The International Journal of Robotics Research
0(0):1–39
©The Author(s) 2015
Reprints and permission:
sagepub.co.uk/journalsPermissions.nav
DOI:doi number
http://mms.sagepub.com

Robotic Adversarial Coverage of Known Environments

Roi Yehoshua,[*] Noa Agmon and Gal A. Kaminka
Computer Science Department, Bar Ilan University, Israel

Abstract
Coverage is a fundamental problem in robotics, where one or more robots are required to visit each point in a target area at least once. Most previous work concentrated on finding a coverage path that minimizes the coverage time. In this paper we consider a new and more general version of the problem: *adversarial coverage*. Here, the robot operates in an environment that contains threats that might stop the robot. The objective is to cover the target area as quickly as possible, while minimizing the probability that the robot will be stopped before completing the coverage. This version of the problem has many real-world applications, from performing coverage missions in hazardous fields such as nuclear power plants, to surveillance of enemy forces in the battlefield and field demining. In this paper we discuss the offline version of adversarial coverage, in which the map of threats is given to the robot in advance. First, we formally define the adversarial coverage problem, and present different optimization criteria used for evaluation of coverage algorithms in adversarial environments. We show that finding an optimal solution to the adversarial coverage problem is NP-Hard. We therefore suggest two heuristic algorithms: STAC, a spanning-tree based coverage algorithm, and GAC, which follows a greedy approach. We establish theoretical bounds on the total risk involved in the coverage paths created by these algorithms and on their lengths. Lastly, we compare the effectiveness of these two algorithms in various types of environments and settings.

1. Introduction

Area coverage is an important task for mobile robots, with many real-world applications in various domains, from automatic floor cleaning (Colegrave and Branch, 1994) and coating in supermarkets (Endres et al., 1998) and train stations (Yaguchi, 1996), to humanitarian missions such as search and rescue and field demining (Nicoud and Habib, 1995). In these, a robot is given a bounded work-area, possibly containing obstacles, and is required to visit every part of it as efficiently as possible. All previous studies of the coverage problem dealt with non-adversarial settings, where nothing in the environment is hindering the robot's task. However, in many occasions, robots and autonomous agents need to perform coverage missions in hazardous environments, such as operations in nuclear power plants, planetary exploration, demining, and surveillance of enemy forces in the battlefield.

[*] Corresponding author; e-mail: yehoshr1@cs.biu.ac.il

Hence, our work addresses the problem of *adversarial coverage*, the task of visiting every point in a given terrain without being detected or damaged by an adversary. Each point in the area, which is not occupied by an obstacle, 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 maximizing its own safety. This is a proper generalization of non-adversarial coverage, which is the same as adversarial coverage in environments with no adversarial presence, or when the objective takes into account only the coverage time and ignores the risk factor.

In this paper we discuss the offline version of this problem, in which the map of threats is given in advance, therefore the coverage path of the robot can be determined prior to its movement. We formally define the offline adversarial coverage problem and show that it is $\mathcal{NP}$-complete. We then propose two algorithms for solving it heuristically in polynomial time. The STAC (Spanning-Tree Adversarial Coverage) algorithm splits the target area into connected areas of safe and dangerous cells, and then it covers the safe areas before moving to the dangerous ones. The GAC (Greedy Adversarial Coverage) algorithm follows a greedy approach, which leads the robot from its current location to the nearest safest location which has not been covered yet, and show empirically, in simulations, that while STAC tends to achieve higher expected coverage, GAC produces coverage paths with lower accumulated risk. We provide optimality bounds on both algorithms, and show that STAC generates coverage paths which are nearly-optimal with respect to the expected coverage percentage, and are also close to optimal with respect to the probability of completing the coverage in environments that consist of contiguous areas of threats.

The paper is organized as follows. Section 2 presents background and related work. Section 3 defines the adversarial coverage problem and discusses its complexity. Sections 4 and 5 present the STAC and GAC algorithms, respectively, and analyze them theoretically. Section 6 presents experimental results evaluating these algorithms. Section 7 concludes with a summary and brief overview of future work. For conciseness and better readability, some of the proofs are deferred to appendices.

2. Background and Related Work

The problem of single and multi-robot coverage has been extensively discussed in the literature and many approaches to coverage path planning have been developed. Galceran and Carreras (Galceran and Carreras, 2013) provide a recent exhaustive survey of coverage path planning (CPP) methods, which are classified according to various criteria.

First, most coverage algorithms decompose the target space in subregions (called cells) to achieve coverage, thus coverage algorithms can be classified according to the type of decomposition used. The survey (Galceran and Carreras, 2013) distinguishes 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 decomposed to a set of regions, whose union fills the entire area exactly. We use approximate decomposition in this paper.

Independently, coverage algorithms can be classified as either offline or online. Offline algorithms rely only on stationary information, and the map of the environment is assumed to be known in advance. Conversely, online algorithms do not assume full prior knowledge of the environment to be covered and utilize real-time sensor measurements to sweep the target space. In this paper we focus on offline coverage.

The coverage problem is analogous to the traveling salesman problem, which is $\mathcal{NP}$-complete even on simple graphs such as grid graphs (Papadimitriou, 1977). However, it is possible to find solutions to the coverage problem that are close to optimal in polynomial or even linear time through heuristics and abstractions (e.g., (Arkin et al., 2000), (Gabriely and Rimon, 2003), (Grigni et al., 1995), (Xu et al., 2011)).

Our own approach builds on earlier work in coverage; specifically, on the Spanning Tree Coverage (STC) family of algorithms, first introduced by Gabriely and Rimon (Gabriely and Rimon, 2001) for single robots, and extended to multiple robots by Hazon and Kaminka (Hazon and Kaminka, 2008). The key idea in this family of algorithms is to approximate a two-dimensional work area using a grid, such that a Hamiltonian cycle is guaranteed to exist through the grid, which can be found by generating a spanning tree in the grid graph. The original STC algorithm generates an optimal coverage path of the environment, which visits every cell in the grid exactly once. However, it treated cells which are partially occupied by obstacles as completely blocked. In a follow up paper (Gabriely and Rimon, 2003), Gabriely and Rimon have refined this algorithm to the Spiral Spanning Tree Coverage (Spiral-STC), which partially removes this limitation and provides close-to-optimal coverage paths in a uniform grid based terrain.

The idea was broadened for a multi-robot system by Hazon and Kaminka in the family of Multi Robot Spanning Tree Coverage (MSTC) algorithms. Their solution, along with decreasing the total coverage time, achieved robustness in the sense that as long as one robot works properly, the coverage of the terrain is guaranteed. They have also shown that in multi robot teams redundancy might be necessary for more efficiency. We also accept redundant coverage, but for safety.

Elmaliach et al. (Elmaliach et al., 2009) extended this representation such that a non-uniform cost is attached to a movement between two adjacent cells in the grid, and provided an algorithm that finds the cyclic path covering the terrain with minimum total cost. Conversely, in our work there is a uniform cost attached to each movement in the grid, and additionally there is a probability of survivability associated with each cell in the grid. We believe that our methods can also be easily extended to the case of non-uniform movement costs.

Zelinsky et al. (Zelinsky et al., 1993) presented the wavefront algorithm, which is another grid-based coverage method. The method requires a start cell and a goal cell. A distance transform that propagates a wave front from goal to start is used to assign a specific number to each grid element. Once the distance transform is calculated, a coverage path can be found by starting on the start cell and selecting the neighboring cell with the highest label that is unvisited. The algorithm is simple and easy to implement. However, it does not provide any guarantees on the coverage time. It also requires a definition of a goal cell, that is not needed by the previously mentioned coverage methods.

Papers in the robotic literature that take into account the presence of an adversary include (Bortoff, 2000), (Likhachev and Stentz, 2007), (Zabarankin et al., 2002). These present algorithms and methods for risk avoidance. These works examine the path planning problem of a single robot, in order to bypass and avoid the adversary's threats. In the patrol problem (Elmaliach et al., 2009), a multi-robot team needs to patrol around a closed area with the existence of an adversary attempting to penetrate into the area. The patrol problem resembles the coverage problem in the sense that both require the robot or group of robots to visit all points in the given terrain. However, while coverage seeks to minimize the number of visits to each point (ideally, visiting it only once), patrolling often seeks to maximize it (while still visiting all points).

The adversarial coverage problem discussed in this paper is unique in the sense that the adversarial nature of the environment is expressed in the possibility of physically harming the robot (rather than reducing the utility of a robotic team, as in (Hazon and Kaminka, 2008)). It has an intrinsic complexity that is not typical to the other problems mentioned, since it presents a delicate tradeoff between minimizing the accumulated risk and minimizing the total coverage time. Trying to minimize the risk involved in the coverage path means making some redundant steps, which in turn can make the coverage path longer, and thus increase the risk involved, as well as increase the coverage time.

Other optimization problems related to adversarial coverage include the Canadian Traveller Problem (CTP) (Papadimitriou and Yannakakis, 1989), in which the objective is to find the expected shortest path between two nodes in a partially-observable graph, where some edges may be non-traversable. In contrast, here the graph is fully-observable and the agent must visit every node in the graph (some of them may stop the robot).

The offline adversarial coverage problem was formally defined in our recent study (Yehoshua et al., 2013). There we proposed a simplistic heuristic algorithm that generates a coverage path which tries to minimize a cost composed of both the survivability of the robot and the coverage path length. However, the heuristic algorithm worked only for obstacle-free areas, and without any guarantees, or analysis of the problem complexity.

In a follow-up paper (Yehoshua et al., 2014) we have addressed a more specific version of the adversarial coverage problem, namely, finding the safest coverage path. There we suggested two heuristic algorithms: STAC, a spanning-tree based coverage algorithm, and GSAC, which follows a greedy approach. We have shown that these algorithms produce close to optimal solutions in polynomial time. However, the heuristic algorithms could handle only one level of threats, i.e., the environment could contain either safe or dangerous areas, in addition to obstacles.

In this paper, we make the following contributions compared to previous works:

1. We extend the heuristic algorithms to handle multiple levels of threats. We analyze the total risk and the coverage time of the paths generated by the algorithms as a function of the number of threat levels.

2. We address the general adversarial coverage problem of finding a coverage path that takes into account both the risk and the coverage time. We analyze its complexity, and show how the heuristic algorithms can be modified to handle the general case.

3. We evaluate the modified algorithms on a new variety of environment types and settings.

3. The Adversarial Coverage Problem Formulation

The adversarial coverage problem can be defined as follows. We are given a map of a target area T, which contains obstacles and also threats, which may stop the robot. We assume that T can be decomposed into a regular square grid with n cells. Let us denote this grid by G. G contains two types of cells: free cells and cells that are occupied by obstacles. Some of the free cells contain threats. In contrast to obstacles which the robot cannot go through, threat locations are places that the robot must visit, but might be stopped at. Each free cell i is associated with a threat probability p_i, which measures the likelihood that a threat in that cell will stop the robot. We define *safe* cells as cells where the threat probability is $p_i = 0$, while *dangerous* cells are those where $p_i > 0$. The robot can move continuously, in the four basic directions (North, South, East, West), and can locate itself within the work-area to within a specific cell.

Figure 1 shows an example world map of size 20×20. Obstacles are represented by black cells, safe cells are colored white, and dangerous cells are represented by 5 different shades of the same color (appears as purple in the paper's online version and gray in its printed version). Darker shades represent higher values of p_i (more dangerous areas).

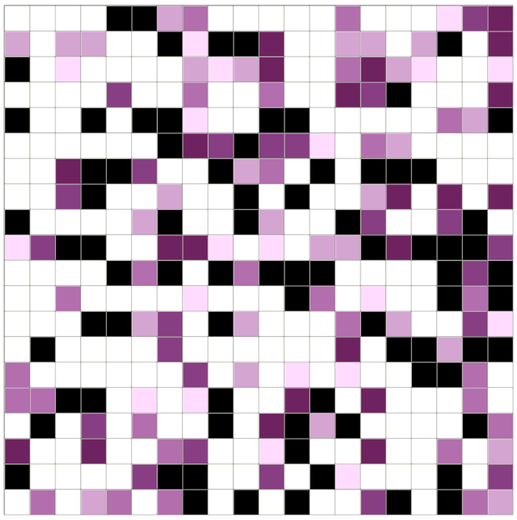

Fig. 1. An example map of the world. Darker shades represent more dangerous areas.

The survivability of the robot can be measured in two different ways, which imply two different objective functions:

1. The total accumulated risk along the coverage path (i.e., maximize the probability of covering all the cells).

2. The coverage percentage of the target area before the robot is first stopped (i.e., maximize the expected number of covered cells).

Let us now formally define these objective functions. First, we denote the coverage path followed by the robot by $A = (a_1, a_2, ..., a_m)$. Note that $m \geq n$, i.e., the number of cells in the coverage path might be greater than the number of cells in the target area, since the robot is allowed to repeat its steps. We define the event S_A as the event that the robot is not stopped when it follows the path A. The probability that the robot is able to complete this path is:

$$P(S_A) = \prod_{i \in (a_1,...,a_m)} (1 - p_i) \tag{1}$$

Thus, the first objective is to find a coverage path A that maximizes the probability $P(S_A)$. Note that in this objective, the order of visits of the cells is not important, as long as the number of visits of threat points along the coverage path is minimized (ideally, visiting each threat point only once).

For the second objective, we denote the sequence of newly discovered cells along the coverage path A by $(b_1, ..., b_n)$. Note that $b_i \neq b_j$ for each $i \neq j$, and the number of cells in this sequence is exactly the number of cells in the grid (n). For each cell in the sequence b_i, we will denote the sub-path in A that leads from the origin cell a_1 to it by g_i. Let the number of the new cells discovered by the robot until it is stopped be C_A. Then, under the threat probability function p, the expected number of new cells that the robot visits until it is stopped can be expressed as:

$$E(C_A) = \sum_{i \in (b_1,...,b_n)} \prod_{j \in g_i} (1 - p_j) \tag{2}$$

i.e., $E(C_A)$ is the sum of the probabilities to reach all the newly discovered cells along the coverage path.

Thus, the second objective is to find a coverage path A that maximizes the expected coverage $E(C_A)$. Note that in this objective, the visit order of the cells is crucial, since the robot is trying to cover as much as possible before getting hit by a threat (ideally, covering all the safe cells before visiting a single threat point).

To illustrate these definitions, let us consider the following simple grid, which is composed of 4 cells: a_{11}, a_{12}, a_{21} and a_{22}, with the probabilities for danger p_{ij} specified in each cell.

0	0.1
0.2	0.5

Assume that the initial location of the robot is in cell a_{11} (top left corner). Since there are no obstacles in this grid, there are coverage paths that visit each cell exactly once. These paths have both minimum length and maximum probability to complete. In our example there are two such coverage paths: $A_1 = (a_{11}, a_{12}, a_{22}, a_{21})$ and $A_2 = (a_{11}, a_{21}, a_{22}, a_{12})$. Their probability to complete is the same and equals to:

$$P(S_{A_1}) = P(S_{A_2}) = 1 \cdot 0.9 \cdot 0.8 \cdot 0.5 = 0.36$$

However, these paths don't have the maximum possible expected coverage. The expected number of covered cells in A_1 is:

$$\begin{aligned} E(C_{A_1}) &= 1 + 1 \cdot 0.9 + 1 \cdot 0.9 \cdot 0.5 + 1 \cdot 0.9 \cdot 0.5 \cdot 0.8 \\ &= 1 + 0.9 + 0.45 + 0.36 = 2.71 \end{aligned}$$

A similar computation shows that the expected number of covered cells in A_2 is: $E(C_{A_1}) = 2.56$.

However, the path with the maximum expected coverage is $A_3 = (a_{11}, a_{12}, a_{11}, a_{21}, a_{22})$. To compute its expected coverage, we first note that the sequence of new cells discovered along this path is $(a_{11}, a_{12}, a_{21}, a_{22})$. Thus, the expected coverage of A_3 is:

$$\begin{aligned} E(C_{A_3}) &= 1 + 1 \cdot 0.9 + 1 \cdot 0.9 \cdot 1 \cdot 0.8 \\ &+ 1 \cdot 0.9 \cdot 1 \cdot 0.8 \cdot 0.5 \\ &= 1 + 0.9 + 0.72 + 0.36 = 2.98 \end{aligned}$$

Therefore, by making one additional step, the robot is able to raise its expected number of covered cells from 2.71 to 2.98.

In the following sections, we will use the notation $\mathcal{S}(A)$ to denote the survivability of the robot following coverage path A. $\mathcal{S}(A)$ may refer to either $P(S_A)$ or $E(C_A)$, depending on the context.

