Adversarial Modeling in the Robotic Coverage Problem

Roi Yehoshua and Noa Agmon
The SMART Group
Bar Ilan University, Israel
{yehoshr1,agmon}@cs.biu.ac.il

ABSTRACT

The general problem of covering an area is a fundamental problem in robotics with applications in various domains. In a recently introduced version of the problem, *adversarial coverage*, the covering robot operates in an environment that contains threats that might stop it. Previous studies of this problem dealt with finding optimal strategies for the coverage, that minimize both the coverage time and the probability that the robot will be stopped before completing the coverage. However, these studies assumed a simplistic adversarial model, in which the threats are randomly scattered across the environment. In this paper, we allow the adversary to choose the locations of the threats in a way that maximizes the probability of stopping the robot performing the coverage. In other words, we discuss the problem of finding the best strategy to defend a given area from being covered by an agent, using k given guards. We show that although in general finding an optimal strategy for an adversary with zero knowledge is $\mathcal{NP}$-Hard, for certain values of k an optimal strategy can be found in polynomial time, and for others we suggest heuristics that can significantly improve the random baseline strategy.

Categories and Subject Descriptors

I.2.9 [**Computing Methodologies**]: Artificial Intelligence—*Robotics*

General Terms

Algorithms, Theory

Keywords

Mobile robot coverage, adversarial coverage, adversarial modeling, motion and path planning, robotics in hazardous fields

1. INTRODUCTION

In robotic coverage, a robot is required to visit every part of a given area using the most efficient path possible ([1], [3], [4], [10], [15]). Coverage has many applications in a multitude of domains, including search and rescue, mapping, and surveillance.

Appears in: *Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2015), Bordini, Elkind, Weiss, Yolum (eds.), May, 4–8, 2015, Istanbul, Turkey.*

In a recently introduced version of the problem, *adversarial coverage* [13], the robot has to cover the given terrain without being detected or damaged 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 as quickly as possible while minimizing the probability that it will be stopped before completing the coverage. This version of the problem has many real-world applications in various domains, 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 problem ([13], [14]), 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, i.e., we are examining the opposite problem of adversarial coverage. In the eyes of the adversary, the threat points are considered as guards that protect its territory from being covered by an intruder. Here we assume that the guards are stationary, and they can catch the robot only if it enters into their positions.

We examine the impact of the adversarial knowledge of the coverage path on the choice of the guards' locations, and provide solutions for adversaries having no knowledge and full knowledge of the coverage path. We show that for a full-knowledge adversary there is a simple algorithm that provides the optimal strategy, whereas finding an optimal strategy for a zero-knowledge adversary is $\mathcal{NP}$-Hard. Nevertheless, we propose an algorithm for a zero-knowledge adversary that outperforms the baseline random strategy, and in some cases can provide the optimal solution. We provide both theoretical and empirical evaluation of these algorithms. Finally, we discuss some cases in which the adversary has partial knowledge of the coverage path (for example, when it only knows where the coverage begins).

2. RELATED WORK

The problem of robot coverage has been extensively discussed in the literature (see [4] for a recent exhaustive survey). The adversarial coverage problem, in which the environment contains threats that may stop the robot, was formally defined in a recent study [13]. There the authors proposed a simple heuristic algorithm that generates a coverage path which tries to minimize a cost function, that takes into account both the survivability of the robot and the cov-

erage path length. In a follow-up paper [14] they have addressed a more specific version of the problem, namely, finding the safest coverage path. They suggested two heuristic algorithms to solve this problem with some theoretical guarantees: GAC, which follows a greedy approach and STAC, a spanning-tree based coverage algorithm. However, these studies assumed a simplistic adversarial model, in which the threats are randomly scattered across the environment.

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 [11] 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 1.

Another related problem is the art gallery problem or museum problem, in which one has to determine the minimum number of guards who together can observe the whole gallery. This problem is known to be $\mathcal{NP}$-Hard [9]. In contrast, here the guards should not only observe the robot but also prevent it from completing its coverage mission, and they can do so only when the robot enters the same position where one of the guards is located. In addition, here we are not interested in finding the minimum number of guards, but in finding the best possible locations of k given guards in order to maximize the probability of stopping the robot.

Pursuit-evasion is another family of problems in which one group (the pursuers) attempts to track down members of another group (the evaders) in an environment. Borie et al. [2] discuss the complexity of several pursuit-evasion variants, namely how many pursuers are needed to clear a given graph (i.e., capture all the evaders) and how a given number of pursuers should move on the graph to clear it with either a minimum sum of their travel distances or minimum task-completion time. In contrast to the problem discussed here, the evaders are not trying to visit all the points in the area, and also they are facing a mobile agent.

3. 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.

