such a

defender as an "offline defender", although the term offline here refers to the covering robot and not to the defender.

The defender's strategy is based on the following two key observations, that establish a connection between the connectivity of the articulation points and vertex cuts of the graph representing the environment, and the number of times the robot must visit them along its coverage path.

First, we prove a lower bound on the number of times each articulation point in G must be visited along any coverage path (Theorem 3). For that purpose, we will need the following definitions:

Definition 12 The **backbone subpath** of the coverage path is a simple subpath (with no repeating vertices) from the starting vertex of the coverage path to its ending vertex.

The way to get the backbone subpath of a coverage path P is by removing all the vertices that appear between two occurrences of the same vertex in P. For example, in the graph depicted in Figure 3, if the robot's coverage path is $P =< 1, 2, 3, 4, 2, 5, 6, 7, 8, 12, 13, 15, 8, 14, 8, 9, 11, 10, 16, 10, 17, 18 >$, then the backbone subpath of P is $< 1, 2, 5, 6, 7, 8, 9, 11, 10, 17, 18 >$.

Theorem 3 *Any coverage path must visit every k-connected articulation point at least k times, except for articulation points that belong to the backbone subpath of the coverage path, which must be visited at least $k-1$ times.*

Proof. Consider a k-connected articulation point v. Removing v from the graph breaks it into k connected components $C_1, ..., C_k$. The robot must visit each of these connected components along its coverage path, and in order to move between these connected components it must go through v. Assume without loss of generality that the order of these components by the first time the robot visits them along its coverage path is $C_1, ..., C_k$. Thus, the coverage path must have the following structure: $P = C_1 \rightsquigarrow v \rightsquigarrow C_2 \rightsquigarrow v \rightsquigarrow ... \rightsquigarrow C_k$. Hence, the coverage path must go through v at least $k-1$ times.

We now show that if v does not belong to the backbone subpath of the coverage path, then it must be visited once more. First, we denote the starting vertex of the coverage path by s and its terminating vertex by t. Now, let w be the last vertex on the backbone subpath that is visited before v (such a vertex must exist, since the starting vertex s always belongs to the backbone subpath).

We now claim that after visiting v, the coverage path must return to w. We prove this by contradiction. Let us assume that the coverage path does not return to w. Since vertex v does not belong to the backbone subpath, then there must be another vertex u that is visited both before v and after v along the coverage path. Thus, if the coverage path does not return to w after visiting v, then there must be another vertex u between w and v, which belongs to the backbone subpath and that the coverage path returns to after visiting v. This contradicts the fact that w is the last vertex on the backbone subpath that is visited before v.

Therefore, the coverage path must have the following structure: $s \rightsquigarrow w \rightsquigarrow v \rightsquigarrow w \rightsquigarrow t$. We also know that removing v from the graph breaks it into k connected components $C_1, ..., C_k$. Again we assume without loss of generality that the order of the components by their first visit is $C_1, ..., C_k$. Thus, $w \in C_1$, and the coverage path must have the following structure: $P = s \rightsquigarrow C_1 \rightsquigarrow v \rightsquigarrow C_2 \rightsquigarrow v \rightsquigarrow ... \rightsquigarrow C_k \rightsquigarrow v \rightsquigarrow C_1 \rightsquigarrow t$. Hence, the coverage path must go through v at least k times. □

To demonstrate the implications of Theorem 3, let us examine the graph in Figure 3. This graph has two 3-connected articulation points (c_1, c_4) and two 2-connected articulation points (c_2, c_3). The number of times each articulation point is visited depends on where the robot terminates its coverage. Let us assume that the robot starts the coverage at vertex 1. Thus, it could terminate the coverage in one of the blocks: b_2, b_5, b_6 or b_7. For example, if the coverage terminates at block b_7, then the backbone subpath of the coverage path would be: $b_1 \rightsquigarrow c_1 \rightsquigarrow b_3 \rightsquigarrow c_2 \rightsquigarrow b_4 \rightsquigarrow c_4 \rightsquigarrow b_7$. In this case, c_1, c_2, c_4 belong to the backbone subpath, thus c_1 and c_4 must be visited at least twice (since they are 3-connected) and c_2 might be visited only once (since it is only 2-connected). In addition, c_3 does not belong to the backbone subpath, thus it must be visited at least twice.

Theorem 3 implies that the optimal coverage strategy for the robot is to choose to end its coverage path in the block-cut subtree with the maximum number of articulation points, as proven by the next corollary.

Corollary 1 *The robot's optimal strategy is to choose to end its coverage path in the block-cut subtree with the maximum number of articulation points.*

Proof. The optimal behavior of the robot is to complete the coverage of a subtree T_i of the block-cut tree once it enters into it. This is because moving to a different subtree T_j in the middle of the coverage of T_i will just make the robot revisit nodes in T_i that have already been visited, when it comes back to finish the coverage of T_i. By Theorem 3 the number of visits to articulation points in the last covered subtree is one less than the number of visits to articulation points in the other subtrees of the block-cut tree. Hence, choosing to finish the coverage in the block-cut subtree with the maximum number of articulation points will minimize the number of visits to articulation points in the graph. □

Similarly, we now establish a lower bound on the number of times each vertex cut in the graph must be visited along the coverage path.

Theorem 4 *Let U be a k-connected vertex cut. Then any coverage path must visit U at least $k-1$ times. In addition, if the starting vertex of the coverage path belongs to U, then U must be visited at least k times.*

Proof. Consider a k-connected vertex cut U. Removing U from the graph breaks it into k connected components $C_1, ..., C_k$. The robot must visit each

of these connected components along its coverage path, and in order to move between these connected components it must go through one of the vertices in U. Assume without loss of generality that the order of these components by their first visit along the coverage path is $C_1, ..., C_k$. Denote the starting vertex of the coverage path by s. Now, consider two cases:

Case 1. $s \notin U$. In this case $s \in C_1$. Thus, the coverage path must have the following structure: $p = C_1 \rightsquigarrow U \rightsquigarrow C_2 \rightsquigarrow U \rightsquigarrow ... \rightsquigarrow C_k$. Hence, the coverage path must go through U at least $k - 1$ times.

Case 2. $s \in U$. In this case s does not belong to any of the connected components C_i. Thus, the coverage path must have the following structure: $p = s \rightsquigarrow C_1 \rightsquigarrow U \rightsquigarrow C_2 \rightsquigarrow U \rightsquigarrow ... \rightsquigarrow C_k$. Hence, the coverage path must go through U at least k times. □

From Theorem 4, we can conclude the following.

Definition 13 Let U be a group of vertices in the graph G. A **repetitive visit** of a coverage path P in U is any visit within U that lands on a vertex that has already been visited by P at least once before.

Corollary 2 *Let U be a k-connected vertex cut with $m < k$ vertices. Then, U will have have at least $k - m - 1$ repetitive visits along any coverage path.*

Proof. By Theorem 4, any coverage path must visit U at least $k - 1$ times. In addition, the coverage path must visit each of the m vertices in U at least once. Thus, the number of repetitive visits of the coverage path to vertices in U will be at least $k - 1 - m \geq 0$. □

Note that Corollary 2 guarantees a minimum number of visits in the vertex cut, but it does not specify which vertices in it will be visited multiple times by the covering robot.

Based on these key observations, we now present algorithm CapturingOfflineCovering Robot (COCR, shown in Algorithm 4), which specifies the assignment of guards for a defender acting against a robot with full knowledge of the environment. The algorithm works in polynomial time and provides an optimal solution in some types of environments (see Section 5.3.1 for a full analysis).

The algorithm consists of the following main steps:

1. Place guards at articulation points that must be visited by the robot more than once, while giving precedence to points that must be visited more frequently.
2. Place guards at vertex cuts of the graph, in which some of the vertices must be visited by the robot more than once, giving precedence to vertex cuts that must be visited more frequently.
3. If there are any more guards left to place after taking the first two steps, use heuristics to choose additional vertex cuts in which to place the remaining guards.

Algorithm 4 CaptureOfflineCoveringRobot(G, S)

input: G – the graph representing the environment, S – the set of guards

```
1: if starting location of the robot is known then
2:     s ← starting vertex of the robot
3: else
4:     s ← choose one of the vertices in G arbitrarily
5: APList ← FindArticulationPoints(s)
6: T ← CreateArticulationPointsTree(s)
7: PlaceGuardsAtArticulationPoints(T, S)
8: if S ≠ ∅ then
9:     PlaceGuardsAtVertexCuts(G, VC, S)
```

The algorithm first chooses the vertex s that will be the root of the block-cut tree. If the defender knows the starting location of the robot, s would be its starting vertex. Otherwise, the defender chooses one of the vertices to be the root arbitrarily. In sections 5.3.1 and 6, we prove both theoretically and empirically that selecting a different root from the robot's starting location does not have a great impact on the probability of the defender to capture the robot.