The robot's task is to plan a path through T such that every accessible free cell in T (including the dangerous cells) is visited by the robot at least once. In particular, given T, three questions may be asked:

1. What is the minimum coverage time for T, and at what survivability?

2. What is the maximum survivability for T, and at what coverage time?

3. Given required levels of survivability and coverage time, what is the optimal coverage path?

In order to help us answer these questions, we will define the following weighted cost function that takes both the survivability and the coverage time factors into consideration. For a given coverage path A, define

$$f(A) = -\alpha \cdot \mathcal{S}(A) + \beta \cdot |A| \tag{3}$$

where $\alpha, \beta \geq 0$ are given up front, according to the required balance between the risk and the time factors. $|A|$ is the number of the steps the robot needs to take in order to complete the coverage path.[1] Thus, the problem is to find a coverage path A that minimizes the cost function $f(A)$, i.e., $f(A) \leq f(B)$ for all possible coverage paths B.

When $\alpha = 0$, objective (3) translates to finding a minimum time coverage path, regardless of the risk involved. Achieving this objective will provide an answer to our first question, which is equivalent to finding a solution to the general coverage problem. This means that the coverage problem becomes a special case of the adversarial coverage problem. When $\beta = 0$, the objective translates to finding a coverage path with a minimal risk, without a limit on the path length. Achieving this objective will provide an answer to our second question. Lastly, setting fixed levels for α and β will provide an answer to our third question. In the last case, the ratio α/β will determine how strongly the objective prefers safer coverage paths over shorter ones.

3.1. Problem Complexity

We now discuss the complexity of the three problems of adversarial coverage. We analyze the complexity of each of the three problems separately.

The first problem, finding a minimum time coverage path ($\alpha = 0$) is equivalent to the problem of finding Hamiltonian walks in graphs. A Hamiltonian walk of a graph G is a shortest closed walk that passes through every vertex of G at least once. The problem of finding a Hamiltonian walk of a given graph is known to be $\mathcal{NP}$-Complete (Nishizeki et al., 1983).

[1] The travel time between cells is uniform along the grid, thus coverage time is measured directly by the number of steps.

We now prove that the second problem, finding a coverage path with minimum risk ($\beta = 0$), is also $\mathcal{NP}$-hard. To prove that finding the safest coverage path is hard, we need to refer to the two different methods for measuring the survivability of the robot (equations (1) and (2)). We prove it separately for each measure.

Definition 3.1. *The Safest Coverage with Minimal Risk Problem (SCMR): Given a grid representation of a world that contains obstacles and threat points, find a coverage path of the grid with minimum total accumulated risk.*

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

Proof. Clearly, this problem is in $\mathcal{NP}$, since one can easily guess the coverage path of the robot and then verify its probability of surviving it in polynomial time. To prove its $\mathcal{NP}$-hardness, we will use a reduction from the Hamiltonian path problem on grid graphs. A *grid graph* is a finite node-induced subgraph of the infinite two-dimensional integer grid (see Figure 2 for an example of a general grid graph). The Hamiltonian path problem on grid graphs, i.e., the construction of a path that visits every node of the grid graph precisely once, has been proven to be $\mathcal{NP}$-complete in (Itai et al., 1982).

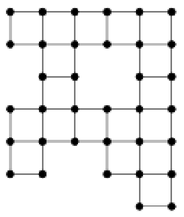

Fig. 2. An example of a general grid graph.

Given an instance of the Hamiltonian path problem on a grid graph G, we construct an instance of the safest coverage problem on a target area T as follows. For each node in G we create a cell with a threat point of probability p in T. All the other cells in T will contain obstacles. This construction can be done in polynomial time. Clearly, there exists a Hamiltonian path in G, if and only if the safest coverage path (with the minimum accumulated risk) in T visits each threat point exactly once. Therefore, we can find if there exists a Hamiltonian path in a given grid graph G, by constructing T, finding the safest coverage path in T and then checking if it covers each threat point exactly once. Thus, SCMR is $\mathcal{NP}$-hard. □

Definition 3.2. *The Safest Path with Maximum Expected Coverage Problem (SPMEC): Given a grid representation of a world that contains obstacles and threat points, find a coverage path that maximizes the expected coverage of the grid.*

Theorem 2. *The SPMEC problem is $\mathcal{NP}$-Hard.*

Proof. The same construction from the previous proof can be used to prove the $\mathcal{NP}$-Hardness of the SPMEC problem, since the target area T that was constructed there contained only threat points with uniform probabilities and obstacles (no free cells). Thus, a coverage path of T with maximum expected coverage percentage is also a coverage path with minimum accumulated risk and vice versa. □

We now prove that the third problem, finding a coverage path that meets required levels of survivability and coverage time, is $\mathcal{NP}$-Complete.

Definition 3.3. *The General Adversarial Coverage Problem (GACP): Given a grid representation of a world that contains obstacles and threat points, does there exist a coverage path whose length is $\leq L$ and its expected number of covered cells is $\geq \alpha$?*

Theorem 3. *The GACP problem is $\mathcal{NP}$-Complete.*

Proof. See Appendix A.1. □

Given the proven hardness of the problem, it is natural to look for approximate solution algorithms.

4. STAC Algorithm

The Spanning Tree Adversarial Coverage algorithm (STAC, shown in Algorithm 1) uses a layered-based approach. It first covers all the safe cells, using a minimum-risk path to move between disconnected cells (if there are any). Then it covers the dangerous areas from the least dangerous ones to the most dangerous ones, using a minimum-risk path to move between disconnected cells. This way the algorithm tries to cover as many safe cells as possible before attempting to cover any dangerous cell, and thus maximize the coverage percentage before the robot is first hit by a threat.

Algorithm 1 Spanning_Tree_Adversarial_Coverage(G, s)

```
Input: a grid $G$ and a starting cell $s$
Output: a coverage path $P$ that covers all reachable cells in $G$ from $s$
1: $P \leftarrow \emptyset$
2: Build the graph $H$ induced from the cells in $G$
3: $R \leftarrow$ all cells in $G$ reachable from $s$
4: Group the cells in $R$ into $l + 1$ threat levels, $T_0, ..., T_l$
5: for every threat level $i$, $0 \leq i \leq l$ do
6:     $D_i \leftarrow \{c | c \in T_i \wedge c \notin P\}$
7:     $P_{D_i} \leftarrow$ Create_Safe_Coverage_Path($G$, $H$, $D_i$) {Algorithm 2}
8:     if $i > 0$ then
9:         $f_i \leftarrow$ last cell in $P_{D_{i-1}}$
10:        $s_i \leftarrow$ first cell in $P_{D_i}$
11:        Run Dijkstra on $H$ from $f_i$
12:        Let $\pi_i$ be the safest route from $f_i$ to $s_i$ found by Dijkstra
13:        Add $\pi_i$ to $P$
14:    Add $P_{D_i}$ to $P$
15: return $P$
```

STAC begins by creating the graph $H = (V, E)$ induced from the grid cells (line 2). In this graph, each cell in G is represented by a vertex in V and vertices that represent adjacent cells in the grid are connected by an edge in E. STAC runs Depth-First-Search (DFS) on this graph in order to find all the cells that are reachable from the starting cell s (line 3). Next, it groups the reachable cells into $l + 1$ ($0 \leq l < n$) distinct threat levels according to their associated threat probabilities, starting from level 0 which contains all the safe cells and ending with level l which contains the most dangerous cells (line 4).

The loop in lines 5–14 iterates over all the threat levels from 0 to l. For each threat level, it creates a coverage path that connects all the cells that belong to that level (lines 6–7). This is performed by using the procedure Create_Safe_Coverage_Path (CSCP, shown in Algorithm 2). In order to combine all the coverage paths for the different threat levels into a complete coverage path, for each two consecutive threat levels i and $i + 1$, the algorithm finds the safest route between the last visited cell in level i and the first visited cell in level $i + 1$. This is performed by running Dijkstra's shortest paths algorithm (Dijkstra, 1959) on the graph H (lines 11–12), using the following edge weights:

$$w_{ij} = \begin{cases} p_j/p_{min} & \text{if cell } j \text{ contains a threat} \\ 1/n & \text{otherwise} \end{cases} \tag{4}$$

This weight function ensures that $w_{ij} \geq 1$ for edges that target a dangerous cell, while $w_{ij} = 1/n$ (where n is the grid size) for edges that target a safe cell. This way, the cost of visiting one dangerous cell is greater than the cost of visiting all the safe cells in the grid. Thus, only when there are two equally safe paths connecting two nodes, we will prefer the shorter one.

The final coverage path P returned by STAC consists of the coverage paths for all the threat levels and the route that connects them.

The procedure Create Safe Coverage Path (CSCP) finds the safest coverage path that connects all the cells that belong to a specific threat level. CSCP is composed of five main stages.

Algorithm 2 Create_Safe_Coverage_Path(G, H, C)

```
Input: a grid $G$, its induced graph $H$ and a group of reachable cells $C$
Output: a path $P$ that covers all the cells in $C$
1: $P \leftarrow \emptyset$
2: Build the graph $G_C$ induced from the grid cells $C$
3: Find the connected components (areas) of $G_C$ using DFS
4: Let $A_1, ..., A_k$ be the connected areas of $G_C$
5: for every area $A_i$, $1 \leq i \leq k$ do
6:     Run Spiral-STC on $A_i$
7:     Let $\pi_i$ be the coverage path generated by Spiral-STC
8:     $s_i \leftarrow$ first node in $\pi_i$
9:     $f_i \leftarrow$ last node in $\pi_i$
10: for every node $f_i$, $1 \leq i \leq k$ do
11:     Run Dijkstra on the graph $H$ starting from $f_i$
12:     for every node $s_j$, $j \neq i$ do
13:         Find the safest path $f_i \rightsquigarrow s_j$ by traversing the shortest paths tree
14: Build the graph $G_A$ of the connected areas
15: Run Christofides on $G_A$
16: Let $A'_1, ..., A'_k$ be the order of the areas along the route found by Christofides
17: for every area $A'_i$, $1 \leq i \leq k$ do
18:     Add the coverage path $\pi'_i$ of $A'_i$ to $P$
19:     if $i < k$ then
20:         Add the connecting path $f'_i \rightsquigarrow s'_{i+1}$ to $P$
21: return $P$
```

First, it finds all the connected areas of cells that belong to the given threat level (lines 2–3). A **connected area** is defined as a connected subset of cells in the grid that belong to the same threat level. We also define a **safe area** as a connected area composed of only safe cells and a **dangerous area** as a connected area composed of only dangerous cells. Figure 3 shows an example of a grid containing two safe areas and five dangerous areas that belong to two different threat levels. The two low-threat level areas are outlined by yellow lines and the three high-threat level areas are outlined by blue lines.

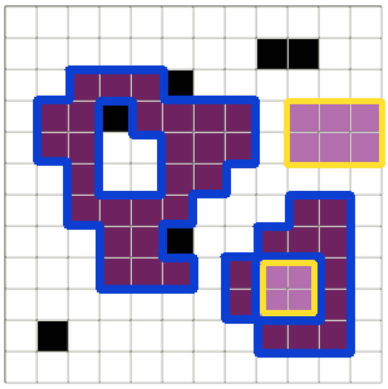

Fig. 3. An example of a grid containing two safe areas and five dangerous areas that belong to two different threat levels. The low-threat level areas are outlined by yellow lines and the high-threat level areas are outlined by blue lines.

CSCP finds the connected areas by first building the graph G_C induced from its input group of cells C (line 2). This graph is built the same way as the induced graph H in Algorithm 1, but using only the cells in C as vertices instead of the entire grid's cells. Then it runs Depth-First-Search (DFS) on this graph (line 3).

In the second stage, CSCP finds a coverage path for each connected area by running the Spiral Spanning Tree Coverage (Spiral-STC) algorithm (Gabriely and Rimon, 2003) (lines 5–9). In this algorithm, a single robot is assumed to be equipped with a square-shaped tool of size D: Any point within the tool's area is taken to be visited. The robot moves by sliding the tool over the area in any of the four basic directions (North, South, East, West). The work area is approximately divided into a grid with cells of size D. The grid is then made coarse, such that each new cell is of size $2D \times 2D$. Cells which are completely occupied by obstacles are discarded. The center-points of all the remaining cells are connected to those of their neighbors in the four basic directions, to form a graph. A spanning tree is then induced from the graph. The robot follows the edges of this spanning tree, while covering each $2D$-cell internally (see example in Figure 4). As can be seen, Spiral-STC generates spiral-like covering patterns.

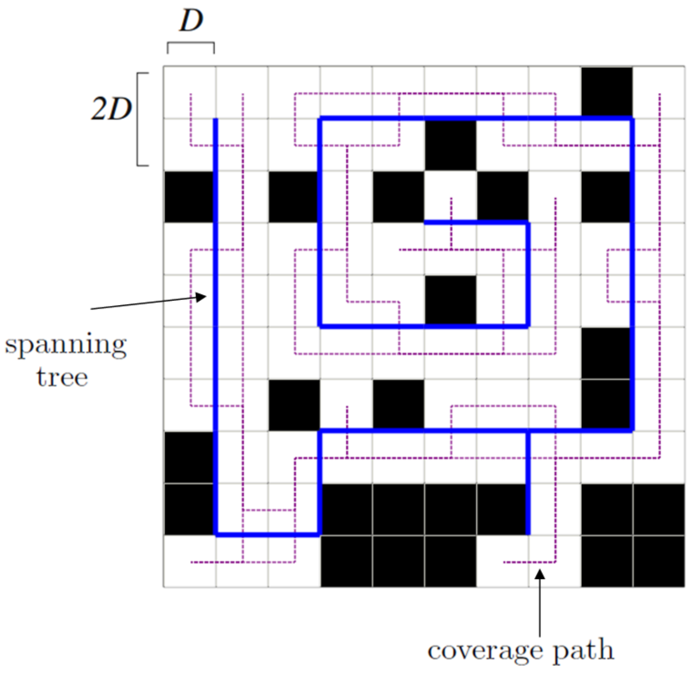

Fig. 4. An example of a spanning tree coverage. Black colored cells are occupied by obstacles. The spanning tree, denoted by a thick blue line, connects all the free coarse grid cells. The coverage path is the dotted line following along the spanning tree.

The main result is that Spiral-STC covers any planar grid in $O(n)$ time using a path whose length is at most $(n+m)D$, where n is the number of D-size cells and $m \leq n$ is the number of boundary cells defined below. In practice, $m \ll n$, thus Spiral-STC generates close-to-optimal coverage paths (Gabriely and Rimon, 2003).

Definition 4.1. ***Boundary cells*** *are cells that share either a point or a segment with a cell containing an obstacle.*

We have introduced a few changes to the original Spiral-STC algorithm. Specifically, the original algorithm always returns to the original cell from which it started the coverage. In our case there is no need to do that. As soon as all the cells in a given connected area are covered, the robot can immediately proceed to the next area. In addition, Spiral-STC assumes that the given grid contains an even number of rows and columns, thus it can be decomposed into $2D$-size cells. We have handled coverage of areas with an odd number of rows and/or columns, by adding one dummy row and/or column of obstacles to the original area, before running Spiral-STC.

In the third stage, CSCP finds the safest route between each pair of connected areas (lines 10–13). For this purpose, it runs Dijkstra's shortest paths algorithm (Dijkstra, 1959) on the induced graph H, using the same edge weights as in Eq. 4.

In the fourth stage, CSCP finds a safest route that connects all the connected areas (lines 14–16). For that purpose, it first builds the graph $G_A = (V_A, E_A)$. In this graph, each area in A is represented by a vertex in V_A, and the safest connecting path between two areas in A is represented by an edge in E_A, whose weight is equal to the the total weight of the connecting path. Then it runs an approximation algorithm for solving TSP (Traveling Salesman Problem) on this graph in order to find the safest possible route that visits each connected area exactly once. Since the cost function here satisfies the triangle inequality, we can use an approximate algorithm for TSP that returns a tour whose cost is not more than 1.5 the cost of an optimal tour (Christofides algorithm (Christofides, 1976)).

Finally, CSCP combines the coverage paths for each connected area and the route that connects them to create the final coverage path P (lines 17–20).

Analysis of the STAC algorithm

We first prove that STAC is complete, i.e., that it generates a path that covers every reachable free cell in the grid. This means that a robot following this path is guaranteed to cover the entire grid with some non-zero probability. We start with the following lemma, that proves the completeness of CSCP.