The objective of the robot that covers the target area is to find a path in G of minimum length that visits every vertex of V at least once. Except for the simplest environments (e.g., without obstacles), the coverage mission cannot be completed without repeated visits of some of the vertices. In graph theory, this problem is also known as finding Hamiltonian walks in graphs [6], which is a generalization of the Hamiltonian cycle problem and thus it is $\mathcal{NP}$-Complete.

A typical representation of the environment used in the robotic coverage literature is that of a grid map (e.g., [3], [10], [15]). 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. It is also usually assumed that the robot can move only in the four basic directions (up/down, left/right), but not diagonally. Thus, the grid graph G is 4-connected.

To make our analysis as general as possible, we will assume a generic graph representation of the environment in the theoretical analysis part. However, in the evaluation part we will use grid maps as the typical representation.

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 [3]), while others do not hold this assumption (e.g., the wavefront algorithm [15]). We will relate to both cases when we explain how to compute the adversarial plan.

3.1 Adversarial Model

The objective of the adversary is to defend its territory from being covered by the robot. The adversary can place $k < n$ guards at different locations of the area. Each guard has a probability $0 < p \leq 1$ of stopping the robot, when the robot enters its location.

The goal of the adversary is to assign the k guards to k locations in the environment such that the probability of stopping the covering robot is maximized. Let us formally define this objective function. First, we denote the coverage path followed by the robot by $A = (v_1, v_2, ..., v_m)$, where $v_1, ..., v_m$ 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 might have to repeat its steps in order to complete the coverage. Let us denote the probability that the robot is stopped at vertex v_i by p_i. $p_i = p$ when v_i has a guard assigned to it, otherwise $p_i = 0$. Now, we define the event S_A as the event that the robot is able to complete its coverage path A, without being stopped by any of the guards. The probability of S_A is:

$$P(S_A) = \prod_{i=1}^{m}(1 - p_i) \quad (1)$$

Thus, the probability that the robot is stopped before completing its coverage mission is $1 - P(S_A)$.

We now distinguish between adversaries having no knowledge and full knowledge of the coverage path. A full-knowledge adversary 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 adversary 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 each point in the target area the least possible number of times. Hence, the objective of a zero-knowledge adversary 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. We also provide a discussion of cases in which the adversary has some knowledge (between full and zero knowledge) of the covering strategy of the robot, for example when it knows only the starting location of the coverage path.

3.2 Full-Knowledge Adversary

For a full-knowledge adversary, there is a simple algorithm with linear run-time that generates an optimal adversarial 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.

3.3 Zero-Knowledge Adversary

For a zero-knowledge adversary, the problem of finding an

Algorithm 1 Place_Guards(G, A, k)

input: $G = (V, E)$ - the graph representing the environment, A - the coverage path, k - the number of guards

1: Compute for every vertex in V the number of times it is visited by A
2: Run a selection algorithm to find the first k vertices with the highest number of visits (break ties randomly)
3: Place guards at these k vertices

optimal strategy becomes $\mathcal{NP}$-Hard, as proven by the next theorem.

Definition 1. *Zero-Knowledge Adversarial Coverage Problem (ZKACP): 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 a robot covering the environment is maximized, assuming that the robot is following an optimal coverage strategy.*

Theorem 1. *The ZKACP problem is $\mathcal{NP}$-Hard.* [1]

Proof. (Sketch). 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 [5]. Given an instance of the Hamiltonian path problem on a graph $G = (V, E)$, we construct an instance of the zero-knowledge adversarial coverage problem on the same graph G with $k = 1$. There exists a Hamiltonian path in G, if and only if the optimal strategy for the adversary is to place its single guard at a random vertex of G. This is because when there is a Hamiltonian path, the optimal coverage path visits every vertex in the graph only once, thus there is no difference at which one the guard is placed. However, when there is no such path, some of the vertices must be visited more than once by the optimal coverage path, and thus one of them must be chosen for the guard's location by the optimal adversarial strategy. □

4. COMPUTING ADVERSARIAL PLAN

In this section we describe a strategy for placing k guards in the environment for a zero-knowledge adversary. The strategy we propose is optimal for certain values of k, and for others it offers a significant improvement over the random baseline strategy. The strategy consists of three main steps:

1. Place guards at vertices that must be visited more than once, giving precedence to vertices that must be visited more frequently.
2. Place guards at groups of vertices, in which some of the vertices must be visited more than once, 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.

4.1 Vertices that must be visited more than once

We begin by characterizing the vertices in G that the covering robot must visit more than once. For that purpose, we will use the following definitions from graph theory [7].

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.*

[1]Full proofs can be found in [12].