We now describe each of the procedures used in the algorithm. First, the procedure CreateArticulationPointsTree is used to build a DFS tree that contains only the articulation points of the graph and is rooted at the starting vertex of the coverage path. Let us denote this tree by T. For each articulation point v in T we store the field $childAPs[v]$, which holds all the articulation points contained in all the attached blocks of v.

```
1: procedure CreateArticulationPointsTree(v)
   input: v – the articulation point at the root node
2:     for each block B in blocks[v] do
3:         for each articulation point p in B do
4:             if p ≠ v then
5:                 Add p to childAPs[p]
6:                 parent[p] ← v
7:                 CreateArticulationPointsTree(p)
```

The procedure PlaceGuardsAtArticulationPoints describes a strategy for placing guards at the articulation points of the graph. Going from the highest connectivity of articulation points in T to the lowest, the algorithm calls the procedure RunBFSOnAPTree to place guards at articulation points with a given connectivity.

The procedure RunBFSOnAPTree assigns guards to articulation points that have a specific connectivity. By Corollary 1, the covering robot will prefer to finish its coverage path in the branch of the block-cut tree that contains the maximum number of guarded articulation points. Hence, the optimal strategy for the defender is to spread its guards evenly across the different branches of the block-cut tree, in order to minimize the number of guarded articulation

```
1: procedure PlaceGuardsAtArticulationPoints(T, S)
   input: T – the articulation points tree, S – the set of guards
2:     c_min ← the minimum connectivity of any articulation point in T
3:     c_max ← the maximum connectivity of any articulation point in T
4:     v ← root[T]
5:     c ← c_max
6:     while c ≥ c_min and S ≠ ∅ do
7:         RunBFSOnAPTree(APList, T, c, S)
8:         c ← c − 1
```

points in the terminating branch of the robot's coverage path. For that purpose, the procedure runs BFS on the articulation points tree, and assigns the guards sorted by their strength (from the guards with the highest probability of capturing the robot to the guards with the lowest probability) according to the **BFS** visit order.

In addition, if the defender knows the starting location of the robot, it can refrain from placing guards at articulation points that reside on the path leading from the root of the block-cut tree to the first split node (the first articulation point with more than one child node). This is because these articulation points are only 2-connected and they belong to the backbone subpath of the coverage, thus by Theorem 3 they could be visited only once by the robot. We mark these articulation points by using a field named $keepFree[v]$.

```
1: procedure RunBFSOnAPTree(APList, T, c, S)
   input: APList – list of articulation points, T – the tree of articulation points, c –
   connectivity, S – the set of guards
2:     v ← root[T]
3:     if defender knows starting location then
4:         while |childAPs[v]| = 1 do
5:             v ← single child of v
6:             keepFree[v] ← true
7:     Create an empty queue Q
8:     Enqueue(Q, v)
9:     while not IsEmpty(Q) do
10:        v ← Dequeue(Q)
11:        if conectivity[v] = c then
12:            g ← argmax_{i∈S} p_i                    ▷ strongest guard left
13:            Place guard g at v
14:            S ← S − {g}
15:            if S = ∅ then return
16:        for each node u ∈ childAPs[v] do
17:            Enqueue(Q, u)
```

Procedure PlaceGuardsAtVertexCuts describes how to place the remaining guards at the vertex cuts of the graph. It starts with placing guards at vertex cuts of size 2 (separating pairs), and then increases the size of the vertex cuts used, until all the remaining guards are positioned. The reason for this strategy is that any vertex cut of size k is also part of a vertex cut of size $k+1$ (adding any vertex in the graph to it creates a vertex cut of size $k+1$),

and by Corollary 2 smaller vertex cuts have potentially more revisits (ideally, we would have to first find all the vertex cuts that satisfy the conditions of Corollary 2, but computing all the vertex cuts takes exponential time). For vertex cuts of a given size, the algorithm first prefers vertex cuts that have higher connectivity, and then prefers those that are more spread across the environment, i.e., whose vertices are farther apart from each other. We define the spread of a vertex cut as the sum of the distances between the vertices in the cut. This heuristic has provided better results in practice than just choosing randomly between the vertex cuts.

1: **procedure** PLACEGUARDSATVERTEXCUTS(G, VC, S)
input: G – the graph representing the environment, VC – the list of vertex cuts, S – the set of guards
2: $d \leftarrow 2$ ▷ the current cut size
3: **while** $S \neq \emptyset$ **do**
4: $VC \leftarrow$ FindVertexCuts(G, d)
5: Sort the vertex cuts in VC first by their connectivity and then by their spread (both in descending order)
6: **for each** vertex cut $C \in L_C$ **do**
7: **for each** vertex $v \in C$ **do**
8: $g \leftarrow \text{argmax}_{i \in S}\, p_i$ ▷ strongest guard left
9: Place guard g at v
10: $S \leftarrow S - \{g\}$
11: **if** $S = \emptyset$ **then return**
12: $d \leftarrow d + 1$

Runtime complexity: The runtime of Algorithm 4 depends on the number of guards that need to be assigned, and on whether there are enough articulation points in the graph for their assignment. Let us denote the number of vertices in the graph by $|V| = n$, and the number of edges by $|E| = m$.

The algorithm begins with finding all the articulation points in the graph, which costs $O(n + m)$ (see Section 5.2.1) and building the articulation points tree, which costs additional $O(n)$. Then, for each connectivity level, the algorithm runs BFS on the articulation points tree, which takes $O(n+m)$. Thus, if c is the number of connectivity levels, this step takes in total $O(c(n + m))$. In grid environments, where each cell has at most four neighboring cells, $c \leq 4$, and the number of edges is $m = O(n)$, thus in such environments the time complexity of the last step is $O(n)$. Therefore, if there are enough articulation points for placing all the guards, the total runtime of Algorithm 4 is $O(n)$.

However, in case that are not enough articulation points for placing all the guards, the vertex cuts of the graph also need to be used. The time for computing the vertex cuts of size d is $O\left(\binom{n}{d} \cdot (n + m)\right)$ (see Section 5.2.2). In the worst-case scenario, the largest vertex cut that needs to be found is of size that equals the number of guards. Thus, if k is the number of guards, then the time needed to compute the vertex cuts is $O\left(\binom{n}{k} \cdot (n+m)\right)$. However, in practice, there was no need to use vertex cuts larger than 2 in any of the tested environments. Thus, assuming $\max_d = 2$ and $m = O(n)$, the time

needed to compute the vertex cuts is $O\left(\binom{n}{2} \cdot n\right) = O(n^3)$. Since the procedure PlaceGuardsAtVertexCuts also sorts the vertex cuts before deciding where to place the guards, its total runtime is $O(n^3 \log n)$.

Hence, in the worst case, where both the articulation points and the vertex cuts of the graph are used for the guards' assignment, the entire complexity of Algorithm 4 is $O(n^3 \log n)$.

5.3.1 Analysis of the Defender Offline Strategy

In this section we show that for certain values of k, the adversarial strategy depicted in Algorithm 4 is optimal, in the sense that its probability of stopping a robot that follows an optimal coverage strategy has the maximum possible value. We start with the following theorem, that provides an upper bound on the maximum number of times an optimal coverage path (i.e., a coverage path with minimum length) must visit every vertex of the graph.

Theorem 5 *Denote the degree of a vertex v by $deg(v)$. An optimal coverage path of a graph $G = (V, E)$ visits every vertex $v \in V$ at most $deg(v)$ times, except for vertices that reside on the backbone subpath of the coverage path which are visited at most $deg(v) - 1$ times.*

Proof. First, we prove that every vertex $v \in V$ which is not on the backbone subpath is visited at most $deg(v)$ times along the optimal coverage path. We prove by contradiction. Let P_{opt} be an optimal coverage path of G. Assume that there is a vertex v that does not belong to the backbone subpath of the coverage, and is visited by P_{opt} more than $deg(v)$ times. We will show that it is possible to build a coverage path P' that is shorter than P_{opt}.

Let us denote by s the starting vertex of the coverage path and by t its terminating vertex. Let $m = deg(v)$. We know that vertex v is visited at least $m + 1$ times along P_{opt}. We also know that v is not part of the backbone subpath, thus there must be at least $m + 1$ entries into v along the coverage path (since there must be a vertex w that is visited before and after v). Thus, P_{opt} has the following structure: $P_{opt} = s \rightsquigarrow^{p_1} v \rightsquigarrow^{p_2} v \ ... \rightsquigarrow^{p_{m+1}} v \rightsquigarrow t$.