Lemma 1. *(completeness) Procedure CSCP creates a path that covers all the cells in its input C.*

Proof. In line 3 of the procedure, CSCP splits the cells in C into k connected areas $A_1, ..., A_k$. Each of these areas is covered by Spiral-STC (line 6 in CSCP), which is known to be complete (Gabriely and Rimon, 2003). Since all cells in C are reachable from the starting cell (this is a pre-condition on the input), there must be a connecting route between all the areas A_i, which is found in line 14 of the procedure. Thus, CSCP creates a path that covers all the cells in its input C. □

Theorem 4. *(completeness) STAC creates a path that covers every free cell accessible from the starting cell s.*

Proof. STAC invokes CSCP $l + 1$ times (line 7 in Algorithm 1), each time with the group D_i, which contains all the reachable cells from s that belong to threat level i ($0 \leq i \leq l$), except for cells that are already part of the path P generated by the algorithm. By lemma 1 CSCP is complete, thus all cells in D_i are covered, and they are added to P in line 14 of the algorithm. Since every free cell reachable from s belongs to one of the threat levels $0 \leq i \leq l$, this guarantees that all free reachable cells are added to P. In addition, line 13 of the algorithm makes sure that P is a legal path, by finding a connecting route between the last visited cell in D_{i-1} and the first visited cell in D_i and adding it to P (such a route must exist since all the cells in D_{i-1} and D_i are reachable from s). Hence, by the end of the algorithm, P is a path that covers all the free cells accessible from s. □

We now discuss the run-time complexity of STAC. We start with the following lemma, that gives the run-time guarantees of CSCP.

Lemma 2. *Let m be the number of cells in the group C given as input to CSCP, and the number of connected areas defined by these cells as a. Then CSCP finds a coverage path of C in $O(a^2 m \log m + a^3)$ time.*

Proof. First, CSCP runs DFS on the graph induced from the cells in C. The running time of DFS is linear in the number of nodes in the graph, since the graph is sparse (each node is connected to at most 4 neighbors), thus its running time is $O(m)$. Second, it runs Spiral-STC on each connected area, which by lemma 4.1 in (Gabriely and Rimon, 2003), covers a region of t cells in $O(t)$ time. Thus, the time to cover all the connected areas is $O(m)$. Third, CSCP runs Dijkstra's algorithm for each pair of connected areas, which takes $O(m \log m)$ (since it searches for the shortest path in the entire graph), for a total of $O(a^2 m\log m)$. Finally, it runs the Christofides algorithm on the graph induced from the connected areas, whose running time is $O(a^3)$. Hence, the entire complexity of CSCP is $O(a^2 m\log m + a^3)$. □

We now use lemma 2 to analyze the run-time complexity of STAC.

Theorem 5. *Let n be the total number of free cells accessible from the starting cell s, a the number of connected areas in the given grid G, and l the number of dangerous threat levels. Then STAC covers the given grid in $O((a^2 + l)n\log n + a^3)$ time.*

Proof. STAC calls CSCP for each threat level separately. Let us denote the number of cells that belong to threat level i ($0 \leq i \leq l$) by n_i and the number of connected areas that belong to that level by a_i. By lemma 2, the running time of calling CSCP with the cells that belong to threat level i is $O(a_i^2 n_i \log n_i + a_i^3)$. Thus, the total running time of these calls is: $O(\sum_{i=0}^{l} a_i^2 n_i \log n_i + a_i^3)$. The sets of connected areas in the different threat levels are mutually exclusive and their union is the set of connected areas in the entire grid, thus $\sum_{i=0}^{l} a_i = a$. Therefore, $\sum_{i=0}^{l} a_i^2 \leq (\sum_{i=0}^{l} a_i)^2 = a^2$ and similarly $\sum_{i=0}^{l} a_i^3 \leq (\sum_{i=0}^{l} a_i)^3 = a^3$. We also know that for every $0 \leq i \leq l$, $n_i \leq n$. Therefore, the total running time of the calls to CSCP is bounded by $O(a^2 n\log n + a^3)$. In addition, STAC executes Dijkstra's algorithm l times on the entire graph, in order to find the safest route between consecutive threat levels. The running time of this step is $O(ln\log n)$. Hence, the entire complexity of STAC is $O((a^2 + l)n\log n + a^3)$. □

The following corollary provides more specific bounds on STAC's run-time for specific types of environments.

Corollary 1. *In environments with randomly scattered threats the run-time complexity of STAC is $O(n^3\log n)$, which is the run-time in the worst case. On the other hand, in environments with a small number of contiguous dangerous areas, STAC's run-time complexity is $O(ln\log n)$.*

Proof. When the threats are randomly scattered across the environment, each dangerous area contains a small number of threats (typically only one threat). Thus, the number of dangerous areas has the same order of magnitude as the number of threats, which is a fraction of the number of cells in the grid. Hence, $a = \Theta(n)$. We also know that the number of threat levels is smaller than the number of cells in the grid, i.e., $l < n$. Thus, by theorem 5, the run-time complexity of STAC is $O(n^3\log n)$. Clearly, this is the worst-case running time, since $a, l \leq n$. On the other hand, when the threats are concentrated in a small number of contiguous areas, $a = O(1)$, and in such environments, the run-time complexity of STAC is $O(ln\log n)$. □

We now establish bounds on the minimum probability of completion and the length of the coverage path generated by STAC. We start with the following definition.

Definition 4.2. ***Connecting cells*** *are cells that reside on a connecting path between two different connected areas.*

The following lemma establishes a bound on the number of times STAC visits a connecting cell.

Lemma 3. *Let l be the number of dangerous threat levels. Then a cell c is visited by STAC at most $2(l + 1)$ times along a connecting path between two different areas.*

Proof. See Appendix A.2. □

Using lemma 3, we can now prove the following theorem.

Theorem 6. *Let l be the number of dangerous threat levels. Denote the number of cells that belong to threat level i by n_i ($0 \leq i \leq l$), and assume that out of them there are b_i boundary cells and c_i connecting cells. Let the threat probabilities of these levels be $p_0, ..., p_l$ ($p_0 = 0$). Then the coverage path generated by STAC covers the given grid using a path whose length is at most $\sum_{i=0}^{l}(n_i + b_i + 2lc_i)$ and its probability to complete is at least $\prod_{i=1}^{l}(1 - p_i)^{n_i + b_i + 2lc_i}$.*

Proof. See Appendix A.2. □

Notice that the ideal coverage path length and survivability probability are achieved when the robot visits every cell exactly once. Thus, the ideal coverage path length is n, where n is the number of free accessible cells in the grid, and the ideal completion probability is $\prod_{i=1}^{n}(1-p_i)^{n_i}$. In most environments this ideal probability cannot be attained, since the robot has to repeat its steps in order to complete the coverage. However, as shown by the next corollary, when the threats are concentrated in contiguous areas, STAC typically generates a coverage path whose survivability probability is very close to the ideal value.

Corollary 2. *When the threats are concentrated in contiguous areas, STAC typically generates a coverage path whose probability to complete is close to the ideal value* $\prod_{i=1}^{n}(1-p_i)^{n_i}$.

Proof. In each connected area, the number of boundary cells is typically much smaller than the total number of cells in that area (Gabriely and Rimon, 2003). This is due to the fact that boundary cells are cells that touch cells with obstacles, thus if there are too many boundary cells within an area, this area cannot remain connected due to the presence of obstacles within that area. Thus, for each threat level i, $b_i \ll n_i$. In addition, when the threats are concentrated in contiguous areas, typically there is a connecting path between the different areas which is entirely safe (i.e., consists of only safe cells). Thus, the number of dangerous connecting cells that belong to threat level i is much smaller than the number of cells that belong to this level, i.e., $c_i \ll n_i$. In addition, the number of threat levels is small in relation to the number of cells that belong to level i, i.e. $l \ll n_i$. Thus, for each threat level i, $n_i + b_i + 2lc_i \approx n_i$. Now, by theorem 6, STAC generates a coverage path whose probability to complete is at least $\prod_{i=1}^{l}(1-p_i)^{n_i+b_i+2lc_i} \approx \prod_{i=1}^{l}(1-p_i)^{n_i}$. □

In addition to finding the safest coverage path, STAC can also be used to find the shortest coverage path, by ignoring the threats in the environment (i.e., treating all the dangerous cells as safe cells). In this case, STAC behaves exactly like Spiral-STC, and the length of its coverage path is bounded by $n + b$, as shown by the next corollary. In most typical environments $b \ll n$, and in such environments STAC generates paths whose length is close to the ideal value n.

Corollary 3. *Let n be the total number of free cells accessible from the starting cell s and b the number of boundary cells. If the given environment contains no threats or threats are ignored, then STAC generates a coverage path whose length is bounded by $n + b$.*

Proof. In environments with no threats, there is only one threat level (level 0), and all the free cells that are reachable from the starting cell belong to one connected safe area. Thus, there are no connecting cells between different areas ($c_i = 0$). By theorem 6, in this case STAC generates a coverage path whose length is at most $n + b$. □

One of the main benefits of the layered approach utilized by STAC is that it can guarantee a minimum expected coverage until the robot is first hit by a threat. The next theorem establishes a minimum bound on the expected coverage obtained by the STAC algorithm. This bound is dependent upon the order in which the different threat-level regions are visited by STAC. Nevertheless, a comparison between this bound and the ideal expected coverage is provided after the theorem.

Theorem 7. *Let l be the number of dangerous threat levels. Let the threat probabilities of these levels be $p_0, ..., p_l$ ($p_0 = 0$). Let $A_{i,1}, ..., A_{i,k_i}$ be the connected areas of threat level i, arranged in the order of their visit by STAC. Let $|A_{i,j}|$ be the size of area $A_{i,j}$ and $m_{i,j}$ the number of cell visits needed to cover this area ($m_{i,j} \geq |A_{i,j}|$). Let $C_{i,j}$ be the set of cells on the connecting path between two consecutive areas $A_{i,j}$ and $A_{i,j+1}$ ($1 \leq j \leq k_i - 1$), and C_{i,k_i} be the set of cells on the connecting path between the last area of threat level i and the first area of threat level $i+1$. Denote by $P(C_{i,j})$ the probability to traverse the cells in $C_{i,j}$ without being hit by a threat. Then the expected number of visited cells before the robot is neutralized is at least:*

$$|A_{0,0}| + \sum_{i=0}^{l}\sum_{j=1}^{k_i}\left[\prod_{x=0}^{i-1}\left[(1-p_x)^{\sum_{y=1}^{k_x} m_{x,y}}\prod_{y=1}^{k_x}P(C_{x,y})\right]\cdot(1-p_i)^{\sum_{z=1}^{j} m_{i,z}}\prod_{z=1}^{j-1}P(C_{i,z})\right]|A_{i,j}| \tag{5}$$

Proof. See Appendix A.3. □

Notice that the ideal expected coverage is attained when the robot visits the cells precisely in increasing order of their threat probabilities, i.e., when it first visits all the safe cells, then all the cells with threat probability p_1, etc. Thus, if we denote the cells that belong to threat level i by $c_{i,1}, ..., c_{i,n_i}$, then the ideal expected coverage is:

$$\sum_{i=0}^{l}\sum_{j=1}^{n_i}\left[\prod_{x=0}^{i-1}(1-p_x)^{n_x}\right]\cdot(1-p_i)^{j-1} \tag{6}$$

In most environments the ideal expected coverage cannot be attained, since typically some of the cells that belong to a given threat level are disconnected, and the robot has to visit cells that belong to a higher threat level or revisit threat points that have already been covered in order to move between the disconnected cells. In these cases, the cell visits cannot precisely follow the order of the threat levels.

By comparing equations (5) and (6), we can see that the gap between the ideal expected coverage and the expected coverage achieved by STAC depends on the quality of the coverage algorithm of each connected area (which determines the gap between n_x and $\sum_{y=1}^{k_x} m_{x,y}$) and the quality of the connecting paths between the areas (which determines the values of the expressions $\prod_{y=1}^{k_x} P(C_{x,y})$). Finding an optimal solution to each of these problems is $\mathcal{NP}$-Complete, since the first one requires finding a coverage path with minimum length of each area (shown to be $\mathcal{NP}$-Complete in section 3), and the second one requires a solution to a TSP problem. However, STAC uses good approximations for both problems. For the coverage of each area it uses Spiral-STC, which creates a close-to-optimal coverage path, and for the connecting paths it uses the Christofides algorithm which provides a 1.5 approximation solution to the underlying TSP problem.

To conclude, in this section we have shown that STAC is complete, analyzed its run-time complexity, and proven optimality bounds on the path length, survivability probability and expected coverage of its generated coverage paths. We have also shown that its expected coverage is close to optimal, and in some types of environments, its coverage path length and survivability probability are also near-optimal. Table 1 summarizes the theoretical results proved in this section.

Question	Complexity or Bounds	Theorem No.
Run-time complexity	$O((a^2+l)n\log n + a^3)$ $O(n^3\log n)$	Theorem 5 Corollary 1
Safest path length	$\leq \sum_{i=0}^{l}(n_i + b_i + 2lc_i)$	Theorem 6
Shortest path length	$\leq n + b$	Corollary 3
Survivability probability	$\geq \prod_{i=1}^{l}(1-p_i)^{n_i+b_i+2lc_i}$	Theorem 6
Expected coverage	$\geq \lvert A_{0,0}\rvert + \sum_{i=0}^{l}\sum_{j=1}^{k_i}\Big[\prod_{x=0}^{i-1}$ $\left[(1-p_x)^{\sum_{y=1}^{k_x} m_{x,y}}\prod_{y=1}^{k_x}P(C_{x,y})\right]$ $\cdot(1-p_i)^{\sum_{y=1}^{j} m_{i,y}}\prod_{y=1}^{j-1}P(C_{i,y})\Big]\lvert A_{i,j}\rvert$	Theorem 7

Table 1: Summary of results in section 4.

5. GAC Algorithm

The Greedy Adversarial Coverage algorithm (GAC for short) follows a greedy approach, where in each step it leads the robot to the safest nearest cell to its current location which has not been covered yet (see Algorithm 3). The algorithm first builds the graph H, which is induced from the grid cells (line 2). This is the same graph that was built in Algorithm 1. Then in each iteration of its main loop (lines 5–10), it runs Dijkstra's shortest path algorithm (Dijkstra, 1959) on H, in order to find a minimum weighted path between the robot's current location and all the other cells in the grid (line 6) [2]. From Dijkstra's shortest paths tree, it finds an unvisited cell, which has a minimum weighted route from the current location of the robot (line 7). Then it extends the coverage path to that cell (line 8), marks this cell as visited (line 9) and continues with that cell (line 10). This loop continues until all the grid cells have been covered.

The weight function used for the graph's edges determines which type of coverage path GAC will search for. Let us denote by w_{ij} the weight of the edge between the nodes representing cells i and j in the grid's graph. Then the weight function is defined as follows:

$$w_{ij} = \begin{cases} -D \cdot \log(1 - p_j) + 1 & \text{if cell } j \text{ contains a threat} \\ 1 & \text{otherwise} \end{cases} \tag{7}$$

Here we can see that edges that target safe cells have a uniform fixed weight, which represents the cost for making one step in the grid. On the other hand, the weight of edges that target dangerous cells include, in addition, a penalty term $-D \cdot \log(1 - p_j)$ for making a risky step. This term uses the function $f(x) = -\log(1 - x)$, which is positive and monotonically increasing over the interval $0 < x < 1$. Since $0 < p_j < 1$, this means that transitions to cells with higher threat probabilities have higher costs than transitions to cells with lower threat probabilities. The reason for choosing f specifically as the weight function will become clearer when we prove the correctness of the algorithm (Theorem 10).

$D \geq 0$ is a fixed penalty assigned to making a risky move in the grid. Its value should be set according to the desired balance between the risk and the coverage time, as defined by the relation α/β in the objective function (Eq. 3). The higher this penalty is, the more the robot would be willing to make redundant steps in order to reduce the risk involved, at the expense of increasing the coverage time. For instance, setting $D = 0$ will make the algorithm find the shortest coverage path, while setting $D = \infty$ will make it find the safest coverage path. To help us calibrate the value of D according to the objective function, we will define that when $\alpha = \beta$, i.e., when the risk and the coverage time factors have equal importance, the penalty on making a move to a dangerous cell (with a minimum threat probability) will be equal to making one step in the grid. In particular, if p_{min} is the minimum threat probability, then D is set to:

$$D = -\frac{\alpha}{\beta} \cdot \frac{1}{\log(1 - p_{min})} \tag{8}$$

Using this value of D will make the weight of an edge that targets a cell with minimum threat probability be equal to $w_{ij} = 2$ when $\alpha = \beta$, i.e., the penalty for making a risky step will be 1, the same cost for making one step in the grid.

Analysis of the GAC Algorithm

We start by proving that GAC is complete.

Theorem 8. *(completeness) GAC creates a path that covers every free cell accessible from the starting cell s.*

Proof. All reachable cells from the starting cell s have a path that connects them. Thus, all of them will eventually become the node v in line 7 of the algorithm, which is added to the coverage path returned by the algorithm. □

[2] Since the induced graph is sparse, running Dijkstra n times here is more efficient than running Floyd-Warshall algorithm to compute the shortest paths between all pairs of nodes.