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

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

The block decomposition theorem [7] states that any connected graph decomposes into a tree of biconnected components called the *block tree* of the graph. In this tree the blocks are attached to each other at the articulation points of the graph, which are the only shared vertices between different blocks. The block tree of a graph is unique.

Figure 1 shows an example for a block decomposition 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 $c_1 = 2, c_2 = 7, c_3 = 8, c_4 = 10$.

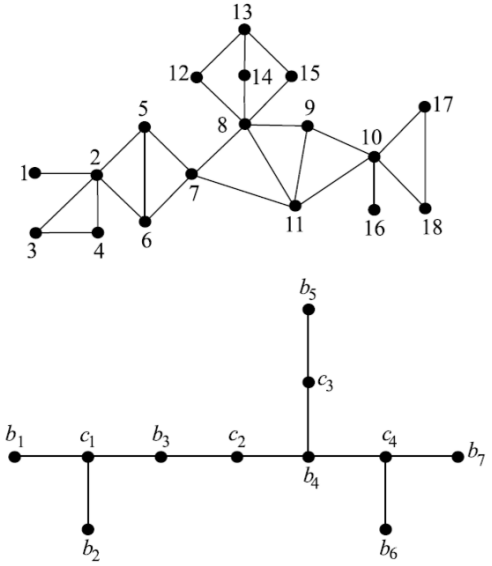

Figure 1: A graph (upper) and its block tree (lower).

We will add the following two definitions.

Definition 5. ***Connectivity**(v) is the number of connected components the graph would split into if v is removed from the graph.*

Definition 6. *A **k-connected articulation point** is an articulation point whose connectivity is k.*

We start by analyzing coverage paths that must return to their starting vertex.

Coverage paths that return to their starting vertex

The next theorem provides a lower bound on the number of times each articulation point in G must be visited along any coverage path that returns to its starting vertex.

Theorem 2. *Any coverage path that returns to its starting vertex must visit every k-connected articulation point at least k times. In addition, if the starting vertex of the coverage path, s, is a k-connected articulation point then it 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 visit of these components is $C_1, ..., C_k$ (the same component may appear more than once in the sequence). Consider two cases:

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

Case 2. $v = s$. In this case s does not belong to any of the connected components C_i. Thus, the coverage path has the following structure: $p = s \rightsquigarrow C_1 \rightsquigarrow s \rightsquigarrow ... \rightsquigarrow C_k \rightsquigarrow s$. Hence, the coverage path must visit s at least $k+1$ times. □

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 [8]. The algorithm is based on Depth-First Search (DFS). The idea is that in a DFS tree of an undirected graph, a node v is an articulation point, if and only if for every child u of v there is no back edge from u to a node higher in the DFS tree than v, i.e., every node in the descendant subtree of v has no way to visit other nodes in the graph without passing through v. The algorithm uses the following definitions.

Definition 7. ***DfsNum(v)*** *is the DFS visit number of* v.

Definition 8. ***DfsLevel(v)*** *is the DFS tree level of* v.

Definition 9. ***Low(v)*** *is the lowest-numbered vertex reachable from* v *using* 0 *or more tree edges and then at most one back edge.*

The algorithm 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.
2. Any other vertex v is an articulation point if and only if v has some child w such that $\text{Low}(w) \geq \text{DfsNum}(v)$, i.e., there is a child w of v that cannot reach a vertex visited before v.

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 the biconnected component are all on top of the stack. The helper procedure Create_Biconnected_Component is then used to pop the edges of this block from the stack.

We have extended the classical algorithm with the following additions:

1. We compute the connectivity of each articulation point (lines 15, 22 and 24 in the algorithm).
2. We maintain for each articulation point a list of all the blocks that are attached to it in the block tree (lines 17 and 26). This extension will be useful when we handle the case of the robot not having to return to its starting point.

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 [8]. We also prove that the connectivity of each articulation point is computed correctly by the algorithm.

Theorem 3. *(correctness) Algorithm 2 computes correctly the connectivity of each articulation point.*

The runtime complexity of the original algorithm for finding biconnected components 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.

Algorithm 2 Find_Articulation_Points(v)

input: v - the current vertex
Global data structures: S - the stack of visited edges, *ArticulationPointList* - the list of articulation points
Global initialization: dfsCounter = 0, s.dfsLevel = 0, where s is the starting vertex.

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

```
1: procedure Create_Biconnected_Component(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 biconnected component C
3:     while Top(S) ≠ (v, w) do
4:                 ▷ Retrieve all edges in the component
5:         (u1, u2) ← Pop(S)
6:         Add (u1, u2) to C
7:     Add Pop(S) to C                          ▷ Add (v, w) to C
8:     return C
```

Given the articulation points of a graph G and their connectivity, we now describe a strategy for placing k guards at vertices in G (algorithm 3). The idea is to place the guards at the k articulation points with highest connectivity. If there are fewer than k articulation points in the graph, the positions of the remaining guards will be determined by exploiting additional features of the graph (see section 4.2). Note that by theorem 2 if the starting vertex s is an articulation point with connectivity c, then it must be visited at least once more than the other articulation points with the same connectivity. In this case the algorithm treats s as if it was an articulation point with connectivity $c + 1$.