Since v is connected to only m different edges in G, by the pigeonhole principle at least two of the subpaths $p_1, ..., p_{m+1}$ terminate with the same edge $e = (u, v)$. Let us denote these subpaths by p_i, p_j $(i < j)$. Let us denote the first vertex of p_i by w (w might be different from v, since p_i could be equal to p_1). Thus, $p_1 = w \rightsquigarrow u \rightarrow v$. Since p_j starts and ends at vertex v, it has the following structure: $v \rightsquigarrow u \rightarrow v$, thus we can reverse the order of its vertices, i.e., we can change p_j to $p'_j = v \rightarrow u \rightsquigarrow v$. Now, if $j \neq i + 1$, then we can replace subpaths p'_j and p_{i+1}, thus the coverage path will have the following structure: $s \rightsquigarrow w \rightsquigarrow u \rightarrow v \rightarrow u \rightsquigarrow v \rightsquigarrow t$. Thus, we can make this path shorter by removing the redundant visit to v, i.e., we can build the following coverage path $P' = s \rightsquigarrow w \rightsquigarrow u \rightsquigarrow v \rightsquigarrow t$. The path P' visits all vertices in the graph, but $|P'| < |P_{opt}|$. This contradicts the optimality of P_{opt}.

Now, let us assume that v does belong to the backbone subpath of the optimal coverage path. We will show that v must be visited at most $m - 1$

times. By contradiction, let us assume that v is visited at least m times. Thus, the coverage path has the following structure: $P_{opt} = s \rightsquigarrow^{p_1} v \rightsquigarrow^{p_2} v \ ... \ \rightsquigarrow^{p_m} v \rightsquigarrow t$. Since v is part of the backbone path, there are only $m-1$ distinct entries into v along the coverage path, since out of the m edges that are connected to v, one must be used only to exit from v. Thus, two of the subpaths $p_1, ..., p_m$ must terminate with the same edge $e = (u, v)$. We can use the same argument as above to show that there is a shorter coverage path than $|P_{opt}|$, which contradicts the optimality of P_{opt}. □

Using Theorem 5, we can now discuss the cases in which Algorithm 4 is guaranteed to produce the optimal defender's strategy.

Theorem 6 *Let the maximum degree in the graph G be d. If the number of articulation points in G whose connectivity is d is equal to or greater than the number of guards k, then the defender's strategy described in Algorithm 4 is optimal.*

Proof. By Lemma 5, an optimal coverage path visits every vertex $v \in V$ at most d times, or $d-1$ times if v resides on the backbone subpath of the coverage path. By Theorem 5, any such coverage path must visit every d-connected articulation point at least d times, or $d-1$ times if it is on the backbone subpath. Thus, the optimal coverage path visits every d-connected articulation point precisely d times if it is not on the backbone subpath, or $d-1$ times if it is on this subpath. Hence, d-connected articulation points are the most frequently visited vertices along the optimal coverage path. As a consequence, placing all the given k guards at these articulation points, sorted by their connectivity, is guaranteed to maximize the probability of capturing a robot that follows an optimal coverage path. □

We now show that the defender's ability to capture the robot is only mildly affected by its knowledge of the starting location. Before that, we introduce the following definition:

Definition 14 The **starting subpath** of the coverage path is a simple subpath (with no repeating vertices) from the starting vertex of the coverage path to the first split node in the block-cut tree (the first node in the tree that has more than one child).

Theorem 7 *Knowing the starting location of the robot may change only the placement of guards that are located at articulation points that reside on the starting subpath of the robot.*

Proof. According to Algorithm 4, the knowledge of the starting location of the robot has the following two effects on the placement of guards:

1. Choosing the robot's starting location as the root of the block-cut tree (lines 1–4 in Algorithm 4).
2. The placement of guards at the starting subpath of the coverage (lines 3–6 in procedure RunBFSOnAPTree).

The choice of the root of the block-cut tree does not affect the placement of the guards, since by the block decomposition theorem, the block decomposition of a graph is unique. Thus, the locations and the connectivity of both the articulation points and the vertex cuts of the graph are not affected by choosing a different root for the block-cut tree. Therefore, the only guards that may change their location as a result of knowing the robot's starting location are those that are placed at articulation points that belong to the robot's starting subpath. This is because when the defender knows the starting location of the robot, it does not place any guards at articulation points that belong the starting subpath, since these are part of the backbone subpath of the coverage, and thus may be visited only once by the robot (Theorem 3). On the other hand, when the defender does not know the starting location of the robot, it cannot assume that these articulation points may be visited only once by the covering robot (since they may belong to different branches of the block-cut tree). □

In practice, most of the articulation points reside outside the starting subpath of the robot, thus the defender's ability to capture the robot is barely affected by its knowledge of the starting location of the robot (as we show empirically in Section 6).

5.4 Capturing an Online Covering Robot

In this section we propose a strategy for a defender acting against an online covering robot, that has no map or a-priori information about the environment (shown in Algorithm 5). This strategy provides the defender a better chance of capturing the robot as compared to the offline strategy presented in Section 5.3, since it takes advantage of the robot's limited sensing capabilities in order to make it visits guarded places more frequently (see Section 6 for the comparison results).

Since the robot gets only one chance of exploring the target area, we assume that it employs a greedy policy, i.e., that at each step it chooses the best action that can be taken given the robot's current knowledge of the environment. More specifically, we assume that the robot would prefer to visit all the unguarded locations that are reachable from its current location before moving to other areas via guarded locations, since leaving an unguarded area in the middle would make the robot return to it via one of the guarded locations, which will increase the number of visits to guarded places. In addition, we assume that when the robot needs to choose between two guarded places, it would prefer to visit first the place guarded by the weaker guard (the one with the lower probability of capturing the robot), since by Theorem 3 locations that are left to the end of the coverage may be visited less times by the robot. In [25] we have described an algorithm for an online adversarial coverage, which in each step leads the robot to an unvisited location with the safest possible path from its current location, and thus fulfills the assumptions

described above. We will use this algorithm to test the defender's strategy against in Section 6.

We also assume that the defender knows the starting location of the robot (which is the case for many real-world scenarios, e.g., when securing a building that has only one entrance). In this case, the defender can employ the following two strategies to increase its chances of capturing the robot. First, the defender can block the entrance to the branch in the block-cut tree with the minimum number of articulation points (by placing a strong guard at its root), thus forcing it to terminate its coverage path at this branch, which in turn will make the robot visit the articulation points in the graph the maximal number of times (according to Theorem 3). Second, the defender can place weaker guards (guards with lower probabilities to capture the robot) closer to the robot's starting position, thus making the robot move back and forth between different branches of the block-cut tree in order to visit all the places guarded by the weaker guards, before going into deeper levels of the tree that contain stronger guards.

Therefore, the strategy described in Algorithm 5 consists of the following main steps:

1. Find the highest subtree of the block-cut tree with the minimum total connectivity of articulation points. Place guards at all the cells that are within the sensing range r of the robot from the root of this subtree (referred from here as the *blocked subtree*), and mark at least one cell as free of guards within the sensing range of the robot from the roots of the other subtrees that share the same parent node. This step will make the robot visit the subtree with the minimum total connectivity of articulation points at the end of its coverage path.
2. Sort the articulation points in the graph by the order of their connectivity.
3. For each connectivity, starting from the highest one, place guards at articulation points with that connectivity, while giving precedence to articulation points that belong to the unblocked subtrees.
4. For each subtree, use BFS (Breadth First Search) to scan the subtree and place the guards by the order of the BFS levels, starting from the weakest guards at the topmost level and finishing with the strongest guards at the bottommost level.
5. If there are any more guards left to place, use the vertex cuts of the graph to choose additional locations in which to place the remaining guards (as in the offline case).

We now describe each of the new procedures that appear in the algorithm. The procedure FindSubtreeWithMinAPConnectivity finds the highest subtree in the block-cut tree that contains the minimum total connectivity of articulation points. For that purpose, it first invokes the recursive procedure ComputeTotalAPConnectivity, which computes the total connectivity of the articulation points in every subtree. For each vertex v, the total connectivity in the subtree rooted at v is stored in a field named $tc[v]$. Then, it finds the highest node

Algorithm 5 CaptureOnlineCoveringRobot$(s, \mathcal{O}, S)$

input: s – the starting vertex of the coverage, $\mathcal{O}$ – robot's observation function, S – the set of guards

APList ← FindArticulationPoints(s)
T ← CreateArticulationPointsTree(APList)
T_{min} ← FindSubtreeWithMinAPConnectivity(T)
U ← subtrees of Tthat share the same parent with T_{min}
BlockSubtree($T_{min}, U, \mathcal{O}$)
c_{min} ← minimum connectivity of any articulation point
c_{max} ← maximum connectivity of any articulation point
$c \leftarrow c_{max}$ ▷ iterate over all connectivities
while $c \geq c_{min}$ **and** $S \neq \emptyset$ **do**
    RunBFSOnAPTreeOnline(APList, T_{min}, c, S)
    for each tree $t \in U$ **do**
        RunBFSOnAPTreeOnline(APList, t, c, S)
    $c \leftarrow c - 1$
if $S \neq \emptyset$ **then**
    PlaceGuardsAtVertexCuts(G, VC, S)

in the block-cut tree that has more than one child, and returns the subtree rooted at the child with the minimum total connectivity.