Algorithm 3 Greedy_Adversarial_Coverage

Input: a grid G and a starting cell s

Output: a coverage path P that covers all reachable cells in G from s

```
1: $P \leftarrow \emptyset$
2: Build the graph $H$ induced from the grid $G$ with the weight function $w$
3: Add the starting cell $s$ to $P$
4: Mark $s$ as visited
5: while not all reachable cells in $G$ have been visited do
6:     Run Dijkstra's shortest paths algorithm from $s$
7:     $v \leftarrow$ the node in $H$ with the minimum weighted distance from $s$ that has not been visited yet
8:     Add the path $s \rightsquigarrow v$ to $P$
9:     Mark $v$ as visited
10:    $s \leftarrow v$
11: return $P$
```

We now consider the run-time complexity of GAC.

Theorem 9. *Let n be the total number of free cells accessible from the starting cell S. Then GAC covers the given area in $O(n^2 \log n)$ time.*

Proof. Since the graph is sparse ($|E| = O(|V|)$), running Dijkstra's algorithm on the entire graph takes $O(n \log n)$ time, and the GAC algorithm runs it n times. □

We now prove the correctness of the greedy step, i.e., that by using the weight function defined in Eq. (7), the algorithm finds the optimal path to the next unvisited cell.

Theorem 10. *(correctness) In each cycle, GAC finds the optimal path from the current location of the robot to an unvisited cell in the grid.*

Proof. See Appendix A.4. □

We now establish bounds on the minimum completion probability and the length of the safest coverage path generated by GAC. We start with the following lemma, which establishes a bound on the number of visits in each cell.

Lemma 4. *Let l be the number of dangerous threat levels. Consider a cell c that belongs to threat level i, $0 \leq i \leq l$. Then c is visited at most $4(l - i + 1)$ times along the safest coverage path generated by GAC.*

Proof. Consider an outgoing edge e from cell c. Let us denote the group of cells which are accessible from e by A. The first time the algorithm traverses e, it is guaranteed that it will cover all the cells in A that belong to a threat level $\leq i$, and can be reached from e without going through a cell that belongs to a threat level higher than i, before it returns back to c. This is due to the fact that any path from a cell in A to a cell not in A will have to go back through c, and thus will have a higher risk than any path to a cell in A that belongs to a threat level $< i$, and will be longer than any path to a cell in A that belongs to threat level i (by the definition of the weight function (7), if two paths are equally safe, GAC will prefer the shorter one).

The next time the algorithm traverses e, it will visit a cell c' in A that belongs to a threat level $k > i$. This time it is guaranteed that the algorithm will cover all the cells in A that belong to a threat level $\leq k$, and can be reached from e without going through a cell that belongs to a threat level higher than k, before it returns back to c. This is because cells that belong to a threat level $\leq k$ which are not in A should have already been visited before the algorithm returned to A (since they had safer path than the path to cell c').

The next time the algorithm traverses e, it will cover all cells that belong to a higher threat level $m > k$ and so on and so forth until the algorithm covers the cells in the highest threat level that are reachable from e. In the worst case, A contains cells that belong to all threat levels $i, i+1, ..., l$. In this case, the algorithm will have to traverse edge e $l - i + 1$ times.

We know that cell c is connected to at most 4 edges in the grid graph. Therefore, c is visited at most $4(l - i + 1)$ times along the safest coverage path generated by GAC. □

Using lemma 4, we can now prove the following theorem.

Theorem 11. *Let l be the number of dangerous threat levels. Denote the number of cells that belong to each threat level by $n_0, ..., n_l$ and the threat probabilities of these levels by $p_0, ..., p_l$ $(p_0 = 0)$. Then the safest coverage path generated by GAC covers the given grid using a path whose length is at most $\sum_{i=0}^{l}\left[4n_i(l-i+1)\right]$ and its probability to complete is at least $\prod_{i=1}^{l}(1-p_i)^{4n_i(l-i+1)}$.*

Proof. Consider a cell c that belongs to threat level i. By lemma 4, c is visited by GAC at most $4(l - i + 1)$ times. Since there are n_i such cells, the number of visits to cells that belong to threat level i is at most $4n_i(l - i + 1)$. Hence, the length of the safest coverage path generated by GAC is at most $\sum_{i=0}^{l}\left[4n_i(l-i+1)\right]$.

The probability that the robot will not be stopped in a cell that belongs to threat level i is $1 - p_i$. Hence, the probability the robot will be able to complete its coverage is at least $\prod_{i=1}^{l}(1-p_i)^{4n_i(l-i+1)}$. □

The next theorem establishes an upper bound on the length of the shortest coverage path generated by the GAC algorithm. The shortest coverage path is obtained when GAC is executed on an environment that contains no threats or when the risk penalty is set to $D = 0$.

Theorem 12. *Let n be the total number of cells in the accessible grid. Then the shortest coverage path generated by GAC covers the given grid using a path that contains at most $4n$ cells.*

Proof. The outline of the proof is similar to the proof of lemma 4. Consider an outgoing edge e from a given cell c. Let us denote the group of cells which are accessible from e by A. The first time the algorithm traverses e, it is guaranteed that it will cover all the cells in A before it returns back to c, since any path from a cell in A to a cell not in A will have to go back through c, and thus will be longer than any path to a cell in A. Cell c is connected to at most 4 edges in the grid graph, thus c is visited by GAC at most 4 times. Hence, the coverage path generated by GAC contains at most $4n$ cells. □

To conclude, in this section we have shown that GAC is complete, analyzed its run-time complexity, and proven optimality bounds on the path length and survivability probability of its generated coverage paths. Although GAC's theoretical bounds are not as tight as STAC's, in the next section we will see that, in practice, GAC often outperforms STAC. Table 2 summarizes the theoretical results proved in this section.

Question	Complexity or Bounds	Theorem No.
Run-time complexity	$O(n^2 \log n)$	Theorem 9
Safest path length	$\leq \sum_{i=0}^{l}\left[4n_i(l-i+1)\right]$	Theorem 11
Shortest path length	$\leq 4n$	Theorem 12
Survivability probability	$\geq \prod_{i=1}^{l}(1-p_i)^{4n_i(l-i+1)}$	Theorem 11
Expected coverage	Open Question	

Table 2: Summary of results in section 5.

6. Experiment Results

In previous sections we have theoretically analyzed the two proposed algorithms, STAC and GAC, 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 algorithms in various types of simulated environments and compare their behavior in practice to their theoretical bounds. 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 threat levels, distribution of the threats, etc. Furthermore, we compare between different types of coverage paths generated by the algorithms. In particular, we examine the shortest path, the safest path and paths that target specific risk and time levels.

6.1. Finding the Shortest Coverage Path

To find the shortest coverage path we set $\alpha = 0$ in the objective function. For GAC, this means that the risk penalty D would be set to 0, thus all the threats in the map would be ignored. In order to use STAC to find the shortest coverage path, we can just treat all the cells in the grid other than obstacles as if they were safe cells.

To illustrate the operations of the algorithm, Figure 5 shows the shortest coverage paths found by STAC (in the upper figure) and GAC (in the bottom figure) on a sample map.

This map consists of 10×10 square cells, out of which 30% contains threat points, 20% contain obstacles and the other 50% cells are free and safe. The threat points are divided into 5 different threat levels with the following threat probabilities: $0.6\%, 1.2\%, 1.8\%, 2.4\%$, and 3%. Changing the absolute values of these probabilities does not affect the coverage paths generated by the algorithms; it just affects the scaling of the results. Both the threat points and the obstacles are randomly scattered across the map. The starting position of the robot is cell (1,1). The number of times the robot has visited each cell along the coverage path is indicated within that cell, and the coverage path is denoted by a solid line.

In this map, the length of the coverage path generated by STAC (upper figure) was 117, while the length of the coverage path generated by GAC (bottom figure) was 105. Since both algorithms in this case ignore the risks on the map, the coverage path length determines the probability to complete the coverage. In this case STAC's coverage probability to complete was 46%, while GAC's coverage probability to complete was 51.31%. The expected coverage percentage can change randomly between different maps, since the order of the threats visits is not determined.

The reason for the longer coverage path of STAC is that its coverage path circumscribes a spanning tree, and when one of the tree edges leads to a dead-end, the robot has to repeat its steps in order to get out of the dead-end. For example, the cells (3,3) and (3,4) are visited by STAC 3 and 2 times respectively, whereas they are visited by GAC only 2 times and once, respectively. This is due to the fact that these cells belong to the 2×2 cell (let us denote it by x) that contains inner cells (3,3), (3,4), (4,3) and (4,4). There is a tree edge that connects this cell to the 2×2 cell y that contains inner cells (3,5), (3,6), (4,5) and (4,6). The algorithm first visits large cell y, and then it enters cell x via its inner cell (3,4). Then cell (3,3) is visited twice on the path to cells (2,3) and (4,3), and once again when the algorithm returns from cell y back to cell x via the same tree edge.

After analyzing the shortest coverage paths generated by the algorithms on a specific map, we now describe the results of running the algorithms on 500 randomly-generated maps with obstacles ratios between 0% and 35%, in steps of 1% (in higher ratios of obstacles, most of the generated maps are disconnected). The threat ratio was set to 30%. However, this ratio was irrelevant, since in order to find the shortest coverage path, both algorithms ignore the threats. In all experiments, we used map sizes of 20×20 and the locations of the obstacles were randomly chosen.

Figure 6 shows the results. In addition to the coverage path length, we also compare the number of redundant steps the robot has made along the coverage path, i.e., the number of cells revisits. This measure reflects more accurately the impact of the obstacles ratio on the algorithms' performance, since as this ratio increases, there are less free cells in the map that

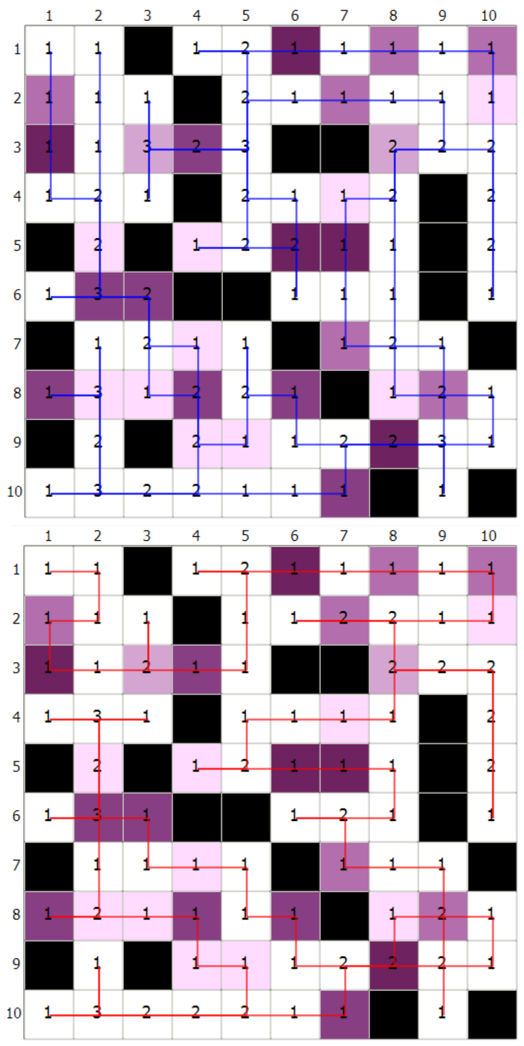

Fig. 5. Shortest coverage paths on a sample map with randomly scattered threats. The upper figure shows STAC's coverage path while the bottom figure shows GAC's coverage path on the same map. The robot starts the coverage in cell (1,1). The number in each cell indicates the number of visits in that cell.

need to be covered, but the robot has to make more redundant steps in order to complete the coverage. We do not report on the expected coverage percentage here, since both algorithms ignore the risks in this case, thus the expected coverage depends entirely on the environment.

As shown in the graph, in both algorithms, as the obstacles ratio increases the coverage path gets longer until some ratio (19% for STAC and 32% for GAC) and then it decreases, since the environment contains less free cells to cover. However, the number of repetitive steps keeps increasing. In both measures, GAC outperforms STAC (the results are statistically significant; one-tailed t-test $p = 1.998 \times 10^{-16}$). On the other hand, the running time of STAC was an order of magnitude faster than GAC's. Its average running time was 0.0006 seconds, while GAC's average running time was 0.189 seconds. Therefore, in environments with no threats or when threats are ignored, GAC attains shorter coverage paths at the expense of increased computation time.

An interesting observation from Figure 6 is that the graph of GAC's redundant steps is almost linear, i.e., the number of redundant steps added to GAC's coverage path is approximately the same as the number of obstacles added to the environment.

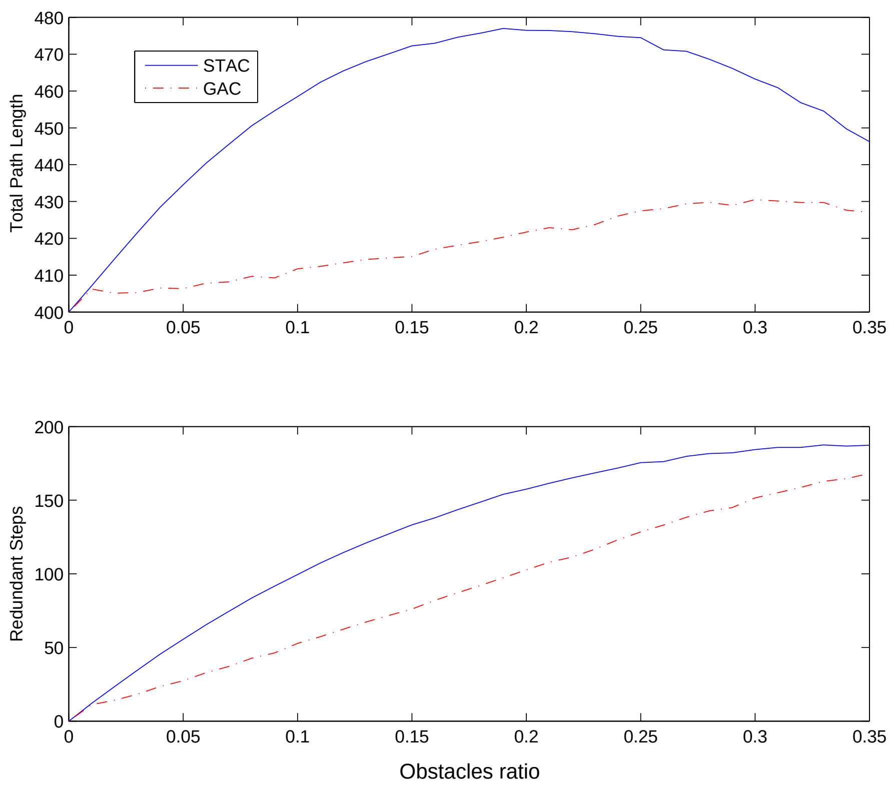

Fig. 6. Total path length and number of redundant steps for different obstacles ratios.

We can also observe that the experimental results match the theoretical bounds. The path length generated by STAC is less than $n + b < 2n$ (Corollary 3), and the path length generated by GAC is less than $4n$ (Theorem 12). In fact, GAC's shortest path length was significantly lower than its theoretical bound: its maximum value was $1.64n$ (this value was obtained when the obstacles ratio was 35%, the number of free cells was $n = 260$ and GAC's path length was $L = 427 = 260 \cdot 1.64$).

6.2. *Finding the Safest Coverage Path*

To find the safest coverage path we set $\beta = 0$ in the objective function. For GAC, this means that the risk penalty value should be set to $D = \infty$ (Eq. (8)). However, when $D = \infty$, the weights of all the edges that connect to dangerous cells are equal, thus the algorithm cannot distinguish between threat points with different threat probabilities. Instead we will set D such that the cost of moving into a cell with a minimum threat probability will be equal to the cost of covering the entire grid, thus in each greedy step GAC will always favor the longest possible safe path over a path with only one risky step. Specifically, if n denotes the number of accessible cells in the grid and p_{min} is the minimum threat probability, then D is set to:

$$D = -\frac{n}{\log(1 - p_{min})} \tag{9}$$

We have examined the safest coverage paths created by the algorithms in various types of environments and settings. In particular, we have observed noticeable difference in the behavior of the algorithms on maps with randomly scattered threat points vs. maps with contiguous dangerous areas.

Randomly scattered threat points.

We first examine the safest coverage paths generated by the algorithms on the same sample map from the previous section. Figure 7 shows the safest coverage paths found by STAC (in the upper figure) and GAC (in the bottom figure) on this map.