Since there are worst-case linear time selection algorithms, the runtime complexity of algorithm 3 is $O(|V| + |E|)$, i.e.,

Algorithm 3 Place_Guards_At_Articulation_Points(s, k)

input: s - the starting vertex of the coverage, k - the number of guards

1: $ArticulationPointList \leftarrow$ Find_Articulation_Points(s)
2: $a \leftarrow$ the number of articulation points
3: $m \leftarrow min(k, a)$
4: **if** s is an articulation point **then**
5: Add 1 to $s.connectivity$
6: Run a selection algorithm to find the first m articulation points with the highest connectivity (break ties randomly)
7: Place guards at these m articulation points

linear in the size of the graph.

We now show that for certain values of k, the adversarial strategy depicted in algorithm 3 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 lemma, 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.

Lemma 1. *Denote the degree of a vertex v by $deg(v)$. An optimal coverage path of a graph $G = (V, E)$ that returns to its starting vertex visits every vertex $v \in V$ at most $deg(v)$ times, except for the starting vertex s that is visited at most $deg(s) + 1$ times.*

Theorem 4. *Let the maximum degree in a 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 adversarial strategy described in algorithm 3 is optimal.*

Proof. By lemma 1, an optimal coverage path that returns to its starting vertex visits every vertex $v \in V$ at most d times, or $d + 1$ times if v is the starting vertex. By theorem 2, any such coverage path must visit every d-connected articulation point at least d times, or $d+1$ times if it is the starting vertex. Thus, the optimal coverage path visits every d-connected articulation point precisely d times if it is not the starting vertex, or $d + 1$ times if it is the starting vertex. 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 is guaranteed to maximize the probability of stopping a robot that follows this path. □

We now discuss a case in which the adversary has some knowledge of the coverage path, namely, it knows where the coverage begins. The following theorem shows that changing the starting vertex of the coverage may affect the placement of only one guard.

Theorem 5. *The strategy for placing k guards as given by algorithm 3 is not affected by changing the starting vertex of the coverage, except for maybe a placement of one guard.*

Proof. The block decomposition of a graph is unique. Thus, the number of articulation points in the graph and their connectivity are not affected by the selection of the starting vertex of the coverage. According to algorithm 3, if the starting vertex is an articulation point, then it gets precedence over the other articulation points with the same connectivity, since it must be visited once more. Therefore, changing the starting vertex of the coverage may affect the placement of only the guard at the starting vertex. □

Coverage paths that do not return to the starting vertex

We now handle the case in which the robot does not have to return to its starting point when the coverage ends. Similarly to the previous case, the adversary can take advantage of the articulation points in the graph as locations that must be visited multiple times by the covering robot. However, in this case, the number of minimum visits in each articulation point may depend on the coverage path itself.

The next theorem provides a lower bound on the number of times each articulation point in G must be visited along any coverage path. It uses the term **terminating subpath** of the coverage path, defined as a simple subpath (with no repeating vertices) of the coverage path that ends at the terminating vertex of the coverage path.

Theorem 6. *Any coverage path must visit every k-connected articulation point at least k times, except for maybe articulation points on the terminating subpath of the coverage path, which must be visited at least $k - 1$ times. In addition, if the starting vertex of the coverage path is a k-connected articulation point, then it must be visited at least $k + 1$ times, unless it is part of the terminating subpath of the coverage, in which case it is visited at least k times.*

According to theorem 6, if the adversary has no information on the coverage path, the best it can do is to use the same strategy for placing the guards as in algorithm 3, since any articulation point can reside on the terminating subpath of some coverage path. However, if the adversary has some knowledge of the coverage path, particularly if it knows where the coverage begins, a better strategy for choosing the articulation points can be devised.

Algorithm 4 describes a strategy for choosing the locations of k guards, assuming that the adversary knows the starting location of the coverage. The algorithm uses the block tree of the graph, rooted at the starting vertex of the coverage path. The main observation is that the terminating subpath of the coverage path belongs to a specific branch of the tree, since moving between branches of the tree requires visiting one of the articulation points more than once. Since the articulation points on the terminating subpath may be visited one time fewer than the other articulation points in the graph (by theorem 6), we assume that the covering robot will prefer to finish its coverage path in the branch that contains the maximum number of articulation points with guards. Hence, the optimal strategy for the adversary is to spread the guards evenly across the different branches of the block tree, such that the number of articulation points with guards in the branch with the maximum number of such articulation points is minimized.