procedure FindSubtreeWithMinAPConnectivity(T)
input: T – the articulation points tree
output: the subtree with minimum total connectivity of articulation points
    ComputeTotalAPConnectivity($root[T]$)
    $v \leftarrow root[T]$
    while $childAPs[v] = 1$ **do**
        $v \leftarrow$ single child of v
    $w \leftarrow \text{argmin}_{u \in childAPs[v]}\, tc[u]$
    return the subtree rooted at w

procedure ComputeTotalAPConnectivity(v)
input: v – current vertex in the articulation points tree
    $tc[v] \leftarrow connectivity[v]$
    for each $u \in childAPs[v]$ **do**
        $tc[v] \leftarrow tc[v]$ + ComputeTotalAPConnectivity(u)

The procedure BlockSubtree "blocks" the entrance to the subtree with the minimum total connectivity of articulation points (denoted by T_{min}), in order to make the robot finish its coverage path within this subtree. This is achieved by placing a guard at the root of T_{min}, and making sure that all the guards at the roots of the other subtrees are weaker than this guard. If there are not enough weaker guards (e.g., all the guards have equal probability of capturing the robot), then we also place guards at all the cells that can be sensed by the robot from the root of T_{min}, while keeping at least one of the cells that can

be sensed by the robot from the roots of the other subtrees free from guards. This way, the robot will prefer to visit all these subtrees before entering the blocked subtree (since the number of safe cells that can be sensed from the roots of these subtrees is greater by at least 1). See Section 5.4.1 for a formal analysis.

```
1:  procedure BlockSubtree(T, U, O)
    input: T – the subtree to be blocked, U – subtrees that share the same parent as T, O
    – the observation function
2:      Place a guard with highest p_i at root[T]
3:      w ← number of guards with probability p_i < p_max
4:      if w < |U| then
5:          {Place guards within the sensing range from root[T]}
6:          for each vertex v ∈ O(root[T]) do
7:              if v ∈ T then
8:                  Place a guard with probability p_min at v
9:          for each subtree A ∈ U do
10:             {Keep one of the cells within the sensing range from root[A] safe}
11:             v ← a vertex in {u|u ∈ descendants of A and u ∈ O(root[A])}
12:             keepFree[v] ← true
```

The procedure RunBFSOnAPTreeOnline assigns guards to the articulation points of the graph by running BFS on the articulation points tree. In contrast to the offline version of this procedure, here the guards are assigned according to the levels of the BFS tree, and not according to the BFS visit order of the nodes. The strength of the assigned guards (i.e., their probability of capturing the robot) is matched to the level of the BFS tree at which they are located, i.e., if the guards have m different probabilities of capturing the robot $p_1, ..., p_m$, then all the guards that have probability p_i of capturing the robot should be located at level i of the BFS tree. This allocation is intended to make a myopic robot move back and forth between the different branches of the block-cut tree before going into deeper levels. Since the number of guards that have probability of p_i of capturing the robot is not necessarily equal to the number of articulation points at level i of the tree, we allow level i of the tree to contain guards that have probability of at least p_i of capturing the robot.

For each vertex v we keep the following fields: $min\_level[v]$ – the minimal probability level that the guard assigned to v should have, and $level[v]$ – the probability level that the guard located at v belongs to. These fields make sure that the assigned guards to the child nodes of v are stronger than the guard assigned to v.

The procedure FindMinGuardLevelForAPs finds the minimum probability level that should be used for assigning guards at articulation points with a given connectivity. In case that there are more guards than articulation points, we would like to use the strongest guards for the articulation points and leave the other ones for the vertex cuts of the graph.

```
1: procedure RunBFSOnAPTreeOnline(APList, T, c, S)
   input: APList – list of articulation points, T – the tree of articulation points, c –
   connectivity, S – the set of guards
2:     v ← root[T]
3:     while |childAPs[v]| = 1 do
4:         v ← single child of v
5:         keepFree[v] ← true
6:     Create an empty queue Q
7:     min_level[v] ← FindMinGuardLevelForAPs
8:     Enqueue(Q, v)
9:     while not empty(Q) do
10:        v ← Dequeue(Q)
11:        if connectivity[v] = c and not keepFree[v] then
12:            g ← a guard in S with probability of at least p_{min_level[v]}
13:            Place g at v
14:            level[v] ← the level of guard g
15:            S ← S − {g}
16:            if S = ∅ then return
17:        for each node u ∈ childNodes[v] do
18:            if connectivity[v] = c and v contains a guard then
19:                min_level[u] ← level[v] + 1
20:            else
21:                min_level[u] ← level[v]
22:            Enqueue(Q, u)
```

```
1: procedure FindMinGuardLevelForAPs(APList, c, S)
   input: APList – the list of articulation points, c – connectivity, S – the set of guards
2:     Sort the guards in S in a decreasing order of their probability of stopping the robot
3:     k ← |{v|v ∈ APList and connectivity[v] = c and not keepFree[v]}|
4:     if |S| ≤ k then
5:         g ← argmin_{i∈S} p_i                          ▷ the weakest guard
6:     else
7:         g ← S[k]                                      ▷ the guard at the k^th place
8:     return the level of g
```

Runtime complexity: We now analyze the runtime complexity of Algorithm 5. As in the offline case, the algorithm's runtime depends on the number of guards that need to be assigned, and on whether there are enough articulation points in the graph for their assignment. The algorithm begins with finding all the articulation points in the graph and building the articulation points tree, which takes time of $O(n)$ (see Section 5.3). Computing the total connectivity of every subtree of the block-cut tree using the recursive procedure ComputeTotalAPConnectivity takes $O(n+m)$ (as every edge of the graph is traversed only once), and finding the subtree with the minimum total connectivity costs additional $O(n)$. The runtime of the procedure BlockSubTree depends on the number of subtrees that share the same parent as the blocked subtree, and the sensing radius of the robot, and in the worst case it is $O(n)$. Then, for each connectivity level, the algorithm runs BFS on each of these subtrees. Since the subtrees do not share any vertices, the runtime of running BFS on all of them is $O(n+m)$. Thus, if the number of connectivity levels is c,

the total runtime of this step is $O(c(n+m))$. In grid environments, $c \leq 4$, and the number of edges is $m = O(n)$. Therefore, in case that there are enough articulation points for placing all the guards, the total runtime of Algorithm 5 is $O(n)$.

However, if there are not enough articulation points for placing all the guards, the vertex cuts of the graph also need to be used. As in the offline case, the algorithm calls the procedure PlaceGuardsAtVertexCuts, whose runtime complexity is $O(n^3 \log n)$ (see Section 5.3). Therefore, the total runtime complexity of the algorithm is $O(n^3 \log n)$, the same as the algorithm in the offline case (Algorithm 4).

5.4.1 Analysis of the Defender Online Strategy

In this section we provide a theoretical analysis of the online defender's strategy described in Algorithm 5. Specifically, we show that it is optimal for certain values of k (the number of guards), in the sense that its probability of stopping an online covering robot with a sensing range r is maximized. First, we denote by T_{min} the blocked subtree with the minimum total connectivity of articulation points and by T the subtrees that share the same parent with T_{min}. Let us also denote by v the common parent of the subtrees in T. We start with the following lemma, that proves that an optimal coverage path of the robot must end in the blocked subtree T_{min}.

Lemma 1 *Given the guards assignment by Algorithm 5, an optimal coverage path of a robot with sensing range r must end in the subtree T_{min}.*

Proof. When the robot first visits the vertex v, it needs to choose between going into subtree T_{min} or one of the subtrees in T that share the same parent. Since the robot can observe only r nodes from its current location, its decision of which subtree to choose can be only based on the number of guards that it can detect from v. The procedure BlockSubTree places guards in T_{min} such that the number of guards in $T_{\min}$ that the robot can sense from v is greater by at least one guard than the number of guards that can be sensed from v in all the other subtrees. Thus, it makes the robot choose T_{min} as the last subtree to be covered. In addition, it places a guard at the roots of all the subtrees that are children of v (which are all articulation points). Thus, after the robot enters into a given subtree $t \in T$ it has no incentive of leaving it and moving to another subtree, since this will make it revisit the vertex containing this guard and some of the other nodes that have already been visited in this subtree at least twice more (once on its way back to v and the second time when it needs to go back to this subtree in order to finish its coverage). This will make the coverage path suboptimal, since the same coverage path without going out from this subtree and getting back would still cover all the nodes in the graph but will visit a node protected by a guard at least one time less. □

We now use Lemma 1 to establish a lower bound on the number of times each articulation point in G must be visited by the optimal coverage path of the robot.