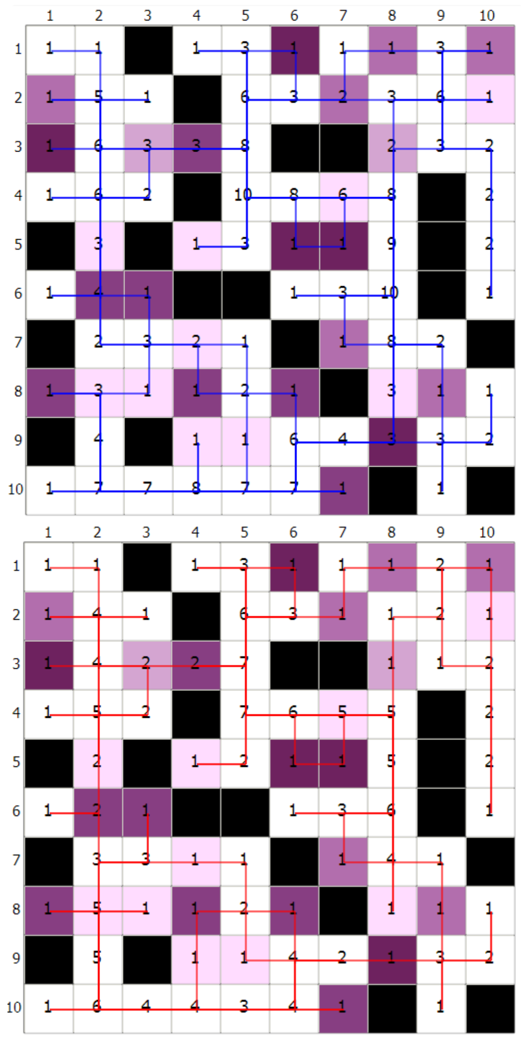

Fig. 7. Safest coverage paths on a sample map with randomly scattered threats. The upper figure shows STAC's coverage path while the bottom figure shows GAC's coverage path on the same map. The robot starts the coverage in cell (1,1). The number in each cell indicates the number of visits in that cell.

The expected coverage for STAC and GAC was 85.01% and 79.91%, respectively, while the probability to complete the coverage was 43.84% and 53.23%, respectively. Thus, STAC achieves better expected coverage, but its coverage is less likely to complete. The coverage path length was 248 for STAC and 184 for GAC.

Comparing the shortest and the safest coverage paths reveals that STAC was able to raise the expected coverage by 12.69% from the shortest to the safest path. This raise was at the expense of increasing the path length by 131 steps, which also caused the probability to complete the coverage to decrease by 2.16%. On the other hand, GAC was able to raise the expected coverage by 6.47% from the shortest to the safest path, while also increasing the probability to complete the coverage by 1.92%. GAC's path length increased by 79 steps. Hence, STAC's increase in the expected coverage was better, however its completion probability decreased and it suffered from a higher increase in its path length.

Analyzing the coverage paths created by the algorithms reveals the difference in their behavior. In Figure 7, both algorithms start with covering the upper-left safe area that contains the cells (1,1), (1,2), (2,2), (2,3), (3,2), (4,1), (4,2) and (4,3). When STAC finishes covering this area, it moves to the next closest safe area that contains the single cell (6,1) via the dangerous cells (5,2) and (6,2). Conversely, when GAC finishes covering this area, it visits cell (5,2) but then it goes back to visit dangeours cells (3,3) and (2,1), since they belong to a lower threat level than cells (6,2) and (3,4) that separate the first safe area from the next safe areas in the map. This causes the greedy algorithm to visit two more dangerous cells before moving to the next safe area, which explains its lower expected coverage.

After analyzing the safest coverage paths generated by the algorithms on a specific map, we now describe the results of running the algorithms on 500 random maps. We compare the expected coverage percentage, probability to complete and the path length of these coverage paths for threat ratios in the range between 0.0 and 0.5, in steps of 0.01. In all experiments, we used a map size of 20×20 and the ratio of obstacles was 20%. As in the previous section, the number of threat levels was 5 and the threat probabilities were uniformly dispersed between 0.6% and 3%. The locations of the threat points and the obstacles were randomly chosen.

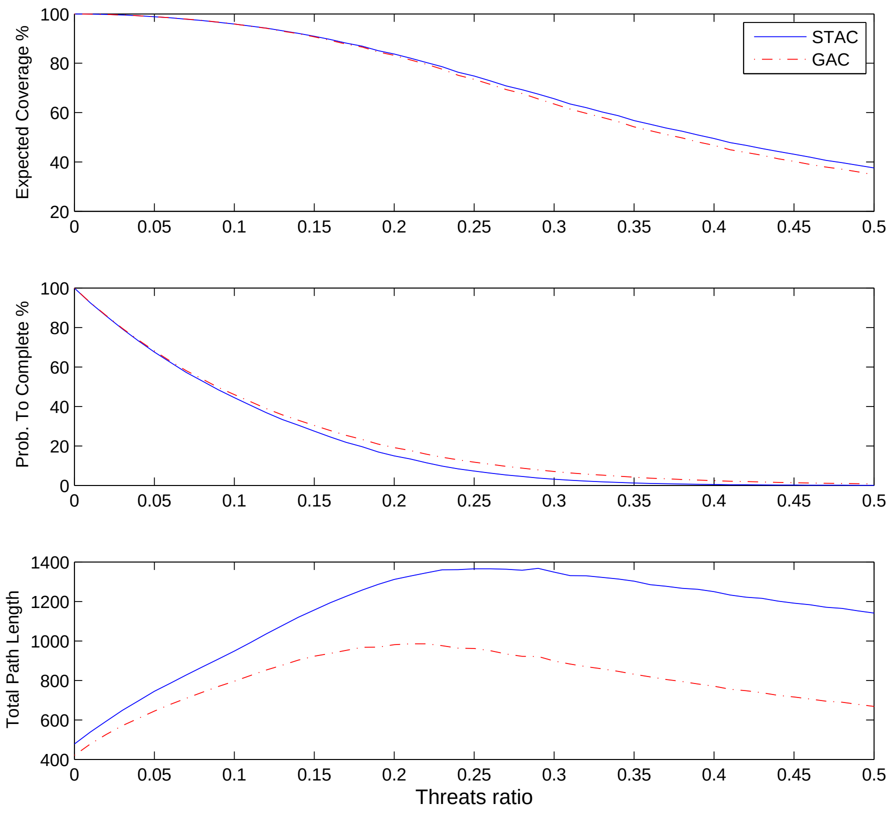

Fig. 8. Expected coverage percentage, probability to complete the coverage and total path length for different threat ratios in environments with randomly scattered threat points.

Figure 8 shows the results. As can be seen, in both algorithms the expected coverage percentage and the probability to complete the coverage decrease as we add more threats to the map. The coverage path length increases until the threat ratio reaches the level of around 25% and then it starts decreasing. The reason for this is that beyond this level, adding more dangerous cells to the grid leads to a decrease in the number of connected dangerous areas, since more dangerous cells become connected to each other. As a result, the connecting path between dangerous areas becomes shorter and there is less repetitive coverage in the transition between different areas.

Figure 8 also shows that in environments with randomly scattered threats both algorithms have similar expected coverage and completion probability. However, STAC consistently produces coverage paths with better expected coverage (up to 3% difference), while GAC produces shorter coverage paths with higher probability to complete (up to 5% difference). The reason for the similar results in this case is that when the threats are randomly scattered across the map, splitting the map into connected areas of the same threat level (typically each one containing only one cell) does not affect the visit order of the dangerous cells, i.e., both algorithms visit the threat points in the order of their threat probability.

We can see that the gap between STAC's and GAC's expected coverage widens as we add more threats to the map. When we add more threats to the map, more dangerous cells become connected and thus the dangerous areas become larger. Thus, STAC gains higher expected coverage, since it does not get stuck in a higher risk dangerous area before moving to a lower risk area.

On the other hand, GAC creates shorter coverage paths than STAC, since it is not forced to visit the cells by the order of their threat levels, thus it suffers less from revisits of cells on the transitions between the different dangerous areas, which may contain dangerous cells from lower threat levels. Moreover, as we saw in the previous section, GAC's internal coverage of each connected area is more efficient than STAC's (at the expense of computation time). The gap between GAC's and STAC's completion probability widens as we add more threats to the map, until the threat reaches the level of around 25%, where the path lengths are maximal, and then it starts narrowing again. The reason is that beyond this threat ratio, there are less dangerous areas, thus STAC suffers less from threats revisits on the connecting paths between the areas.

We now examine the effect of changing the number of threat levels on the performance of the algorithms. We compare the performance of both algorithms for a varying number of threat levels in the range between 1 and 20. The threat ratio was set to a fixed level of 30%. The other map settings remained as in the previous experiment. The results are shown in Figure 9.

We can see that as the number of threat levels increases, the expected coverage percentage gets higher for both algorithms. The reason is that as we add more threat levels, there are more cells with lower risk levels, thus both algorithms are able to cover more cells before visiting the cells with higher risk levels. STAC obtains statistically significant higher coverage percentage than GAC (average one-tailed t-test $p = 0.0021$). However, the difference between the two algorithms gets narrower as the number of levels increases. This is because when there are more cells that belong to lower threat levels, GAC can find its way out of a dangerous area more easily via cells with lower risk levels.

We can also see that as we add more threat levels, the coverage path length of both algorithms increase. STAC's path length is more affected than GAC's path length by the additional number of threat levels. This is due to the fact that when the number of threat levels increase, STAC has to perform more iterations of coverage. In each iteration it needs to cover all the dangerous areas that belong to a certain threat level, while revisiting cells that belong to lower threat levels on the connecting paths between these areas. Thus, adding more threat levels incur more cells revisits, which also affects the expected coverage and the completion probability.

As we add more threat levels, GAC's completion probability keeps increasing. This is because the threat ratio and the maximum threat probability remain the same throughout the experiments, thus as we add more threat levels the total risk of all the threat points in the map decrease. On the other hand, STAC's probability to complete the coverage almost does not change when there are more than 4 threat levels, since the decrease in the total risk is balanced out with the increase in the path length (which in turn adds more threats revisits).

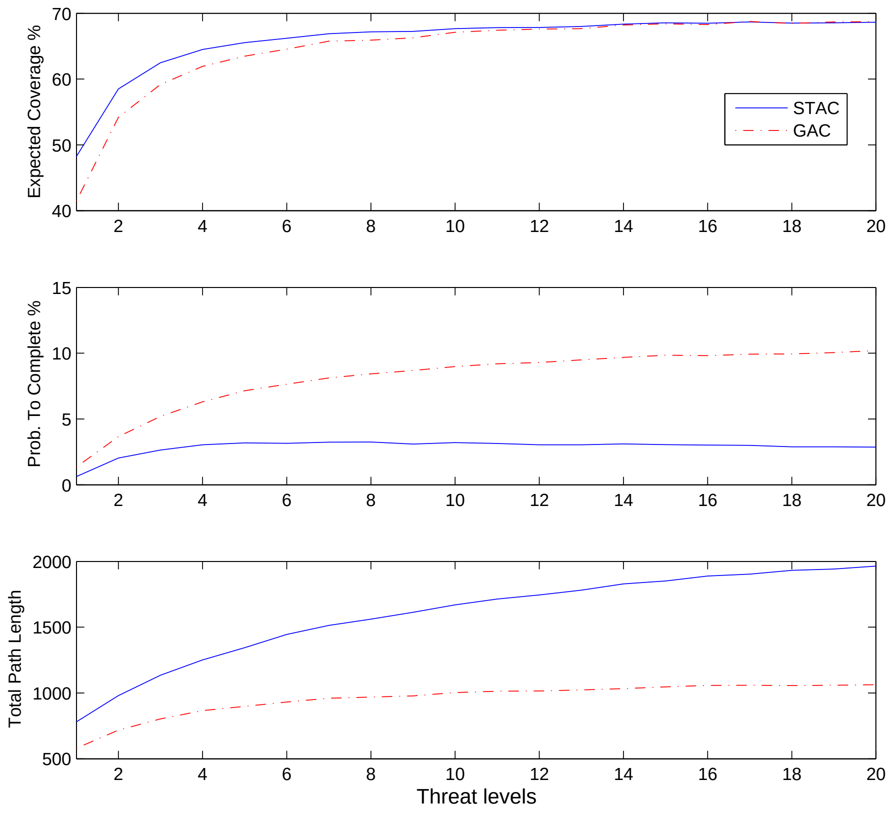

Fig. 9. Expected coverage percentage, probability to complete the coverage and total path length for different numbers of threat levels.

Contiguous areas of threats.

Up until now, we have considered environments that contain randomly scattered threats. However, in real-world environments, the threats may be concentrated in specific areas of the environment, e.g., in damaged parts of a nuclear plant that needs to be surveyed by the robot, or in areas of the battlefield with concentrated enemy forces. Therefore, we now examine environments where the threats are confined to contiguous areas and not scattered across the map. All the cells inside a given dangerous area belong to the same threat level, which is randomly chosen. Figure 10 shows an example for a map of such an environment. This map contains 6 dangerous areas that belong to 5 different threat levels. The ratio of obstacles was 20% and their locations were randomly chosen. The ratio of dangerous cells was 30%.

The expected coverage obtained by STAC and GAC was 71.68% and 60.51%, respectively. The probability to complete the coverage for STAC was 3.44% and for GAC it was 8.9%. Thus, STAC achieves better expected coverage, but its coverage is more unlikely to complete. STAC's coverage path length was 768, and GAC's coverage path length was 538.

We can learn from this map why STAC typically achieves better expected coverage than GAC at the expense of lower completion probability. GAC does not take into account the order of the threat levels when covering the connected areas. After finishing to cover the safe cells in the upper region of the map (from the first row to row no. 7), it covers the two dangerous areas in the upper region, before moving to visit other safe areas. On the other hand, STAC, after finishing to cover the safe areas in the upper region, it moves to the lower region of the map before covering any dangerous area, using the cells (9,12) and (10,12) as a connecting path between the regions. Dangerous cells that reside on a connecting path between different areas suffer from repeated visits by STAC (for example, cells (2, 12), (3, 12), (4, 12), (9, 12) and (10,

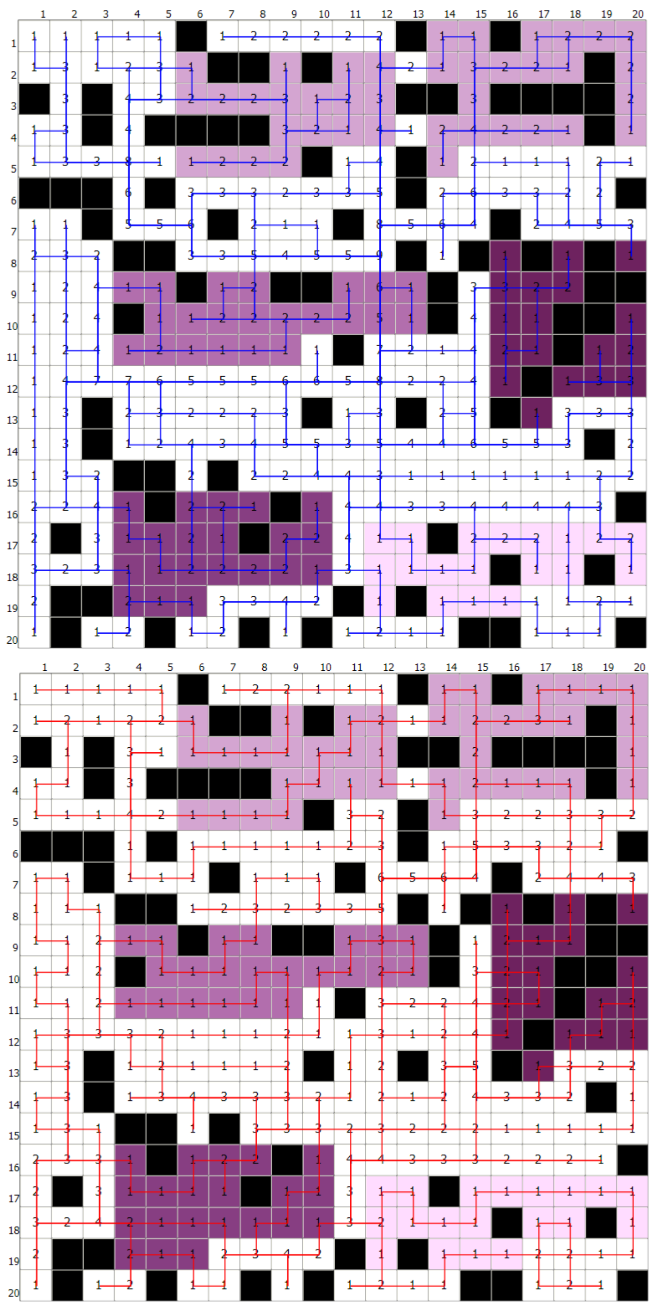

Fig. 10. An example map with contiguous dangerous areas. The upper figure shows STAC's coverage path while the bottom figure shows GAC's coverage path on the same map. The robot starts the coverage in cell (1,1). The number in each cell indicates the number of visits in that cell.