First, the helper procedure Create_Articulation_Points_Tree 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. The child nodes of an articulation point v in T are defined as the articulation points contained in all the attached blocks of v.

Going from the highest connectivity level of articulation points in T to the lowest level, the algorithm calls the procedure Place_Guards to place guards at articulation points with a given connectivity level. This procedure uses a DFS scan of T in order to place one guard at every branch of the tree, starting from the lowest vertices of each branch and going upwards in every new scan of the tree until reaching the root. The procedure is continuously invoked, until guards

are assigned to all the articulation points within the current connectivity level.

In order to decide when a branch of the tree cannot be used anymore to place guards, we maintain two fields for each vertex in the tree:

- $v.containsGuard$ - a flag indicating whether a guard has been placed at v
- $v.allDescendantsContainGuards$ - a flag indicating whether all the descendant vertices of v in T with the same connectivity as v contain guards. For vertices with no descendants of the same connectivity, this flag is also set to true.

When a guard is assigned to an articulation point v, the field $v.containsGuard$ is updated and also the field $allDescendantsContainGuards$ of its ancestors in the tree is updated using the procedure Update_Vertex. The complete pseudocode of the helper procedures Create_Articulation_Points_Tree and Update_Vertex can be found in [12].

Algorithm 4 Place_Guards_At_Articulation_Points2(s, k)

input: s - the starting vertex of the coverage, k - the number of guards

```
1: ArticulationPointList ← Find_Articulation_Points(s)
2: T ← Create_Articulation_Points_Tree(s)
3: c_min ← the minimum connectivity of any articulation point
4: c_max ← the maximum connectivity of any articulation point
5: g ← 0                                          ▷ guards placed so far
6: v ← T.root
7: for c = c_max downto c_min do
8:     artPointsLeft = true    ▷ is there any articulation point with connectivity c left without a guard
9:     while artPointsLeft do
10:        if Place_Guards(v, c, k) then exit
11:        if v.allDescendantsContainGuards and (v.containsGuard or v.connectivity ≠ c) then
12:            artPointsLeft ← false
```

```
1: procedure Place_Guards(v, c, k)
   input: v - a vertex in the articulation points tree, c - connectivity level, k - number of guards to place
   output: A flag that indicates whether there are any more guards to place
   globals: g - number of guards placed so far
2:     if v.allDescendantsContainGuards then
3:         if not v.containsGuard and v.connectivity = c then
4:             Place a guard at v
5:             v.containsGuard ← true
6:             g ← g + 1
7:             if v.parent ≠ null then
8:                 Update_Vertex(v.parent)    ▷ update the flag allDescendantsContainGuards in v's ancestors
9:             if g = k then return true
10:    else
11:        for each w in v.childNodes do
12:            if Place_Guards(w, c, k) then return true
13:    return false
```

Lemma 2. *The runtime complexity of algorithm 4 on a graph $G = (V, E)$ is $O(|V|^2)$.*

4.2 Groups of vertices in which some of the vertices must be visited more than once

When the number of guards is greater than the number of articulation points in the graph, we need to find additional locations in the environment that must be visited by the covering robot more than once. Another potential group of such locations is the vertex cuts of the graph, defined as follows.

Definition 10. *A **vertex cut** in a connected graph G is a set of vertices whose removal renders G disconnected.*

An articulation point is a private case of a vertex cut which contains only one vertex. As for articulation points, we define the connectivity of a vertex cut as the number of connected components the graph splits into when the vertex cut is removed from the graph. We also define a k-connected vertex cut as a vertex cut with connectivity k.

The next two theorems provide a lower bound on the number of times each vertex cut in the graph must be visited along the coverage path. These are generalizations of theorems 2 and 6.

Theorem 7. *Let U be a k-connected vertex cut. Then any coverage path that returns to its starting vertex must visit vertices in U at least k times. In addition, if the starting vertex of the coverage path belongs to U, then the vertices in U must be visited at least $k + 1$ times.*

Theorem 8. *Let U be a k-connected vertex cut. Then any coverage path must visit vertices in U at least $k-1$ times. In addition, if the starting vertex of the coverage path belongs to U, then the vertices in U must be visited at least k times.*

From theorem 8, we can conclude that a k-connected vertex cut with $m < k$ vertices will have at least $k - m - 1$ repetitive visits along any coverage path. However, the theorem does not specify which of the vertices in the vertex cut will be visited multiple times.

Algorithm 5 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 theorem 8 smaller vertex cuts have potentially more revisits (ideally, we would have to first find all the vertex cuts that satisfy the conditions of theorem 8, 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. This heuristic has provided better results in practice than just choosing randomly between the vertex cuts.