Theorem 8 *Given the guards assignment by Algorithm 5, an optimal coverage path of a robot with sensing range r must visit every k-connected articulation point in a subtree $t \in T$ at least k times, and every k-connected articulation point in the subtree T_{min} at least $k-1$ times.*

Proof. By Theorem 3, any coverage path must visit every k-connected articulation point at least k times, except for articulation points on the backbone subpath of the coverage path, which must be visited at least $k-1$ times. By Lemma 1, an optimal coverage path of a robot with sensing range r must end in the subtree T_{min}, thus all the articulation points that belong to the other subtrees are not part of the backbone subpath. This is because the robot must visit v (the common parent of the subtrees) before and after each of these articulation points, thus they cannot be part of a simple path that leads from the starting vertex of the coverage to its terminating vertex. Therefore, the robot must visit every k-connected articulation point in the subtrees $t \in T$ at least k times and the articulation points in the subtree T_{min} at least $k-1$ times. □

Next, we prove that the Algorithm 5 is optimal with respect to the assignment of guards to the articulation points in the graph.

Theorem 9 *The assignment of guards by Algorithm 5 maximizes the number of times a robot with sensing range r must visit each articulation point in G along its coverage path.*

Proof. Denote by v the parent of subtree T_{min}. v is chosen as the highest node in the block-cut tree with more than one child. Thus, all the articulation points that reside on the path from the root of the block-cut tree to v need to be visited by the robot only once, and need not be protected by guards. Indeed, Algorithm 5 places guards only at articulation points that belong to the subtrees that are children of v. The guards are placed in the order of the articulation points' connectivity. Thus, articulation points that are more frequently visited by the covering robot receive a guard assignment before those that are less frequently visited. In addition, for each connectivity k, the algorithm assigns guards to all the k-connected articulation points in the unblocked trees T before assigning guards to k-connected articulation points in T_{min}, which may be visited one time less than the articulation points in T (Theorem 8). □

As in the offline case, we can now prove that for certain number of guards k and in certain types of environments, the online defender strategy is optimal.

Theorem 10 *Let the maximum degree in the graph G be d. If the number of articulation points in G whose connectivity is d is equal to or greater than the number of guards k, then the defender's online strategy described in Algorithm 5 is optimal.*

Proof. By Lemma 5, an optimal coverage path visits every vertex $v \in V$ at most d times, except for vertices that belong to the backbone subpath that are visited at most $d-1$ times. By Theorem 8, given the guard assignment by Algorithm 5, an optimal coverage path of a robot with sensing range r must visit every d-connected articulation point at least d times, except for d-connected articulation points on the backbone subpath of the coverage that must be visited at least $d-1$ times. Thus, when the maximum degree of each articulation point is d, the optimal coverage path visits every d-connected articulation point precisely d times, except for d-connected articulation points on the backbone subpath that are visited precisely $d-1$ times. Hence, d-connected articulation points are the most frequently visited vertices along the optimal coverage path. As a consequence, placing all the given k guards at these articulation points, while giving precedence to articulation points that belong to one of the subtrees in T (and thus are not part of the backbone subpath), is guaranteed to maximize the probability of capturing a robot that follows this path. □

6 Empirical Evaluation

In previous sections we have theoretically analyzed the two proposed defender strategies against an offline and an online covering robot, and provided various optimality bounds on their solutions. As we have shown, some of these bounds are not tight, and others are tight only in certain types of environments. Therefore, in this section we evaluate the performance of these strategies in various types of simulated environments and compare them with a baseline strategy, which scatters the guards randomly across the environment. We use specific maps to illustrate the operation of the algorithms and we also report on the statistical analysis of their behavior based on multiple randomly generated maps with varying parameters, such as number of obstacles, number of guards, etc.

In the offline case, we evaluate the defender's strategy against the robot's state-of-the-art offline coverage algorithm GAC (Greedy Adversarial Coverage) [28], and for small instances we also evaluate it against the robot's optimal coverage strategy, which can be computed using Value Iteration [27]. In the online case, we evaluate the defender's strategy against the state-of-the-art online coverage algorithm OAC (Online Adversarial Coverage) [25]. Note that in the online case, the robot's optimal coverage strategy cannot be computed in advance, since the robot's knowledge of the environment is changing during the coverage.

We first use a specific grid map of size 10×10 to illuminate the differences between the defender's strategies. 25% of the map's cells contain obstacles, whose locations are randomly chosen. Figure 5 shows the sample map, and the block-cut tree of its representative graph. Each articulation point is marked with the prefix AP followed by its index, and each block is marked with the

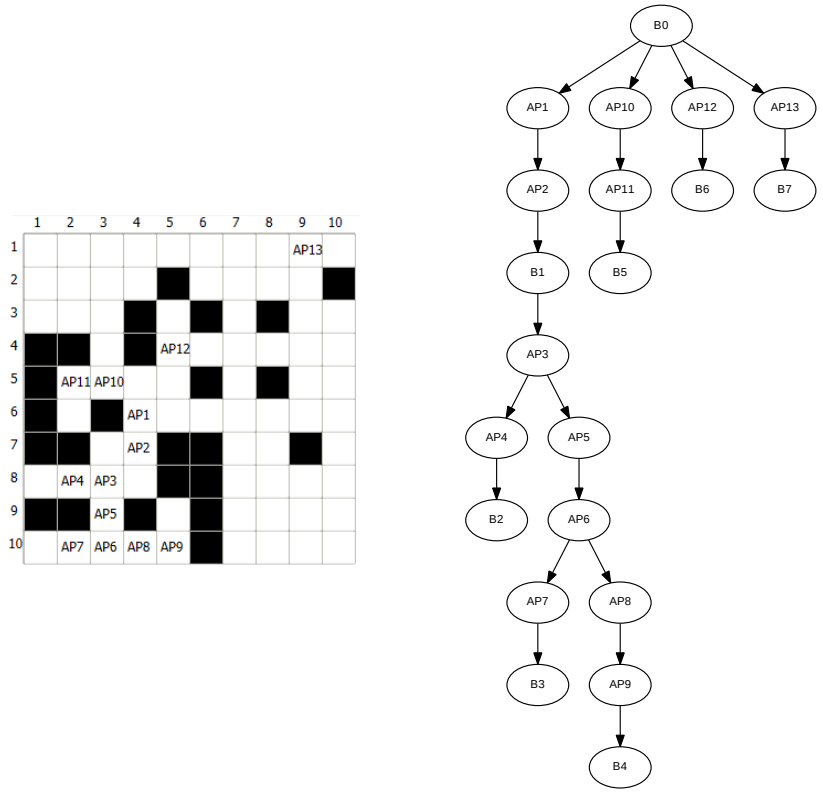

Fig. 5: The sample map (left) and the block-cut tree of its underlying graph (right).

prefix B followed by its index. We assume here that the covering robot starts at cell (1, 1), and its starting location is known to the defender.

As can be seen, the block-cut tree of this map contains 13 articulation points, two of which are 3-connected (located at cells (8, 3) and (10, 3)) and all the others are 2-connected.

In our sample run, the defender possesses 15 guards, which belong to 5 groups with probabilities of capturing the robot ranging from $p_1 = 2\%$ to $p_5 = 10\%$ (in intervals of 2%). Changing the absolute values of these probabilities affects only the scaling of the results, but not how the defender's strategy works. Figure 6 shows where the defender has chosen to place its guards (the magenta cells) according to its offline strategy (left figure) and its online strategy (right figure). Each articulation point is marked with the prefix AP followed by its connectivity, and each guarded vertex cut is marked with the prefix VC followed by its spread.

Since the defender owns 15 guards, 13 of them have been assigned to the 13 articulation points and the remaining two have been assigned to the vertex cut consisting of cells (1,6) and (4,3), which has the largest distance between its vertices amongst all the vertex cuts.

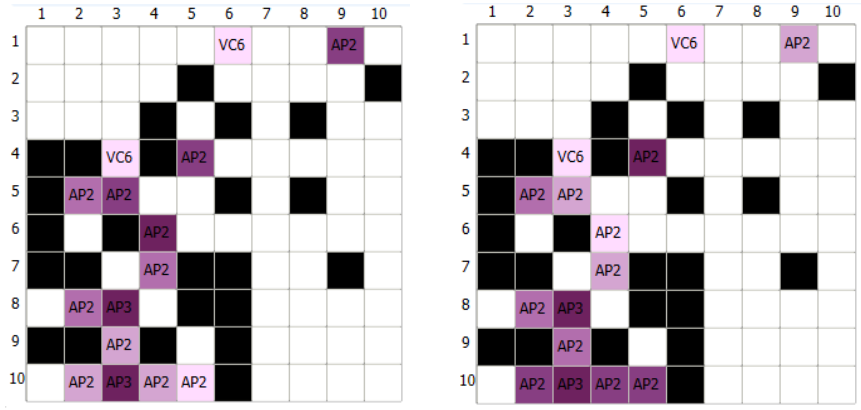

Fig. 6: The locations of the guards as determined by the defender's strategies. Cells colored with darker shades contain guards with higher probabilities of capturing the robot. The left figure shows the offline defender strategy while the right figure shows the online strategy on the same map. The robot starts the coverage in cell (1,1). Each articulation point is specified by the prefix AP followed by its connectivity, and each vertex cut is specified by the prefix VC followed by its spread.