12)), which causes STAC's coverage path to have higher accumulated risk. In addition, we can see that STAC's internal coverage of each dangerous area is less efficient than GAC's (from the same reason that explained the difference between their shortest coverage path's lengths).

After analyzing the safest coverage paths generated by the algorithms on a specific map, we now describe the results of running the algorithms on 500 random maps with contiguous dangerous areas. Figure 11 compares the algorithms' performance for a varying number of dangerous areas between 2 and 40. We used the same map settings as in the maps with randomly scattered threats. The ratio of dangerous cells was set to 30%.

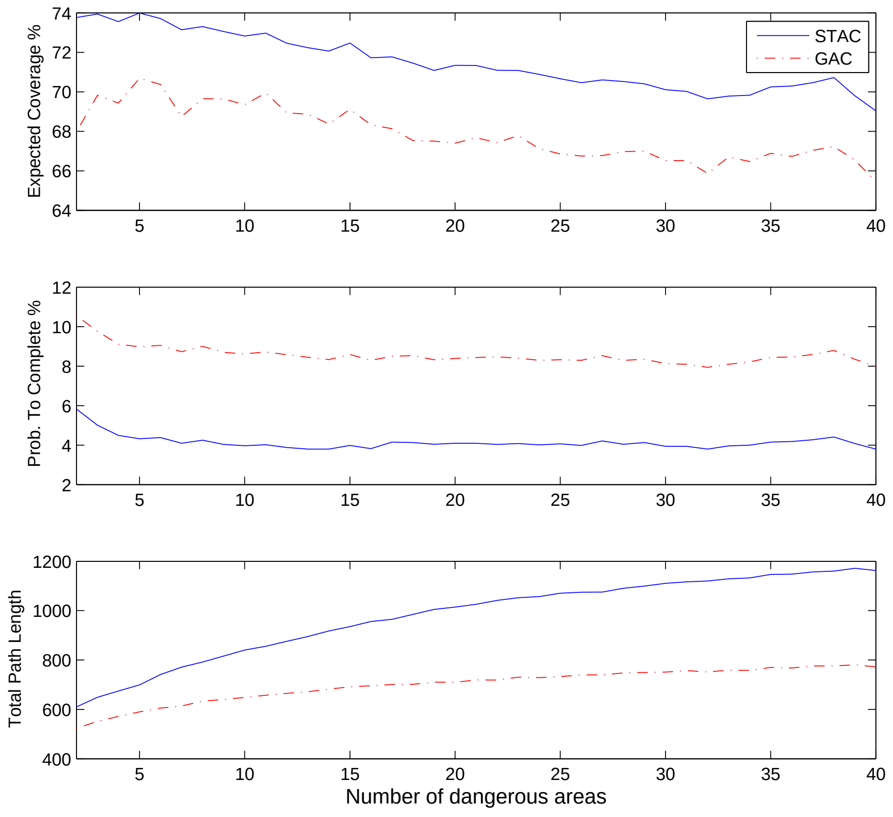

Fig. 11. Expected coverage percentage, probability to complete the coverage and total path length for different numbers of dangerous areas in environments with contiguous areas of threats.

Figure 11 shows that in both algorithms, as we add more dangerous areas the expected coverage and the probability to complete the coverage decrease, while the coverage path length increases. The reason for the decrease in the expected coverage is that when there are more dangerous areas, there is a greater partition of the safe cells into separate safe areas. Thus, the robot is able to cover less safe cells before hitting the first dangerous cell.

The reason for the increase in the coverage path length is that when there are more dangerous areas, the connecting path between dangerous areas that belong to the same level is longer, thus the algorithms need to revisit more lower risk cells when moving between the higher risk cells. This, in turn, causes the probability to complete the coverage to decrease. It has a more significant impact on STAC's coverage path than on GAC's, since STAC needs to cover all the areas that belong to a given threat level before moving to the next level.

The probability to complete the coverage also decreases as the dangerous cells are split into more areas. However, when the the number of areas reaches 10, adding more areas does not lead to any additional effect on coverage completion probability (it stays around 4% for STAC and 8% for GAC).

In environments with contiguous areas, the gap between STAC and GAC's performance becomes more conspicuous. STAC consistently generates coverage paths that have better expected coverage than GAC (between 4% and 6% difference, which is statistically significant; one-tailed t-test $p = 2.669 \times 10^{-29}$), while GAC consistently achieves a higher probability to complete the coverage (between 4% and 5% difference, which is statistically significant; one-tailed t-test

$p = 2.696 \times 10^{-54}$). As we add more dangerous areas, the gap between STAC's and GAC's expected coverage narrows, since when there are more dangerous areas, each dangerous area contains a smaller number of cells, thus there is less chance that GAC will get stuck in a dangerous area before finishing to cover all the safe cells. We can also notice that as the number of dangerous areas grows, the results become close to those of the randomly scattered threats case.

We now examine the effect of changing the number of threat levels in maps with contiguous dangerous areas. We compare the performance of both algorithms for a varying number of threat levels in the range between 1 and 20. The number of dangerous areas was set to 8. All the other map settings remained the same as in the previous experiments.

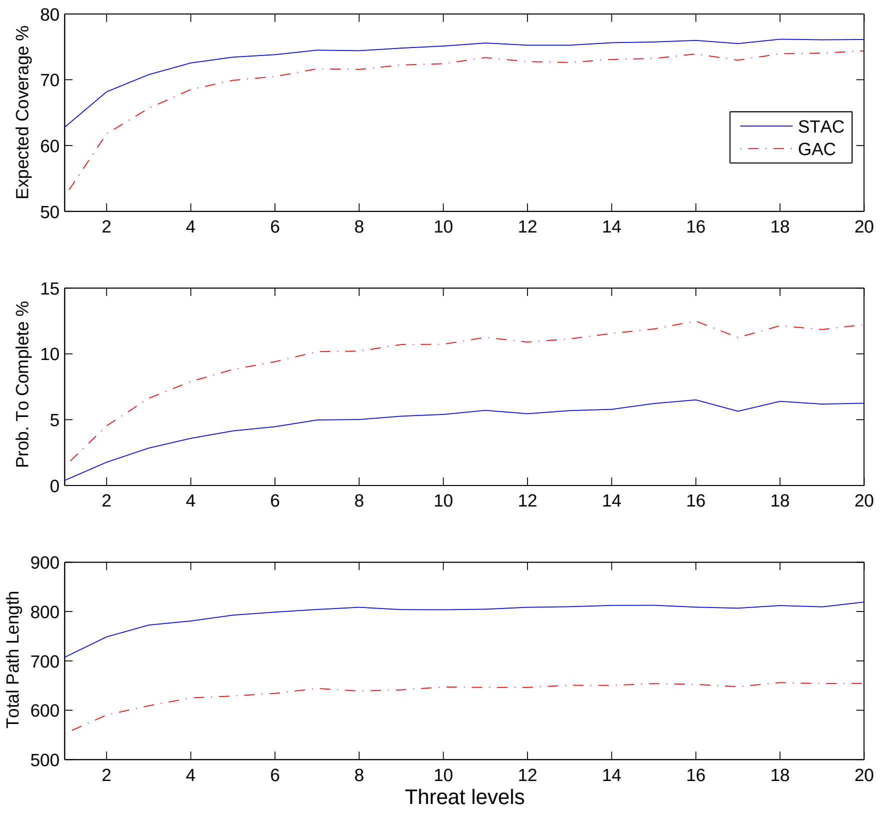

Fig. 12. Expected coverage percentage, probability to complete the coverage and total path length for different numbers of threat levels in environments with contiguous areas of threats.

Figure 12 shows the results. When there are more threat levels, the expected coverage and the probability of completing the coverage increase, at the expense of extending the coverage path length. In comparison to the randomly scattered threats case, STAC has a more clear advantage here in the expected coverage, while the difference in the completion probability is approximately the same. For example, in the uniform threats case (when there is only one threat level) STAC achieves a significantly higher expected coverage than GAC (10% higher). In environments with contiguous dangerous areas, there is an advantage to a global planning of the coverage, that arranges the areas by the order of their threat levels. Since GAC plans only one move in the grid at a time, it has a chance of getting stuck in a higher-level dangerous area before moving to a lower-level one (such as in scenario described in the sample map before). Thus, it suffers from a lower expected coverage.

When the number of threat levels gets above 8, the graphs reach a plateau, since the map contains only 8 dangerous areas and in each area all the cells belong to the same threat level, thus only 8 different threat levels can exist in the map.

Comparing the Experimental Results with the Theoretical Bounds.
Now, we would like to compare the experimental results with the predicted theoretical bounds as computed in sections 4 and 5. Figure 13 shows the comparison for a varying number of threat levels (averaged on 500 maps). We used the same map settings as in the previous experiments, except for the maximum threat probability that was reduced from 3% to 1% (otherwise some of the bounds were too low to be seen on the graph). Note that the bounds for GAC's expected coverage are not shown, as these are not known for the time being. Also, the bounds for the expected coverage and the completion probability are lower bounds, while the bounds for the coverage path length are upper bounds.

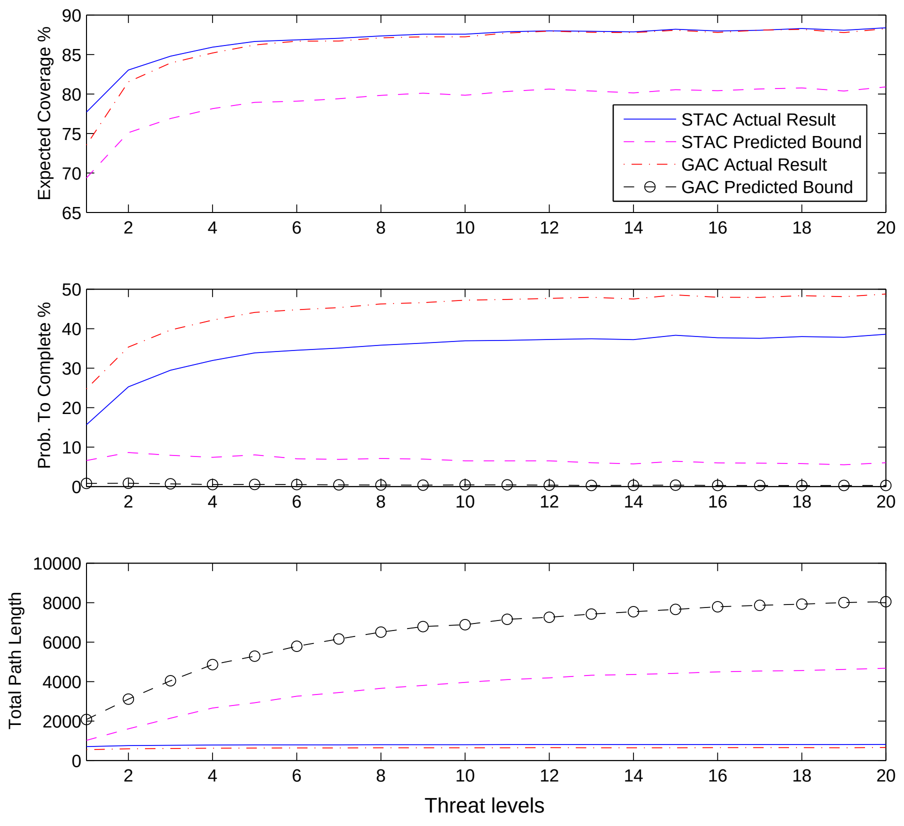

Fig. 13. Expected coverage percentage, probability to complete the coverage and total path length, and their respective theoretical bounds for different numbers of threat levels.

As expected, STAC's theoretical bounds are more tight than those of GAC. Specifically, STAC's actual expected coverage is close to its lower bound. Since this bound was proven to be close to the optimal expected coverage (see section 4), this means that the expected coverage percentage attained by STAC in practice is close to its optimal value. We can also observe that as the number of threat levels increases, the gaps between the probability of completing the coverage and the path length and their respective bounds widen, as these bounds have a strong dependency on the number of threat levels.

6.3. Finding Coverage Paths that Trade Risk and Time

We now consider the case of finding coverage paths that meet desired levels of risk and coverage time (number of steps). In GAC, these coverage paths can be found by adjusting the risk penalty D to the proportion α/β between the desired risk and time levels (Eq. (8)). In other words, they trade risk and time.

On the other hand, STAC cannot be used to find coverage paths for specific risk and time levels, since it always covers the cells in the order of their threat probability. The only way to modify the behavior of STAC is to regroup the grid cells into smaller number of threat levels than their original number of threat levels. For example, if the given map contains 10 threat levels, then we can reorganize the cells into 5 threat levels (cells that belong to threat levels 1 and 2 will be regrouped into the first threat level, cells that belong to threat levels 3 and 4 will be regrouped into the second threat level, etc.). This will make the coverage path generated by STAC shorter at the expense of a lower expected coverage. However, this technique cannot be used to precisely target the desired risk and time levels. Therefore, in this section we use only GAC for finding coverage paths that meet the specified risk and time levels.

Figure 14 shows the coverage path generated by GAC for the sample map with randomly scattered threats from the previous sections. The ratio between the risk and the time levels was set to $\alpha/\beta = 0.2$, which means that visiting a cell with a minimum threat probability incurs an additional 20% penalty to the cost for making one step in the grid.

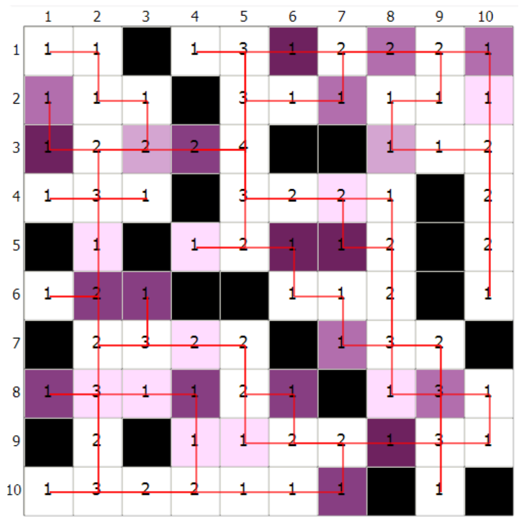

Fig. 14. GAC's coverage path for a ratio of 0.2 between the risk and the time levels on a sample map with randomly scattered threats. The robot starts the coverage in cell (1,1). The number in each cell indicates the number of visits in that cell.

In this case, the expected coverage was 78.88%, the probability to complete the coverage was 51.95% and the coverage's path length was 127. Compared to the shortest coverage path, the expected coverage raised by 6.44% and the probability to complete the coverage raised by 0.64%, at the expense of increasing the path length by 22 steps. Compared to the safest coverage path, the expected coverage of this path is lower by only 1.03% and the probability to complete is lower by 1.28%, but the path length is significantly lower (about 70% shorter).

Examining the coverage path in this case shows that the algorithm prefers to move from a given cell to its dangerous neighbor (a move that costs 1.2) over going back to its previous location and visiting another safe cell (a move that costs 2). For example, when the algorithm reaches cell (2,3) it prefers to move to its dangerous neighbor (3,3), then to go back to its already visited neighbor (2,2), and from there to the unvisited safe cell (3,2). This strategy enables the algorithm to make the coverage path much shorter, without significantly affecting the total accumulated risk (the dangerous cell (3,3) would have to be visited anyway at some point).

We now analyze GAC's performance for different ratios between the risk and time levels (α/β) in the range between 0.001 and $100{,}000$, with steps that double (i.e., 0.001, 0.002, 0.004, etc.). In all experiments, we used map sizes of 20×20, the ratio of obstacles was 20%, ratio of threats was 30% and the number of threat levels was 5. The locations of the threat points and the obstacles were randomly chosen.

Figure 15 shows the results averaged on 500 randomly-generated maps. Note that the x axis is plotted on a logarithmic scale.