Algorithm 5 uses procedure Find_Vertex_Cuts to find all the vertex cuts of size k in the graph. Since we are interested only in finding vertex cuts that do not contain any proper subset of vertices which is also a vertex cut, the search for vertex cuts can be limited to the biconnected components of the graph. This is proven by the next theorem.

Theorem 9. *Any vertex cut that does not contain a proper subset of vertices, which is also a vertex cut, belongs entirely to one biconnected component.*

We now discuss the runtime complexity of algorithm 5.

Lemma 3. *The runtime complexity of Find_Vertex_Cuts is $O\left(\binom{|V|}{k} \cdot (|V| + |E|)\right)$, where k is the given vertex cut size.*

Note that for a constant k (which is not dependant on $|V|$), the runtime complexity of the procedure is polynomial in the graph size. On the other hand, if k depends on $|V|$

Algorithm 5 Place_Guards_At_Vertex_Cuts(G, L_B, g)

input: G - the graph representing the environment, L_B - the list of blocks, g - the number of guards to place

```
1: n ← 0                                   ▷ the number of guards placed so far
2: k ← 2                                   ▷ the current cut size
3: while n < g do
4:     L_C ← Find_Vertex_Cuts(G, L_B, k)
5:     Sort the vertex cuts in L_C first by their connectivity and
   then by their spread (both in descending order)
6:     for each vertex cut C ∈ L_C do
7:         for each vertex v ∈ C do
8:             Place a guard in v
9:             n ← n + 1
10:            if n = g then exit
11:    k ← k + 1
```

```
1: procedure Find_Vertex_Cuts(G, L, k)
   input: G - the graph representing the environment, L_B - the
   list of blocks, k - the size of the vertex cut
2:     Create a new list of vertex cuts C
3:     for every block B ∈ L_B do
4:         G_B ← subgraph of G induced by the nodes in B
5:         F ← list of nodes in B that don't contain guards
6:         for every subset S of k nodes in F do
7:             Find the connected components in G_B − S by run-
   ning DFS
8:             n ← the number of connected components
9:             if n > 1 then
10:                Add the subset S to C
11:                S.connectivity ← n
12:    return C
```

(for example, if $k = \frac{|V|}{2}$), then the runtime is exponential.

Theorem 10. *The runtime complexity of algorithm 5 is* $O\big(2^{|V|} \cdot (|V| + |E|)\big)$.

Although the upper bound on the runtime complexity is exponential in the number of nodes, in practice using vertex cuts of sizes 2 and 3 was enough to place all the remaining guards, thus it was never needed to compute vertex cuts of size more than $k = 3$.

5. EMPIRICAL EVALUATION

In this section we evaluate the following four adversarial strategies:

1. Strategy level 0 is the baseline strategy, in which the adversary randomly chooses the positions of the guards.
2. Strategy level 1, in which the adversary uses only the articulation points to place guards according to algorithm 3. If there are more guards to place than articulation points, then the positions of the remaining guards are randomly chosen.
3. Strategy level 2, in which the adversary uses only the articulation points to place guards, but chooses them in a more clever way (by running algorithm 4), assuming that it knows the starting location of the coverage. As in strategy level 1, if there are more guards to place than articulation points, then the positions of the remaining guards are randomly chosen.
4. Strategy level 3, in which the adversary uses both the articulation points and the vertex cuts of the graph in order to place guards, by running algorithms 4 and 5.

In the experiments we assume that the coverage path does not have to return to its starting location.

Note that when the number of guards is greater than the number of articulation points in the grid's graph, there is no difference between the first and the second level strategies, since they both use all the articulation points for placing guards. On the other hand, when the number of guards is lower than or equal to the number of the articulation points, there is no difference between the second and the third level strategies, since there is no need to use the vertex cuts in the graph.

We first use specific maps to illustrate the operations of the different strategies and then we also report on the statistical analysis of its behavior based on multiple randomly generated maps with varying parameters. For the coverage strategy of the robot we have used GAC (Greedy Adversarial Coverage), the state-of-the-art solution to the adversarial coverage problem, as described in [14].

Figure 2 demonstrates the effectiveness of algorithm 4 in choosing the articulation points at which to place guards, as compared to algorithm 3. The sample map consists of 10×10 square cells, 30% of which contain obstacles. The locations of the obstacles are randomly chosen. There are 20 guards placed on the map. Each guard has a probability of 3% of stopping the robot. Changing this probability does not affect the results of the strategies - it just modifies the overall probability of stopping the robot. In all experiments the starting position of the robot was set to cell (1,1).