Both strategies place two of the strongest guards at the 3-connected articulation points, and spread the other guards evenly across the various branches of the block-cut tree. In the online strategy, the defender places one of the strongest guards at the articulation point (4,5) in order to block the entrance to the branch that consists of cells (4,5) and (3,5). This branch contains the least number of articulation points, thus blocking it would cause the robot to visit the articulation points in all the other branches the maximal number of times. In addition, the strength of the guards match the depth of the block-cut tree at which they are placed. For example, if we examine the BFS subtree that is rooted at cell (6,4), the guard placed at this root has probability p_1 of capturing the robot, while the cell (7,4) which belongs to the second level of the subtree contains a guard with probability p_2, the cells (8,2) and (9,3) which belong to the third level contain a guard with probability p_3, the cells (10,2) and (10,4) which belong to the fourth level contain a guard with probability p_4. Lastly, the cell (10,5) contains a guard with probability p_4 and not p_5, since all the strongest guards have already been used (for blocking the branch with the minimum number of articulation points and for the 3-connected articulation points).

Figure 7 shows the results of running the robot's coverage algorithm OAC using a sensor with $r = 2$ against both defender's strategies. The number in each cell indicates the number of visits by the robot in that cell.

The defender's probability of capturing the robot when using its offline strategy on this map is 83.27%, whereas its probability of capturing the robot is 87.21% when it uses its online strategy. As can be seen in Figure 7, in the online strategy the blocking of the branch that consists of cells (4,5) and (3,5) caused the robot to visit all the 3-connected articulation points 3 times and all

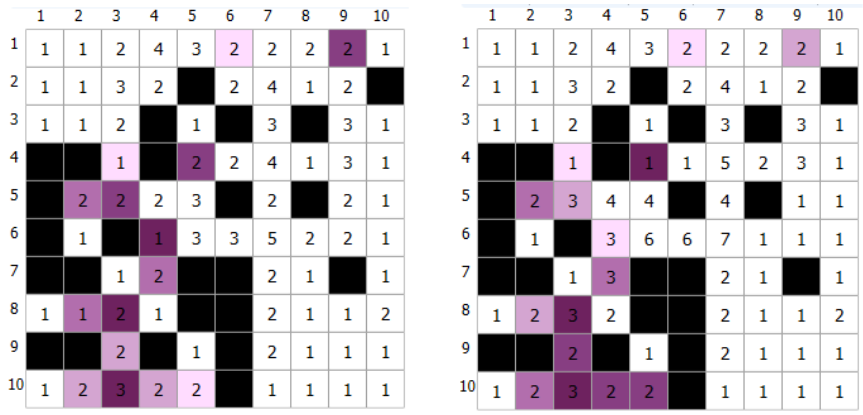

Fig. 7: The number of times an online covering robot visits each of the cells of the sample map, when acting against a defender with an offline strategy (left) and a defender with an online strategy (right).

the 2-connected articulation points at least twice, since the robot finished the coverage at the blocked branch. This in turn increased the number of times the robot has visited guarded cells from 28 to 33.

Next, we evaluate the defender's strategy against an offline covering robot (Algorithm 4) on 50 randomized maps of size 20×20. 30% of the cells in each map contain obstacles, whose locations were randomly chosen. The number of guards that have been placed on the map varied between 1 and 100, and they belonged to 5 different groups with probabilities of capturing the robot ranging from 0.6% to 3% (in intervals of 0.6%). A low probability was chosen so we could measure the effect of adding a large number of guards to the graph, up to 25% of the map's size. Figure 8 shows the probability that the defender will be able to capture the robot along its coverage path when using one of the following three strategies:

1. The random baseline strategy, in which the guards' locations are chosen randomly.
2. The defender's offline strategy (Algorithm 4), when the defender does not know the starting location of the robot.
3. The defender's offline strategy (Algorithm 4), when the defender knows the starting location of the robot.

The graphs are shown with standard deviation error bars.

The suggested defender's strategy clearly outperforms the random baseline strategy. On average, it improves the chances of capturing the robot by 33.8%, and for specific numbers of guards, it can increase those chances by more than 100% (e.g., for $k = 5$ the increase is 108%). The difference between the strategies is statistically significant (ANOVA test yields $p = 5.88 \cdot 10^{-5}$).

As the number of allocated guards gets higher, the difference between the strategies' performances gets smaller. This is because when the environment

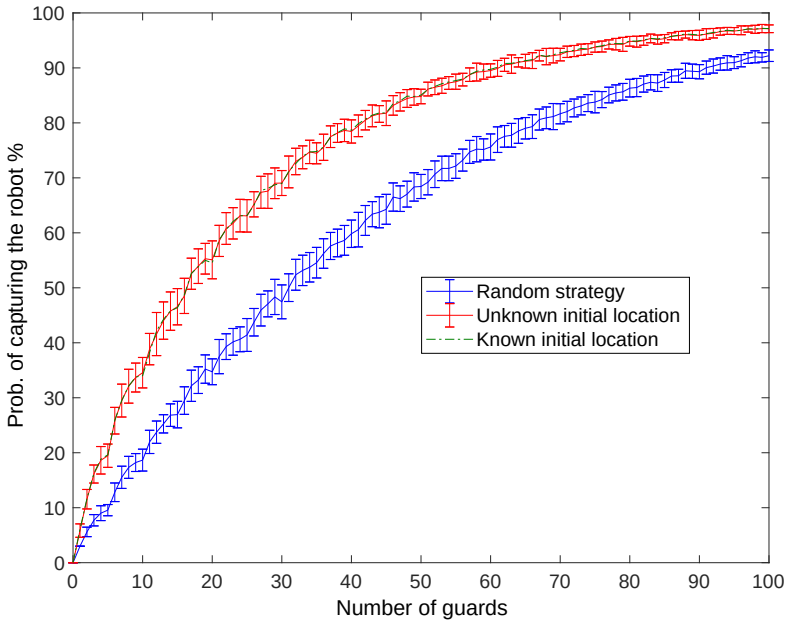

Fig. 8: The probability of capturing an offline covering robot when the defender uses its offline strategy with and without knowledge of the robot's starting location vs. using the random baseline strategy.

is filled with guards, their specific locations has a smaller affect on the overall probability of capturing the robot. In addition, we can notice that the knowledge of the starting location does not change much the ability of the defender to capture the robot.

For small-sized maps (up to 7×7) it is possible to compute the optimal coverage path for the robot, i.e., the coverage path with minimum probability of being captured [27]. Thus, to ensure the robustness of the defender's strategy, we have again tested it against the random baseline strategy, but now when the robot is following an optimal coverage path. For this experiment we have used 50 randomized maps of size 5×5 with the same settings as in the previous experiment, except for the number of guards that varied between 1 and 15 (due to the smaller map size). Figure 9 shows the results. Clearly, the defender's offline strategy outperforms the random strategy in this case too. However, the gap between the strategies' performances is narrower here. This is due to the smaller environment sizes, in which there are very few vulnerability points (e.g., articulation points) that can be exploited by the defender.

We now evaluate the defender's strategies against an online covering robot. We have used the same map settings as in the previous experiment. The robot's sensing range was set to $r = 2$. The robot's starting location was randomly

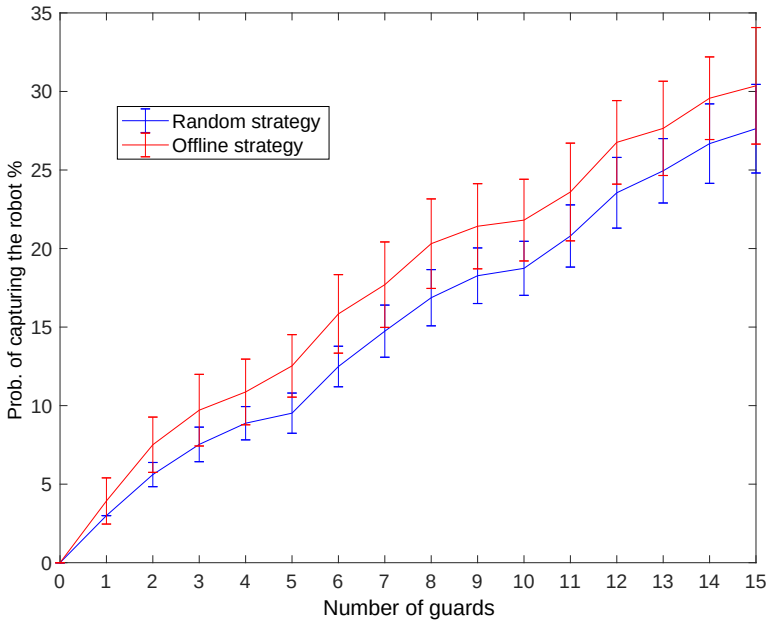

Fig. 9: The probability of capturing a robot following an optimal coverage path when the defender uses its offline strategy vs. using the random baseline strategy.

chosen. However, this time we assumed that the defender knows the starting location of the robot. Figure 10 shows the probability of capturing the covering robot, when the defender is using its offline strategy (Algorithm 4), online strategy (Algorithm 5) and the random baseline strategy.