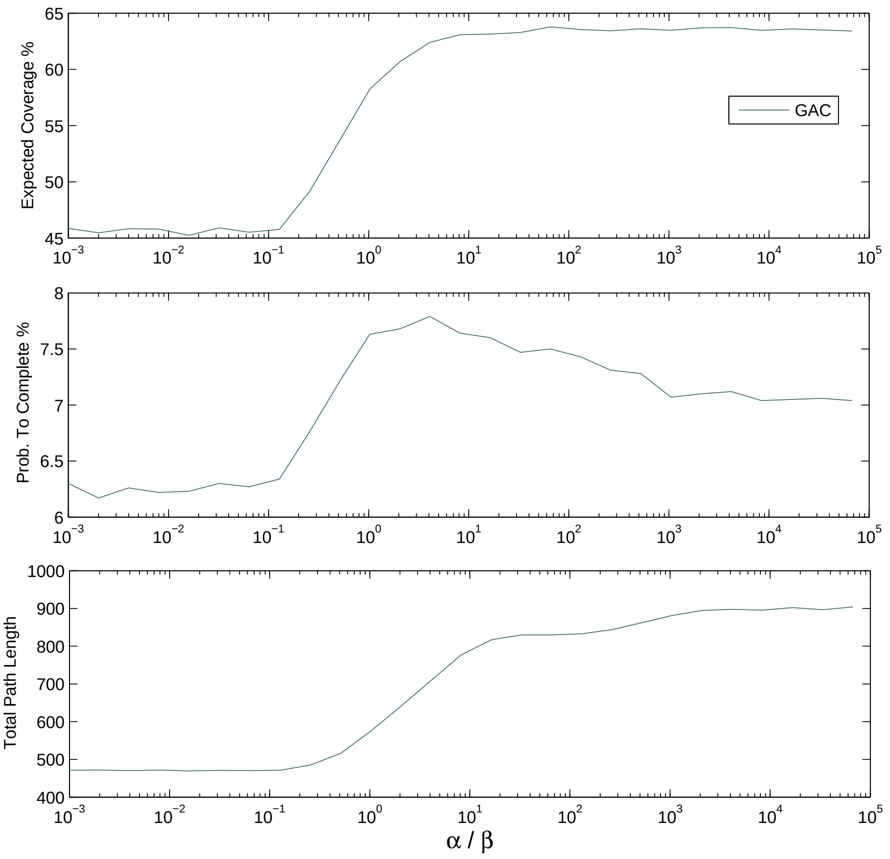

Fig. 15. GAC's expected coverage percentage, probability to complete the coverage and total path length for different ratios between the risk and time levels. Ratio α/β is plotted on a logarithmic scale.

As can be seen, raising the ratio α/β, i.e., making the risk factor more dominant, leads to higher expected coverage and completion probability at the expense of increased coverage time. The majority of the changes in the observed measures occur when this ratio is in the range between 0.1 and 10. We can also notice that when this ratio reaches a certain level (when α/β is about 5), there is a decrease in the completion probability while the expected coverage keeps increasing. This is due to the fact that when the ratio α/β is high, the algorithm tends to move back and forth between different dangerous areas in order to find the next safest cell from the robot's current position. This can lead to many redundant steps on the connecting paths between the dangerous cells, which in turn can cause the probability of completing the coverage to decrease.

7. Discussion and Future Research

In this paper we have discussed the adversarial coverage problem and its various aspects. First, we have suggested optimization criteria for the evaluation of coverage algorithms in adversarial environments. These criteria take into account both the survivability of the robot and the total coverage time. Second, we have analyzed the complexity of the problem and shown that is $\mathcal{NP}$-Complete. Third, we described two polynomial-time heuristic algorithms for finding a coverage path that meets the optimization criteria: STAC, which uses a layer-based approach, and GAC, which follows a greedy approach. We have provided optimality bounds on the total risk involved in the coverage paths generated by these algorithms and on their coverage time. Lastly, we have conducted systematic experiments with our implementation in order to evaluate the algorithms' effectiveness in various types of environments and under different settings. We have also examined the coverage paths generated by the algorithms on sample maps and analyzed their structures.

Comparing the two heuristic algorithms have shown that, in general, STAC creates coverage paths with higher expected coverage percentages, while the coverage paths generated by GAC are shorter and have lower accumulated risk. STAC has a clear advantage when handling maps with contiguous areas of threats, since it performs a global planning of the coverage according to the map's structure before the coverage begins, whereas GAC plans only one move of the robot at a time. Moreover, the theoretical bounds on the accumulated risk and the total path length of STAC's coverage paths are more tight than those of GAC's, and it runs faster than GAC. On the other hand, STAC is limited to finding only the safest and the shortest coverage paths, while GAC can be used to find coverage paths that meet any desired risk and time tradeoffs.

To demonstrate the practical utility of these algorithms, let us consider the case of the robot sent by TEPCO (Tokyo Electric Power Company) to explore Fukushima's damaged nuclear reactor (Ackerman, 2015). The robot's task was to collect data on radiation and temperature levels, and to find missing fuel rods, which apparently have fallen to the bottom of the reactor. The robot got stuck after covering about three-quarters of its originally planned route (five hours into the mission), when it ran into a fallen object while trying to go through a narrow passage. The information gathered by the robot will allow TEPCO to carry out more robot missions. In such a scenario, the adversarial coverage algorithms could have been used to plan a better coverage route for the next robots that will be sent to explore the nuclear site. First, the map that was generated by the first robot can be split into regular cells in the size of the covering robot. Each cell will be assigned a risk level, according to the probability of having a fallen object or another hazard that may harm the robot in that area. Second, the site operator can set the desired tradeoff between the coverage time and the total risk of the coverage path (being exposed to high levels of temperature or radiation for a long time might also affect the robot's ability to carry its task). Lastly, the operator can execute the adversarial coverage algorithms on the given map, in order to generate a coverage path for the robot, that takes into account the desired time and risk levels. This coverage path can be used either by the robot for autonomous exploration, or by the site operator for manually controlling the robot.

There are several areas we plan to pursue in future work. First, we are interested in extending the algorithms to handle online coverage, in which the coverage has to be completed without the use of a map or any a-priori knowledge of the area. Second, we would like to consider non-stationary environments, where the locations of the threat points can change over time. Finally, we would like to extend the algorithms for multi-robot systems. Using multiple robots for coverage has the potential for more efficient coverage and greater robustness; even if one robot is totally damaged, others may take over its coverage subtask.

Funding

This research was supported in part by ISF grant numbers 1337/15 and 1511/12.

References

Ackerman, E. (2015). *Unlucky robot gets stranded inside Fukushima nuclear reactor, sends back critical data. IEEE Spectrum*, 20 April 2015. Available: `http://spectrum.ieee.org/automaton/robotics/industrial-robots/robot-stranded-inside-fukushima-nuclear-reactor`.

Arkin, E. M., Fekete, S. P., and Mitchell, J. S. B. (2000). "Approximation algorithms for lawn mowing and milling". In: *Computational Geometry* 17.1-2, pp. 25–50.

Bortoff, S. A. (2000). "Path planning for UAVs". In: *Proceedings of the American Control Conference*. Vol. 1. 6, pp. 364–368.

Christofides, N. (1976). *Worst-case analysis of a new heuristic for the travelling salesman problem*. Tech. rep. 388. Graduate School of Industrial Administration, Carnegie-Mellon University.

Colegrave, J and Branch, A (1994). "A case study of autonomous household vacuum cleaner". In: *AIAA/NASA CIRFFSS*, p. 107.

Dijkstra, E. W. (1959). "A note on two problems in connexion with graphs". In: *Numerische mathematik* 1.1, pp. 269–271.

Elmaliach, Y., Agmon, N., and Kaminka, G. A. (2009). "Multi-robot area patrol under frequency constraints". In: *Annals of Mathematics and Artificial Intelligence* 57.3-4, pp. 293–320.

Endres, H., Feiten, W., and Lawitzky, G. (1998). "Field test of a navigation system: Autonomous cleaning in supermarkets". In: *IEEE International Conference on Robotics and Automation (ICRA-98)*. Vol. 2, pp. 1779–1781.

Gabriely, Y. and Rimon, E. (2001). "Spanning-tree based coverage of continuous areas by a mobile robot". In: *Annals of Mathematics and Artificial Intelligence* 31.1-4, pp. 77–98.

Gabriely, Y. and Rimon, E. (2003). "Competitive on-line coverage of grid environments by a mobile robot". In: *Computational Geometry* 24.3, pp. 197–224.

Galceran, E. and Carreras, M. (2013). "A survey on coverage path planning for robotics". In: *Robotics and Autonomous Systems* 61.12, pp. 1258–1276.

Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman.

Grigni, M., Koutsoupias, E., and Papadimitriou, C. (1995). "An approximation scheme for planar graph TSP". In: *36th IEEE Annual Symposium on Foundations of Computer Science*, pp. 640–645.

Hazon, N. and Kaminka, G. A. (2008). "On redundancy, efficiency, and robustness in coverage for multiple robots". In: *Robotics and Autonomous Systems* 56.12, pp. 1102–1114.

Itai, A., Papadimitriou, C. H., and Szwarcfiter, J. L. (1982). "Hamilton paths in grid graphs". In: *SIAM Journal on Computing* 11.4, pp. 676–686.

Likhachev, M. and Stentz, A. (2007). "Goal directed navigation with uncertainty in adversary locations". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS-07)*, pp. 4127–4134.

Nicoud, J. D. and Habib, M. K. (1995). "The Pemex-B autonomous demining robot: perception and navigation strategies". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems, 'Human Robot Interaction and Cooperative Robots'*. Vol. 1, pp. 419–424.

Nishizeki, T., Asano, T., and Watanabe, T. (1983). "An approximation algorithm for the Hamiltonian walk problem on maximal planar graphs". In: *Discrete applied mathematics* 5.2, pp. 211–222.

Papadimitriou, C. H. (1977). "The Euclidean travelling salesman problem is NP-complete". In: *Theoretical Computer Science* 4.3, pp. 237–244.

Papadimitriou, C. H. and Yannakakis, M. (1989). "Shortest paths without a map". In: *Automata, Languages and Programming*. Springer, pp. 610–620.

Xu, A., Viriyasuthee, C., and Rekleitis, I. (2011). "Optimal complete terrain coverage using an unmanned aerial vehicle". In: *IEEE International Conference on Robotics and Automation (ICRA-11)*, pp. 2513–2519.

Yaguchi, H. (1996). "Robot introduction to cleaning work in the East Japan Railway Company". In: *Advanced Robotics* 10.4, pp. 403–414.

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

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

Zabarankin, M., Uryasev, S., and Pardalos, P. (2002). “Optimal risk path algorithms”. In: Springer, pp. 273–298.

Zelinsky, A. et al. (1993). “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.

Appendices

A. Proofs

A.1. Proof of Theorem 3

Theorem 3: The GACP problem is $\mathcal{NP}$-Complete.

Proof. Clearly, GACP is in $\mathcal{NP}$, since one can easily guess the coverage path of the robot and then verify its length and probability of survival in polynomial time. To prove its $\mathcal{NP}$-hardness, we use a reduction from the Partition problem (Garey and Johnson, 1979), in which we are given a set N of items with positive integer weights a_i, and ask "Does there exist a subset $S \subset N$ such that $\sum_{i \in S} a_i = \frac{1}{2} \sum_{i=1}^{n} a_i$?". For the proof we use a scaled version of Partition, in which each a_i is less than 1 (and thus is not an integer).

Given an instance of the Partition problem, we construct an instance of the adversarial coverage problem as follows. We build a "corridor" consisting of n vertically adjacent large cells, numbered from 1 (top) to n (bottom). Each large cell consists of 4 adjacent small cells. We also create a gap consisting of one small cell between two consecutive large cells, so they will not overlap. The probability of danger is 0 for all the cells, except for the cell in the lower right corner of each large cell, which contains a threat with probability a_i (the value of the ith item in the Partition problem).

The following figure shows a description of the grid with the probabilities for danger specified in each cell. The dark colored cells represent obstacles.

0	0
0	a_1
0	■
0	0
0	a_2
0	■
...	...
0	0
0	a_n

Now, let us define the weights of the edges between adjacent cells in the grid, which represent the lengths of the paths that connect these cells. First, we denote the small cells in each large cell by:

c_{11}	c_{12}
c_{21}	c_{22}

Next, we define the weights of the edges between the small cells as: $w_{11,12} = w_{21,22} = \frac{a_i}{2}$ and $w_{11,21} = w_{12,22} = 1$. We also define the weight of the edges connecting two consecutive large cells as 1.

This completes the construction, which can be done in polynomial time.

Note that the optimal coverage path of the given grid must traverse each one of the large cells sequentially, otherwise the robot would have to repeat its steps in order to visit skipped cells. We now compute the length and the expected coverage of the shortest possible path and the safest possible path (with the maximum expected coverage) in each large cell.

The shortest coverage path of a large cell is $A = (c_{11}, c_{12}, c_{22}, c_{21})$. The length of this path is:

$$L(A) = 2 \cdot \frac{a_i}{2} + 1 = a_i + 1 \tag{A.10}$$

The expected number of covered cells in A is:

$$E(C_A) = 1 + 1 + (1 - a_i) + (1 - a_i) = 4 - 2a_i \tag{A.11}$$

On the other hand, there are two possible safest coverage paths (with maximum expected coverage): $B_1 = (c_{11}, c_{12}, c_{11}, c_{21}, c_{22}, c_{21})$ and $B_2 = (c_{11}, c_{21}, c_{11}, c_{12}, c_{22}, c_{21})$. For both paths, the sequence of unexplored cells discovered along this path is $(c_{11}, c_{12}, c_{21}, c_{22})$, thus the expected number of covered cells is:

$$E(C_B) = 1 + 1 + 1 + (1 - a_i) = 4 - a_i \tag{A.12}$$

However, path B_1 is shorter than B_2, since $L(B_1) = 4 \cdot \frac{a_i}{2} + 1 = 2a_i + 1$, and $L(B_2) = 2 \cdot \frac{a_i}{2} + 3 = a_i + 3$ $(a_i < 1)$. Thus, the optimal coverage path should prefer path B_1 over B_2 in covering cell i. Thus, from now on, we will refer to path B_1 as the safest path.

Hence, the safest path is longer than the shortest path by precisely a_i (the value of the ith item in the Partition problem), and its expected coverage is higher than the expected coverage of the shortest path also by a_i.

Now, we prove that there exists a coverage path of the grid of length at most $\frac{3}{2}\sum_{i=1}^{n} a_i + 3n - 2$ and expected coverage of at least $-\frac{3}{2}\sum_{i=1}^{n} a_i + 5n - 1$, if and only if the given numbers a_i can be partitioned evenly into two sets.

($\Rightarrow$) Suppose the numbers a_i can be partitioned evenly into two sets: S_1 and S_2. Thus, the sum of the numbers in S_1 and S_2 is $\frac{1}{2}\sum_{i=1}^{n} a_i$. Now we build the following coverage path P. For each a_i, if $a_i \in S_1$, then P uses the shortest path in order to cover the ith large cell, and if $a_i \in S_2$, then P uses the safest path in order to cover this cell. Hence, the length of P is:

$$L = \frac{1}{2}\sum_{i=1}^{n}(a_i + 1) + \frac{1}{2}\sum_{i=1}^{n}(2a_i + 1) + 2(n - 1) = \frac{3}{2}\sum_{i=1}^{n} a_i + 3n - 2 \tag{A.13}$$

and its expected coverage is:

$$E = \frac{1}{2}\sum_{i=1}^{n}(4 - a_i) + \frac{1}{2}\sum_{i=1}^{n}(4 - 2a_i) + (n - 1) = -\frac{3}{2}\sum_{i=1}^{n} a_i + 5n - 1 \tag{A.14}$$

($\Leftarrow$) Suppose there exists a coverage path P with length $L \leq \frac{3}{2}\sum_{i=1}^{n} a_i + 3n - 2$ and expected coverage $E \geq -\frac{3}{2}\sum_{i=1}^{n} a_i + 5n - 1$. We will show that in such case, the numbers a_i can be partitioned evenly into two sets: S_1 and S_2.

Without loss of generality, we can assume that P is an optimal coverage path, i.e., there is no other coverage path P' with both length $L' < L$ and expected coverage $E' < E$. If P were not optimal, then we could have chosen a coverage path P', which satisfies the same constraints as P and is optimal.

We first show that for each large cell i, P must either choose the shortest path or the safest path in order to cover it. Let us denote the shortest path by A and the safest path by B. Before reaching the dangerous cell c_{22}, P must either visit two safe cells (c_{11}, c_{12} or c_{11}, c_{21}) or all the three safe cells (c_{11}, c_{12}, and c_{21}). We now discuss these two cases.

Case 1. P visits two safe cells before reaching c_{22}. In this case, $E(P) \leq E(A)$, i.e., its expected coverage is at most the same as the shortest path, since the shortest path also visits two safe cells before reaching c_{22} and then it visits the last safe cell c_{21}. Since the shortest path visits each small cell only once, we must also have $L(P) \geq L(A)$. However, P is optimal, that it must satisfy $L(P) = L(A)$ (since $E(P) \leq E(A)$), hence $P = A$ (there is only one coverage path that visits each cell only once).