There are 26 articulation points in the graph representing this map, thus it is enough to use the articulation points to place all the 20 guards on this map. Comparing the two maps shows that algorithm 4 spreads the guards more evenly across the different branches of the blocks tree. Additionally, it prefers to place guards at articulation points that are located deeper in the blocks tree than at those located near the beginning of the coverage path. Running GAC on the map generated by the first level strategy creates a coverage path that visits cells with guards 37 times and its probability to complete is 32.4%. On the other hand, running GAC on the map generated by the second level strategy creates a coverage path that visits cells with guards 48 times and its probability to complete is 23.18%.

Figure 3 shows the result of running algorithm 5 on the same map. In this case, we increased the number of guards from 20 to 30, thus four of the guards had to be placed at the vertex cuts of the graph. The algorithm has chosen two vertex cuts of size 2 in which to place the guards. The first one consists of cells (7,2) and (8,1). This is the only vertex cut that splits the graph into 3 connected components. When GAC is run on this map it visits one of these cells twice (cell (7,2)). The second vertex cut consists of cells (7,4) and (10,1). This vertex cut splits the graph into only two connected components. The distance between its vertices (6) is the largest amongst the vertex cuts with connectivity 2. Indeed, GAC had to visit one of its cells twice (cell (7,4)).

Next, we have compared the different strategies on 50 randomized maps of size 20×20. 30% of the cells in the map contain obstacles and the other cells are free. The number of guards that have been placed on the map varied between 0 and 100. Each guard has a probability of 1% of stopping the robot (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 4 shows the probability that the robot will be stopped along its coverage path, when the adversary is using one of the four mentioned strategies.

As anticipated, for all strategies the probability of stop-

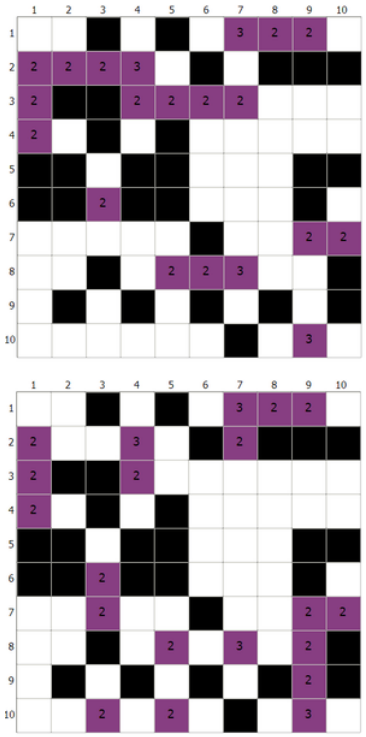

Figure 2: An example map with the results of strategy level 1 (upper figure) and strategy level 2 (lower figure). Cells containing guards are colored with purple. The numbers in the cells indicate the connectivity of the articulation points.

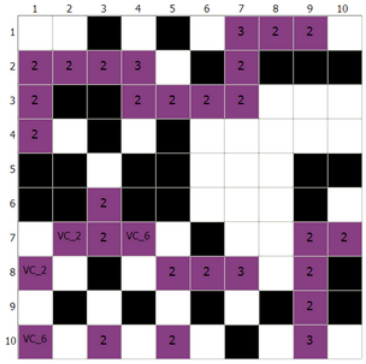

Figure 3: The results of strategy level 3 on the sample map. Cells labeled with VC and a number belong to the vertex cuts of the graph. The number in the label indicates the spread of the cut.

ping the robot increases as we add more guards to the map. We can also see that higher-level strategies dominate the lower-level strategies. The differences between the results of the strategies are statistically significant. A one-tailed t-test between strategies 0 and 1 yields $p = 3.67 \cdot 10^{-51}$, between strategies 1 and 2 it yields $p = 9.3 \cdot 10^{-13}$, and between strategies 2 and 3 it yields $p = 5.83 \cdot 10^{-19}$.

On average, the third-level strategy causes the robot to visit 31 more cells with guards than the baseline random strategy, and by that to increase the probability of stopping

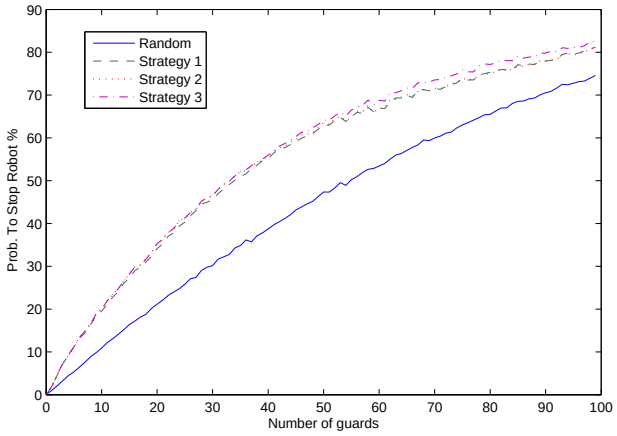

Figure 4: The probability of stopping the covering robot for different numbers of guards.

the robot by 12.89%.