As anticipated, for both strategies the probability of stopping the robot increases as we add more guards to the map. Both the offline and online defender's strategies show significant improvement over the random baseline strategy. When the number of guards is lower than 20, the online and the offline strategies provide similar results. This is because there is a small number of guards that belong to each probability level, thus the online strategy cannot exploit the depth of the block-cut tree in order to match the strength of the guards with the levels of the tree. However, when the number of guards gets above 20, the online strategy dominates the offline strategy; on average the online strategy increases the probability of capturing the robot by 1.64%. The difference between the strategies is statistically significant (ANOVA test yields $p = 6.685 \cdot 10^{-5}$).

The relatively small gap between the offline and the online strategies can be explained by the fact that in both cases the adversary has zero knowledge on the robot's coverage strategy. The only added information the defender has in the online case is that it knows the sensing radius of the robot, but this

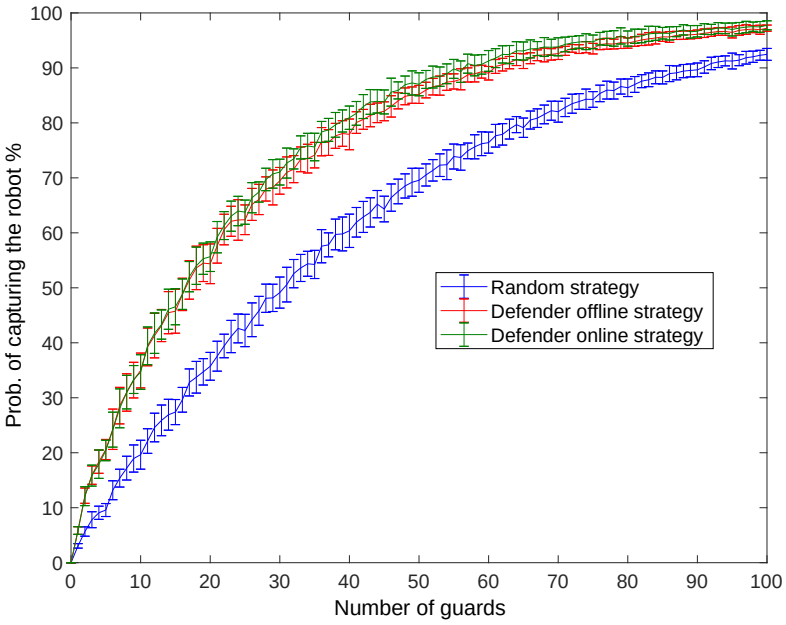

Fig. 10: The probability of capturing an online covering robot when the defender uses its online strategy vs. the offline strategy and the random baseline strategy for various number of guards.

parameter alone has almost no affect on the defender's ability to capture the robot (as will be shown in Figure 12). Moreover, in both cases the defender assigns guards to locations that must be visited multiple times by any covering robot (regardless of its sensing capabilities or its coverage strategy).

Next, we wanted to examine the effect of changing the obstacle densities in the environment on the ability of the defender to capture the robot using the three different strategies. We have run the defender's strategies on 50 randomly-generated maps of size 20×20 with obstacle ratios between 0% and 35%, in steps of 1% (in higher ratios of obstacles, most of the generated maps are disconnected). The defender was given 30 guards in all maps. Figure 11 shows the results.

As can be seen in Figure 11, the random baseline strategy is not affected much by the obstacles density, since it does not exploit the topology of the environment. The increase in the capture probability when the obstacle ratio gets above 20% is due to the fact that in higher obstacle ratios, there are less free cells for the robot to cover, thus its chance of being caught by the same number of guards increases, regardless of the strategy used. On the other hand, the success rate of the other two strategies grows consistently with the increase in the number of obstacles. This is because the more obstacles there are in the environment, the more places that the robot has to visit multiple times (more

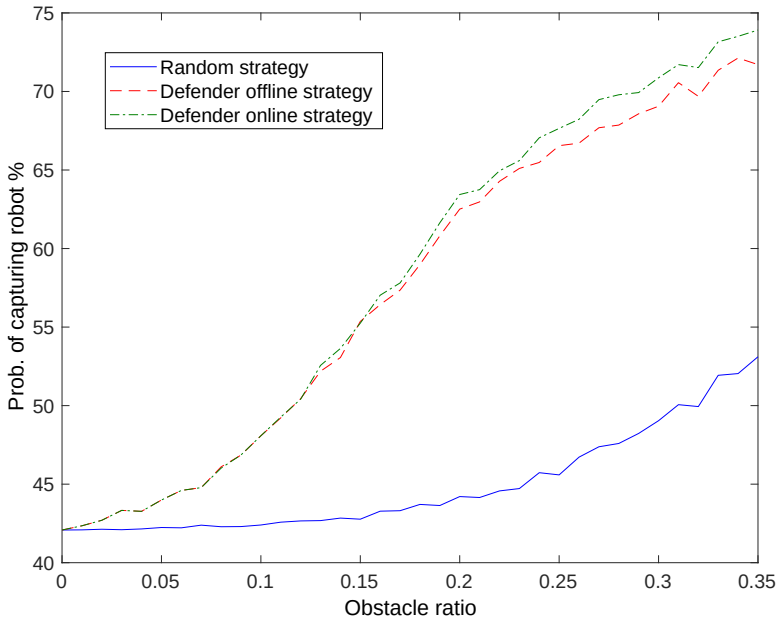

Fig. 11: The probability of capturing an online covering robot when the defender uses its online strategy vs. the offline strategy and the random baseline strategy for various obstacle densities.

articulation points and vertex cuts in the graph), which both strategies take advantage of.

We also wanted to examine the effect of changing the robot's sensor range on its ability to complete the coverage. For that purpose, we have evaluated the defender's online strategy against an online covering robot with varying sensor ranges between 1 and 10, using the same settings as in the previous experiments (50 randomly-generated maps with size 20×20, 30% of the cells contain obstacles and 5 levels of guards). The number of guards the defender could use was fixed to 30. Figure 12 shows the results. As evident, the sensing radius of the robot has almost no affect on its probability of completing its coverage task. There is a two-fold explanation for this fact. First, the defender needs to use its knowledge of the robot's sensing capabilities only in rare cases. The only place in Algorithm 5 where the robot's observation function is used is in the procedure BlockSubtree, and only in the case that there are not enough guards that are weaker than the strongest guard to assign to the roots of the non-blocked subtrees. In most environments the number of non-blocked subtrees is smaller than the number of weak guards, thus the knowledge on the sensing radius of the robot is usually not used. Second, the robot's general coverage policy is also not greatly influenced by its sensing radius. The robot

employs a greedy policy which leads it to visit the branches of the block-cut tree in a BFS order, since it gives precedence to visiting all the places guarded by weaker guards which are located at the higher levels of the tree, before going into deeper levels of the tree that contain stronger guards. Changing the robot's sensing radius might change the order in which the robot visits the nodes in the tree that belong to the same BFS level (which contain guards of the same strength), but not the order by which it visits the tree levels, which determines the number of visits to each guarded location.

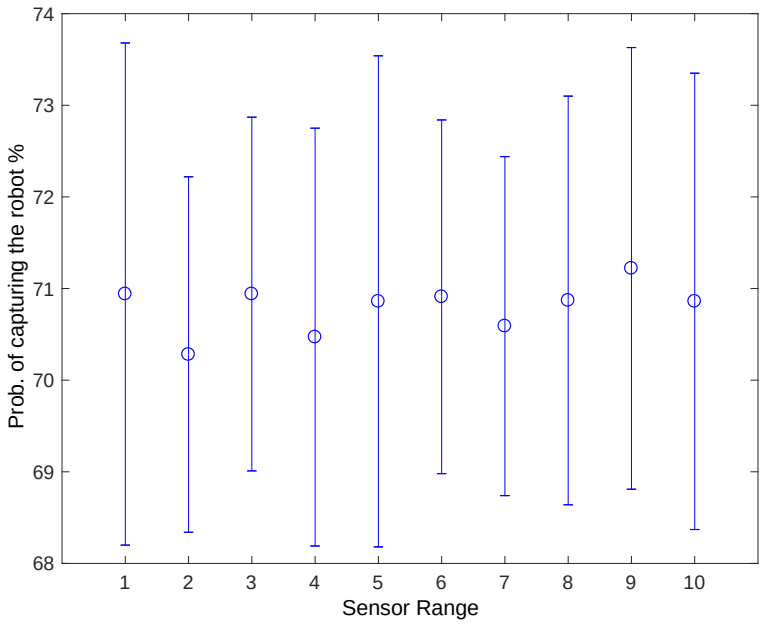

Fig. 12: The probability of capturing an online covering robot with various sensor ranges.

Lastly, we have measured the runtime needed to compute the defender's offline and the online strategies for increasing environment sizes: from a grid of 10×10 squares to a grid of 40×40 squares. The average CPU runtime was measured on a single core of an Intel Core i7-3770 with 3.4GHz and 8GB RAM. Figure 13 shows the results.