Case 2. P visits three safe cells before reaching c_{22}. In this case, $E(P) \leq E(B)$, i.e., its expected coverage is at most the same as the safest path, since the safest path also visits three safe cells before reaching c_{22}. In order to visit three safe cells before reaching c_{22}, P must visit cells c_{11} and c_{21} at least twice. Thus, it either must traverse the edges (c_{11}, c_{12}) and (c_{21}, c_{22}) twice, or traverse the edge (c_{11}, c_{21}) twice. The path length between c_{11} and c_{21} is 1, which is more than twice longer than the length of the edges (c_{11}, c_{12}) and (c_{21}, c_{22}), which is $\frac{a_i}{2}$. Since P is optimal, it should prefer to traverse the edges (c_{11}, c_{12}) and (c_{21}, c_{22}) twice. In addition to traversing these edges, it must also traverse at least one edge of length 1 ((c_{12}, c_{22}) or (c_{11}, c_{21})). Thus, its total length is at least $L(P) \geq 2a_i + 1 = L(B)$. From the optimality of P, we must have $L(P) = L(B)$ and $E(P) = E(B)$. Thus, $P = B$ (there is no other coverage path with the same length and expected coverage as B).

Therefore, P must either choose the shortest path or the safest path to cover each large cell i. We will now define the sets S_1 and S_2. For each large cell i, if P chooses the shortest path in order to cover it, we will assign a_1 to S_1, otherwise we will assign it to S_2.

We will now show that $\sum_{i \in S_1} a_i = \sum_{j \in S_2} a_j = \frac{1}{2}\sum_{i=1}^{n} a_i$. First, we denote: $k = \sum_{i \in S_1} a_i$ and $s = \sum_{i=1}^{n} a_i$. Thus, $\sum_{i \in S_2} a_i = s - k$. We need to prove that $k = \frac{s}{2}$. Assume by contradiction that $k \neq \frac{s}{2}$. Thus, there are two possible cases:

Case 1. $k < \frac{s}{2}$. In this case, the length of P is:

$$\begin{aligned}\sum_{i \in S_1}(a_i + 1) + \sum_{j \in S_2}(2a_j + 1) + 2(n-1) &= \sum_{i \in S_1} a_i + \sum_{i \in S_1} 1 + 2\sum_{j \in S_2} a_j + \sum_{j \in S_2} 1 + 2n - 2 = \\ &= k + 2(s-k) + \sum_{i=1}^{n} 1 + 2n - 2 = 2s - k + 3n - 2 > \frac{3s}{2} + 3n - 2 \geq L \end{aligned} \tag{A.15}$$

which leads to a contradiction.

Case 2. $k > \frac{s}{2}$. In this case, the expected number of covered cells in P is:

$$\begin{aligned}\sum_{i \in S_1}(4 - 2a_i) + \sum_{j \in S_2}(4 - a_j) + (n-1) &= -2\sum_{i \in S_1} a_i + \sum_{i \in S_1} 4 - \sum_{j \in S_2} a_j + \sum_{j \in S_2} 4 + n - 1 = \\ &= -2k - (s-k) + \sum_{i=1}^{n} 4 + n - 1 = -s - k + 5n - 1 < -\frac{3s}{2} + 5n - 1 \leq E \end{aligned} \tag{A.16}$$

which also leads to a contradiction.

Therefore, we must have that $k = \frac{s}{2}$, i.e., $\sum_{i \in S_1} a_i = \sum_{j \in S_2} a_j = \frac{1}{2}\sum_{i=1}^{n} a_i$, and thus the numbers a_i can be evenly partitioned into two sets.

Thus, the GACP solution solves Partition. Therefore, GACP is $\mathcal{NP}$-complete. □

A.2. Proof of Theorem 6

Lemma 3: Let l be the number of dangerous threat levels. Then a cell c is visited by STAC at most $2(l+1)$ times along a connecting path between two different areas.

Proof. During the coverage of a given threat level i, procedure CSCP computes the connecting route between the areas that belong to this level by running the Christofides approximation algorithm to TSP (line 14). This algorithm is based on creating an MST (Minimum Spanning Tree) of the graph and then adding some edges to the graph in order to create an Euler circuit. Just by traversing the MST twice, the tour will be within a factor of 2 of the optimal TSP solution. Christofides has a 1.5 approximation factor, since the Euler circuit helps it visit some of the tree edges less than twice. In any case, the route produced by the Christofides algorithm cannot traverse the cells on the connecting path more than twice.

In the worst case scenario, all the cells that belong to threat level i lie on a connecting path between two areas. Such a scenario is illustrated in Figure 16. In this map there are two threat levels, one containing safe cells and the second containing dangerous cells. The separating area between the upper and lower safe areas has an S-shape form, composed of alternating rows of dangerous cells and obstacles. In this case the robot must traverse all the dangerous cells in order to move between the safe areas.

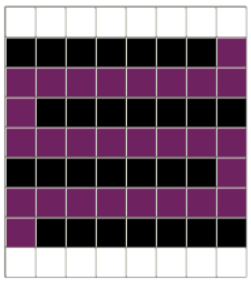

Fig. 16. An example for an environment where all the dangerous cells lie on a connecting path between two safe areas.

In the worst case, c lies on the connecting route of every threat level, in which it can be visited at most twice. Since there are $l + 1$ threat levels, c is visited by STAC at most $2(l + 1)$ times along a connecting path. □

Theorem 6: Let l be the number of dangerous threat levels. Denote the number of cells that belong to threat level i by n_i $(0 \leq i \leq l)$, and assume that out of them there are b_i boundary cells and c_i connecting cells. Let the threat probabilities of these levels be $p_0, ..., p_l$ $(p_0 = 0)$. Then the coverage path generated by STAC covers the given grid using a path whose length is at most $\sum_{i=0}^{l}(n_i + b_i + 2lc_i)$ and its probability to complete is at least $\prod_{i=1}^{l}(1 - p_i)^{n_i+b_i+2lc_i}$.

Proof. Consider the group of cells that belong to threat level $i, 0 \leq i \leq l$. Let us denote this group by G_i. The cells in G_i can be visited by STAC during the coverage of the connected areas that belong to threat level i (line 6 in CSCP) or along the connecting path between different areas (line 18 in CSCP or line 7 in Algorithm 1).

First, we count the number of visits to cells in G_i during the coverage of threat level i. Let us denote the connected areas that belong to this level by $A_1, ..., A_k$, and the number of boundary cells in area A_j by b_{A_j} $(1 \leq j \leq k)$. By theorem 1 in (Gabriely and Rimon, 2003), the total number of cells revisits in Spiral-STC is bounded by the number of boundary cells in the work-area grid. Since STAC runs Spiral-STC on each connected area separately, the total number of visits to cells during the coverage of threat level i is:

$$\sum_{j=1}^{k}(|A_j| + b_{A_j}) \leq n_i + b_i \tag{A.17}$$

The inequality is due to the fact that some of the cells may already have been visited along the connecting path of areas that belong to a lower threat level (line 18 in CSCP), and thus are not covered again here.

Next, we count the number of visits to cells in G_i along the connecting paths between different areas. By lemma 3, the algorithm has to traverse each connecting cell c_i at most $2(l + 1)$ times. However, if a connecting cell c_i has been already visited along a path that connects areas that belong to a threat level $< i$, then it will not be covered during the coverage of threat level i. Thus, the total number of visits to cells in G_i along connecting paths between areas is bounded by $2lc_i$.

Therefore, the total number of visits to cells in G_i is bounded by $n_i + b_i + 2lc_i$. Hence, the length of the coverage path generated by STAC is at most $\sum_{i=0}^{l}(n_i + b_i + 2lc_i)$.

The probability that the robot will not be stopped in a cell that belongs to threat level i is $1 - p_i$. Hence, the probability the robot will be able to complete its coverage is at least $\prod_{i=1}^{l}(1 - p_i)^{n_i+b_i+2lc_i}$. □

A.3. Proof of Theorem 7

Theorem 7: Let l be the number of dangerous threat levels. Let the threat probabilities of these levels be $p_0, ..., p_l$ ($p_0 = 0$). Let $A_{i,1}, ..., A_{i,k_i}$ be the connected areas of threat level i, arranged in the order of their visit by STAC. Let $|A_{i,j}|$ be the size of area $A_{i,j}$ and $m_{i,j}$ the number of cell visits needed to cover this area ($m_{i,j} \geq |A_{i,j}|$). Let $C_{i,j}$ be the set of cells on the connecting path between two consecutive areas $A_{i,j}$ and $A_{i,j+1}$ ($1 \leq j \leq k_i - 1$), and C_{i,k_i} be the set of cells on the connecting path between the last area of threat level i and the first area of threat level $i + 1$. Denote by $P(C_{i,j})$ the probability to traverse the cells in $C_{i,j}$ without being hit by a threat. Then the expected number of visited cells before the robot is neutralized is at least:

$$|A_{0,0}| + \sum_{i=0}^{l}\sum_{j=1}^{k_i}\left[\prod_{x=0}^{i-1}\left[(1-p_x)^{\sum_{y=1}^{k_x} m_{x,y}}\prod_{y=1}^{k_x}P(C_{x,y})\right]\cdot(1-p_i)^{\sum_{y=1}^{j} m_{i,y}}\prod_{y=1}^{j-1}P(C_{i,y})\right]|A_{i,j}| \tag{A.18}$$

Proof. According to Eq. (2), the expected coverage is the sum of the probabilities to reach all the new cells along the coverage path:

$$E(C_A) = \sum_{i\in(b_1,...,b_n)}\prod_{j\in g_i}(1-p_j) \tag{A.19}$$

Consider a newly discovered cell c that belongs to a connected area $A_{i,j}$ in threat level i, $0 \leq i \leq l$. This cell could be first discovered when STAC covers the areas of threat level i, or when it is used as a connecting cell to move between areas of a lower threat level. Let us assume that this cell is discovered when STAC covers the areas of threat level i. In this case, the probability that the robot is able to reach cell c before being hit by a threat equals to the probability that it will be able to cover all threat levels $< i$ and all the areas of threat level i preceding the area that c belongs to.

The probability that STAC will be able to cover a given threat level x equals to the probability that it will be able to cover all its connected areas and the connecting paths between these areas. The probability to cover a connected area $A_{x,y}$ that belongs to threat level x is $(1-p_x)^{m_{x,y}}$, where $m_{x,y}$ is the number of cell visits needed to cover $A_{x,y}$ ($m_{x,y} \leq |A_{x,y}|+b_{x,y}$, where $b_{x,y}$ is the number of boundary cells in $A_{x,y}$). There are k_x connected areas that belong to threat level x, thus the probability to cover all the areas that belong to this threat level is: $(1 - p_x)^{\sum_{y=1}^{k_x} m_{x,y}}$. Now, the probability to cover all the connecting paths between the areas that belong to threat level x, including the connecting path between threat levels x and $x + 1$, is $\prod_{y=1}^{k_x} P(C_{x,y})$. Thus, the probability to cover threat level x including its areas and all the connecting paths is: $(1-p_x)^{\sum_{y=1}^{k_x} m_{x,y}} \prod_{y=1}^{k_x} P(C_{x,y})$. Therefore, the probability that the robot will be able to cover all threat levels preceding level i is:

$$\prod_{x=0}^{i-1}\left[(1-p_x)^{\sum_{y=1}^{k_x} m_{x,y}}\prod_{y=1}^{k_x}P(C_{x,y})\right] \tag{A.20}$$

In addition, to reach cell c the algorithm has to cover all the areas of threat level i preceding the area that c belongs to and their connecting paths. In the worst case, c is the last cell visited in its area, and in this case the algorithm also has to cover the area that c belongs to before reaching c. Thus, the probability that the robot will be able to cover all areas in threat level i before reaching c is at least:

$$(1-p_i)^{\sum_{y=1}^{j} m_{i,y}} \prod_{y=1}^{j-1} P(C_{i,y}) \tag{A.21}$$

By multiplying equations (A.20) and (A.21) we get the total probability to reach cell c:

$$\prod_{x=0}^{i-1} \Big[(1-p_x)^{\sum_{y=1}^{k_x} m_{x,y}} \prod_{y=1}^{k_x} P(C_{x,y})\Big] \cdot (1-p_i)^{\sum_{y=1}^{j} m_{i,y}} \prod_{y=1}^{j-1} P(C_{i,y}) \tag{A.22}$$

Note that this is a lower bound on the probability to reach cell c, since c could also be discovered in an earlier stage, on a connecting path between areas of a lower threat level.

Since the probability in Eq. (A.22) is the same for all cells in area $A_{i,j}$, the expected number of cells that will be covered in this area is at least:

$$\left[\prod_{x=0}^{i-1} \Big[(1-p_x)^{\sum_{y=1}^{k_x} m_{x,y}} \prod_{y=1}^{k_x} P(C_{x,y})\Big] \cdot (1-p_i)^{\sum_{y=1}^{j} m_{i,y}} \prod_{y=1}^{j-1} P(C_{i,y})\right] |A_{i,j}| \tag{A.23}$$

Finally, the expected coverage is the sum of the expected number of visited cells in all areas $A_{i,j}$:

$$\sum_{i=0}^{l}\sum_{j=1}^{k_i} \left[\prod_{x=0}^{i-1} \Big[(1-p_x)^{\sum_{y=1}^{k_x} m_{x,y}} \prod_{y=1}^{k_x} P(C_{x,y})\Big] \cdot (1-p_i)^{\sum_{y=1}^{j} m_{i,y}} \prod_{y=1}^{j-1} P(C_{i,y})\right] |A_{i,j}| \tag{A.24}$$

To this sum we add the number of cells in $A_{0,0}$, which is the first safe area and thus is guaranteed to be covered with probability 1. □

A.4. Proof of Theorem 10

Theorem 10: (correctness) In each cycle, GAC finds the optimal path from the current location of the robot to an unvisited cell in the grid.

Proof. Let us denote the path chosen by GAC in a given cycle by $P = (u_1, ..., u_k)$, where u_1 is the robot's current location and u_k is an unvisited cell in the grid. Assume that out of the k cells visited by P, there were d visits to dangerous cells and $k-d$ visits to safe cells. Let us denote the threat probabilities of the dangerous cells that were visited along the path by $(p_1, ..., p_d)$. Thus, according to Eq. (7) the total weight of path P is:

$$\sum_{i\in(u_1,...,u_{k-1})} w_{i,i+1} = (k-d) + \sum_{i=1}^{d}\big[-D\cdot\log(1-p_i)+1\big] = k - D\sum_{i=1}^{d}\log(1-p_i) \tag{A.25}$$

The result is a sum of two expressions - the first is determined by the coverage path length (k) and the second is determined by the accumulated risk taken by the robot along the path.

From the correctness of Dijkstra's shortest paths algorithm, we know that the path P has a minimum weight in relation to all the other possible paths, i.e., every other path $P' = (v_1, ..., v_m)$ from the robot's current location $v_1 = u_1$ to an unvisited cell in the grid v_m satisfies:

$$\sum_{i\in(u_1,...,u_{k-1})} w_{i,i+1} \le \sum_{j\in(v_1,...,v_{m-1})} w_{j,j+1} \tag{A.26}$$

Assume that out of the m cells visited by P', there were e visits to dangerous cells, and the threat probabilities in these cells were $(p'_1, ..., p'_e)$. Thus, from equations (A.25) and (A.26) we get:

$$k - D\sum_{i=1}^{d}\log(1-p_i) \leq m - D\sum_{j=1}^{e}\log(1-p'_j) \tag{A.27}$$

First, we will show that if P and P' have the same length, then P is safer than or equally safe to P'. If P and P' have the same length, then $k = m$, and thus from Eq. (A.27) we have:

$$\sum_{i=1}^{d}\log(1-p_i) \geq \sum_{j=1}^{e}\log(1-p'_j) \tag{A.28}$$

Since the natural exponential is monotonically increasing function of its argument, the above expression is equivalent to:

$$\prod_{i=1}^{d}(1-p_i) \geq \prod_{j=1}^{e}(1-p'_j) \tag{A.29}$$

Hence, the probability to complete path P is greater or equal to the probability to complete path P'.

Second, we will show that if P and P' have the same accumulated risk, then P must be shorter than or have the same length as P'. If P and P' have the same accumulated risk, then:

$$\prod_{i=1}^{d}(1-p_i) = \prod_{j=1}^{e}(1-p'_j) \tag{A.30}$$

Taking log of both sides of the equation:

$$\sum_{i=1}^{d}\log(1-p_i) = \sum_{j=1}^{e}\log(1-p'_j) \tag{A.31}$$

Thus, from Eq. (A.27) we can conclude that $k \leq m$, i.e., P must be shorter than or have the same length as P'.

Lastly, when P and P' have both different lengths and different risks, then GAC will choose the path that minimizes the weighted sum of the length and the accumulated risk according to Eq. (A.25). The penalty term D, which is dependent upon the relation between the desired risk and path length (Eq. (8)), will determine whether a safer path will be favored over a shorter one or vice versa. $\square$