The first- and second-level strategies differ only when the number of guards is below 35. As the number of guards increases from 0 to 35, the gap between these strategies widens, since there is a greater choice of which articulation points to choose. For 35 guards, the second-level strategy, which chooses the articulation points in a more clever way, causes the robot to visit 2 more cells with guards (on average), and thus increases the probability of stopping it by 1%.

On the other hand, the second- and third-level strategies differ only when the number of guards is above 35. The gap between these strategies increases as we add more guards. For 100 guards, the third-level strategy, that also uses the vertex cuts to place guards, causes the robot to visit 9 more cells with guards (on average), and thus increases the probability to catch it by 1.7%.

6. CONCLUSIONS AND FUTURE WORK

We have described how to model an adversary in the robotic coverage problem, and examined the impact of the adversarial knowledge on the model. By analyzing the graph representing the target area for the coverage, we have shown how the adversary can take advantage of the vulnerability points in the environment (i.e., those that must be visited by the covering robot more than once), in order to increase the probability of stopping the covering robot. Although finding an optimal strategy for a zero-knowledge adversary is generally hard, when the number of guards that can be used by the adversary to protect its territory is limited to certain values, such an optimal strategy can be found in polynomial time. For other cases, we have suggested how to exploit additional features of the representative graph in order to devise an adversarial strategy that outperforms the random baseline strategy. Finally, we have evaluated different levels of adversarial strategies in an extensive set of experiments.

In the future, we intend to use the knowledge gained from this research in order to develop new coverage strategies that take into account the model of the adversary. We also plan to investigate adversarial strategies that can deal with a multi-robot team that covers the given area.

7. REFERENCES

[1] E. M. Arkin, S. P. Fekete, and J. S. Mitchell. Approximation algorithms for lawn mowing and milling. *Computational Geometry*, 17(1):25–50, 2000.

[2] R. Borie, C. Tovey, and S. Koenig. Algorithms and complexity results for graph-based pursuit evasion. *Autonomous Robots*, 31(4):317–332, 2011.

[3] Y. Gabriely and E. Rimon. Competitive on-line coverage of grid environments by a mobile robot. *Computational Geometry*, 24(3):197–224, 2003.

[4] E. Galceran and M. Carreras. A survey on coverage path planning for robotics. *Robotics and Autonomous Systems*, 61(12):1258–1276, 2013.

[5] M. R. Garey and D. S. Johnson. Computers and intractability: a guide to the theory of NP-completeness. *Freeman, San Francisco*, 1979.

[6] S. Goodman and S. T. Hedetniemi. On Hamiltonian walks in graphs. *SIAM Journal on Computing*, 3(3):214–221, 1974.

[7] F. Harary. Graph theory, 1969.

[8] J. E. Hopcroft and R. E. Tarjan. Efficient algorithms for graph manipulation. *Communications of the ACM*, 16(6):372–378, 1973.

[9] D. T. Lee and A. Lin. Computational complexity of art gallery problems. *IEEE Transactions on Information Theory*, 32(2):276–282, 1986.

[10] C. Luo, S. X. Yang, D. A. Stacey, and J. C. Jofriet. A solution to vicinity problem of obstacles in complete coverage path planning. In *Proc. IEEE International Conference on Robotics and Automation (ICRA-02)*, volume 1, pages 612–617, 2002.

[11] D. Portugal and R. Rocha. A survey on multi-robot patrolling algorithms. In *Technological Innovation for Sustainability*, pages 139–146. Springer, 2011.

[12] R. Yehoshua and N. Agmon. Adversarial modeling in the robotic coverage problem: Proofs and algorithm details. Technical Report SMART 2015/01, available at http://www.cs.biu.ac.il/~yehoshr1/, Bar Ilan University, Computer Science Department, SMART Group, 2015.

[13] R. Yehoshua, N. Agmon, and G. A. Kaminka. Robotic adversarial coverage: Introduction and preliminary results. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS-13)*, pages 6000–6005, 2013.

[14] R. Yehoshua, N. Agmon, and G. A. Kaminka. Safest path adversarial coverage. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS-14)*, pages 3027–3032, 2014.

[15] A. Zelinsky, R. A. Jarvis, J. Byrne, and S. Yuta. Planning paths of complete coverage of an unstructured environment by a mobile robot. In *Proceedings of international conference on advanced robotics*, volume 13, pages 533–538, 1993.