As expected by the runtime complexity analysis, the runtime graph fits the graph of polynomial of degree 3 (found using the Matlab function polyfit). The runtime of the offline and the online strategies is approximately the same, since the part that takes the longest time is the computation of the vertex cuts of the graph, which is common in both algorithms.

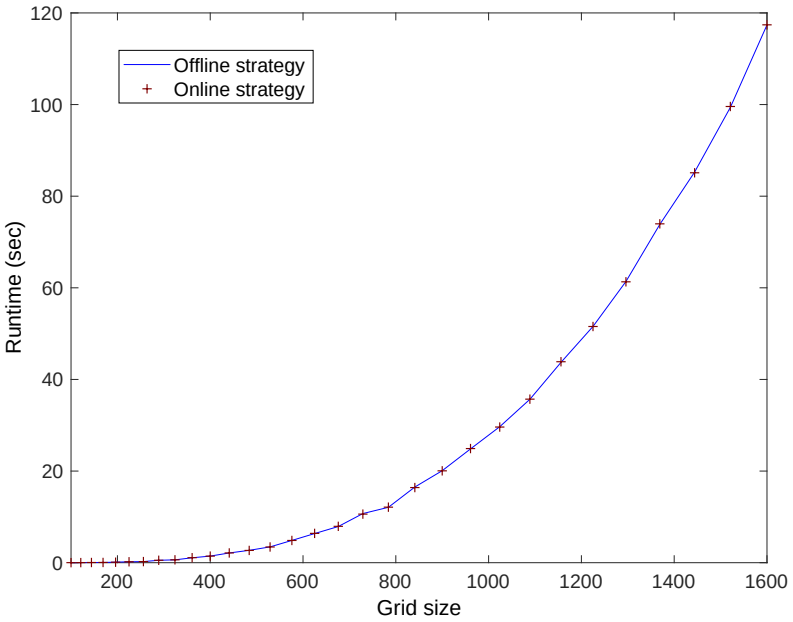

Fig. 13: The runtime for computing the defender's offline and online strategies in various environment sizes.

7 Discussion and Future Research

In this paper we have presented the coverage defending problem and discussed its various aspects. First, we have shown that there is an optimal and easily computed strategy for a full-knowledge defender, who knows the coverage path of the robot ahead of time. Second, we have shown that finding an optimal strategy for a zero-knowledge defender is an $\mathcal{NP}$-Hard problem. Third, we proposed a strategy for a zero-knowledge defender that takes advantage of the target area's topology in order to make the covering robot visit guarded locations more frequently. We have proven that this strategy, which can be computed in polynomial time, is optimal with respect to the probability of capturing the robot in certain types of environments and for certain numbers of guards. We have also compared the performance of this strategy to a random baseline strategy, in which the guards are randomly scattered across the environment. We have shown empirically that our suggested strategy increases the chances of capturing the robot by 33.8% on average, and sometimes by more than 100% (for a small number of guards).

We have also discussed different models of the covering robot: an offline covering robot, which possesses a map of the target area, vs. an online covering robot, which has no prior knowledge of the area. We have shown that when protecting an area from an online covering robot, the defender can take

advantage of the limited sensory abilities of the robot in order to control the order in which it covers the various parts of the target area, thus increasing the number of times that it visits the guarded locations in the area.

There are several areas we plan to pursue in future work. First, we would like to develop new coverage strategies for the robot, assuming that it knows the defender's strategy ahead of time. Second, we would like to extend the defender's model to cases where it needs to protect its territory from being covered by a multi-robot group instead of a single robot. Finally, we would like to consider stronger models of the guards, e.g., when the guards have some finite range of detecting the robot or when they are allowed to move.

References

1. Arkin, E.M., Fekete, S.P., Mitchell, J.S.: The lawnmower problem. In: Canadian Conference on Computational Geometry (CCCG-93), pp. 461–466 (1993)
2. Basilico, N., De Nittis, G., Gatti, N.: Adversarial patrolling with spatially uncertain alarm signals. Artificial Intelligence **246**, 220–257 (2017)
3. Bochkarev, S., Smith, S.L.: On minimizing turns in robot coverage path planning. In: IEEE International Conference on Automation Science and Engineering (CASE-16), pp. 1237–1242 (2016)
4. Borie, R., Tovey, C., Koenig, S.: Algorithms and complexity results for graph-based pursuit evasion. Autonomous Robots **31**(4), 317–332 (2011)
5. Choset, H.: Coverage for robotics–a survey of recent results. Annals of mathematics and artificial intelligence **31**(1-4), 113–126 (2001)
6. Chung, T.H., Hollinger, G.A., Isler, V.: Search and pursuit-evasion in mobile robotics. Autonomous robots **31**(4), 299–316 (2011)
7. Colegrave, J., Branch, A.: A case study of autonomous household vacuum cleaner. AIAA/NASA CIRFFSS p. 107 (1994)
8. Fekete, S.P., Friedrichs, S., Kröller, A., Schmidt, C.: Facets for art gallery problems. Algorithmica **73**(2), 411–440 (2015)
9. Gabriely, Y., Rimon, E.: Spanning-tree based coverage of continuous areas by a mobile robot. Annals of Mathematics and Artificial Intelligence **31**(1-4), 77–98 (2001)
10. Gabriely, Y., Rimon, E.: Competitive on-line coverage of grid environments by a mobile robot. Computational Geometry **24**(3), 197–224 (2003)
11. Galceran, E., Carreras, M.: A survey on coverage path planning for robotics. Robotics and Autonomous Systems **61**(12), 1258–1276 (2013)
12. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman (1979)
13. Girard, A.R., Howell, A.S., Hedrick, J.K.: Border patrol and surveillance missions using multiple unmanned air vehicles. In: IEEE Conference on Decision and Control (CDC-04), vol. 1, pp. 620–625 (2004)
14. Harary, F.: Graph theory. Addison-Wesley, Reading, MA (1969)

15. Hazon, N., Kaminka, G.A.: On redundancy, efficiency, and robustness in coverage for multiple robots. Robotics and Autonomous Systems **56**(12), 1102–1114 (2008)
16. Hopcroft, J.E., Tarjan, R.E.: Dividing a graph into triconnected components. SIAM Journal on Computing **2**(3), 135–158 (1973)
17. Hopcroft, J.E., Tarjan, R.E.: Efficient algorithms for graph manipulation. Communications of the ACM **16**(6), 372–378 (1973)
18. Lee, D.T., Lin, A.K.: Computational complexity of art gallery problems. IEEE Transactions on Information Theory **32**(2), 276–282 (1986)
19. Luo, C., Yang, S.X., Stacey, D.A., Jofriet, J.C.: A solution to vicinity problem of obstacles in complete coverage path planning. In: IEEE International Conference on Robotics and Automation (ICRA-02), vol. 1, pp. 612–617 (2002)
20. Messous, M.A., Senouci, S.M., Sedjelmaci, H.: Network connectivity and area coverage for UAV fleet mobility model with energy constraint. In: IEEE Wireless Communications and Networking Conference (WCNC-16), pp. 1–6 (2016)
21. Nicoud, J.D., Habib, M.K.: The Pemex-B autonomous demining robot: perception and navigation strategies. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS-95), vol. 1, pp. 419–424 (1995)
22. Pita, J., Jain, M., Marecki, J., Ordóñez, F., Portway, C., Tambe, M., Western, C., Paruchuri, P., Kraus, S.: Deployed ARMOR protection: the application of a game theoretic model for security at the Los Angeles International Airport. In: International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-08), Industry Track, pp. 125–132 (2008)
23. Portugal, D., Rocha, R.: A survey on multi-robot patrolling algorithms. In: Technological Innovation for Sustainability, pp. 139–146. Springer (2011)
24. Yehoshua, R., Agmon, N.: Adversarial modeling in the robotic coverage problem. In: International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-15), pp. 891–899 (2015)
25. Yehoshua, R., Agmon, N.: Online robotic adversarial coverage. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS-15), pp. 3830–3835 (2015)
26. Yehoshua, R., Agmon, N., Kaminka, G.A.: Robotic adversarial coverage: Introduction and preliminary results. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS-13), pp. 6000–6005 (2013)
27. Yehoshua, R., Agmon, N., Kaminka, G.A.: Frontier-based RTDP: A new approach to solving the robotic adversarial coverage problem. In: International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-15), pp. 861–869 (2015)
28. Yehoshua, R., Agmon, N., Kaminka, G.A.: Robotic adversarial coverage of known environments. International Journal of Robotics Research **35**(12), 1419–1444 (2016)

29. Zelinsky, A., Jarvis, R.A., Byrne, J., Yuta, S.: Planning paths of complete coverage of an unstructured environment by a mobile robot. In: International Conference on Advanced Robotics (ICAR-93), vol. 13, pp. 533–538 (1993)