Safest Path Adversarial Coverage

Roi Yehoshua[1], Noa Agmon[1] and Gal A. Kaminka[1,2]
[1]Computer Science Department and [2]Gonda Brain Research Center
Bar Ilan University, Israel
{yehoshr1,agmon,galk}@cs.biu.ac.il

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. While most previous work concentrated on finding a solution that completes the coverage as quickly as possible, in this paper we consider a new version of the problem: *adversarial coverage*. Here, the robot operates in an environment that contains threats that might stop the robot. We introduce the problem of finding the safest adversarial coverage path, and present different optimization criteria for the evaluation of these paths. We show that finding an optimal solution to the safest coverage problem is NP-Complete. We therefore suggest two heuristic algorithms: STAC, a spanning-tree based coverage algorithm, and GSAC, which follows a greedy approach. These algorithms produce close to optimal solutions in polynomial time. 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.**

I. Introduction

In robotic coverage, a robot is required to visit every part of a given area as efficiently as possible ([1], [3], [4], [5], [7], [8]). Coverage has many applications in a multitude of domains, including search and rescue, mapping, and surveillance. The coverage problem is analogous to the traveling salesman problem, which is known to be $\mathcal{NP}$-complete [1]. However, it is possible to find polynomial-time solutions to the coverage problem that are close to optimal through heuristics and reductions (e.g., [1], [3], [5], [8]).

Almost 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, exploration of Mars, demining and surveillance of enemy forces in the battle field.

In the adversarial coverage problem [10], the target area contains locations with potential threats of harming the robot, in addition to obstacles which the robot cannot go through. The robot's task is to cover the *entire* target area as quickly as possible without being damaged by a threat point. In this paper we discuss the offline version of this problem, where a map with the locations of the potential threats is given in advance, therefore the coverage path of the robot can be determined prior to its movement. In addition, here we focus on the problem of finding the *safest* coverage path in an adversarial environment, i.e., we are only concerned about the survivability of the robot and not the coverage time. Nevertheless, the algorithms we propose will also try to minimize the coverage time, in cases where the robot's safety is not compromised. We will refer to this problem as the *Safest Coverage Path Problem.*

We formally define the safest coverage planning problem and show 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 GSAC (Greedy Safest 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. We provide optimality bounds on both algorithms, and show that while STAC tends to achieve higher expected coverage, GSAC produces coverage paths with lower accumulated risk.

II. Related Work

The problem of robot coverage has been extensively discussed in the literature (see [4] for a recent exhaustive survey). Grid-based coverage methods, such as we utilize here, use a representation of the environment decomposed into a collection of uniform grid cells (e.g., [3], [7]).

As a basis for our adversarial coverage planning algorithm, we chose to use the Spiral Spanning Tree Coverage (Spiral-STC) algorithm. This algorithm, introduced by Gabriely and Rimon [3], provides close-to-optimal coverage paths in a uniform grid based terrain. Spiral-STC assumes that a single robot is equipped with a square shaped tool of size D placed on grid. The grid is then coarsened such that each new cell is of size $2D \times 2D$, and a spanning tree is built over this new coarse grid. Then the robot follows the edges of this spanning tree, while covering each $2D$-cell internally. The main result is that Spiral-STC covers any planar grid in $O(n)$ time using a path whose length is at most $(n + m)D$. Here, n is the number of D-size cells and $m \leq n$ is the number of cells that share at least one point with the grid boundary.

The offline adversarial coverage problem was formally defined by us in a recent study [10]. There we proposed a simplistic heuristic algorithm that generates a coverage path which tries to minimize a cost function, which takes into account both the survivability of the robot and the coverage path length. However, the heuristic algorithm worked

Support in part by ISF grant #1511/12.

only for obstacle-free areas, and without any guarantees, or analysis of the problem complexity, in contrast to the novel algorithms and analysis suggested in this paper.

III. Safest Coverage Path Problem

In this section we will formally define the safest coverage path problem and discuss its complexity.

A. Problem Definition

We are given a map of a target area T, which contains obstacles and also points with threats, which may stop the robot. We assume that T can be decomposed into a regular square grid with n cells, whose size equals the size of the robot. Some cells in T contain threat points. Each threat point i is associated with a threat probability p_i, which measures the likelihood that the threat will stop the robot. We assume the robot can move continuously, in the four basic directions (up/down, left/right), and can locate itself within the work-area to within a specific cell. The robot's task is to plan a path through T such that every accessible free cell in T (including the threat points) is visited by the robot at least once. Refer to [10] for a more detailed description of the problem.

In this paper, two objective functions will be considered with respect to the safest coverage path problem:

1) Minimize the total accumulated risk along the coverage path (i.e., maximize the probability of covering the whole target area).
2) Maximize the coverage percentage of the target area before the robot is first hit (i.e., maximize the expected coverage percentage).

The second objective function has been used in [10], while the first one has not been considered before.

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:

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

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 will use the definition of expected coverage from [10]. Given a coverage path A, we denote the sequence of new cells discovered along A by $(b_1, ..., b_n)$, and the number of new cells visited by the robot until it is stopped by C_A. Furthermore, 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. Then, under the threat probability function p, the expected number of new cells that the robot visits can be expressed as:

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

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 threat point).

In this paper we focus on a special case of the safest coverage problem, where all the threat points are associated with the same threat probability p. We will refer to this case as the *Uniform-Threat Safest Coverage Problem*.

B. Problem Complexity

We now prove that the uniform-threat safest coverage path problem is $\mathcal{NP}$-hard for both objectives.

Definition 3.1: The Uniform-Threat Maximize Coverage Completeness Probability Problem (UMCP): Given a grid representation of a world that contains obstacles and threat points with uniform threat probability, find a coverage path of the grid that contains minimal number of threat points.

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

Proof: (Sketch). The $\mathcal{NP}$-hardness of the problem be shown by reduction from the Hamiltonian path problem on grid graphs, which is known to be $\mathcal{NP}$-complete [6]. ■

Definition 3.2: The Uniform-Threat Maximize Expected Coverage Problem (UMEC): Given a grid representation of a world that contains obstacles and threat points with uniform threat probability, find a coverage path with maximum expected coverage percentage.

Theorem 2: The UMEC problem is $\mathcal{NP}$-Hard.

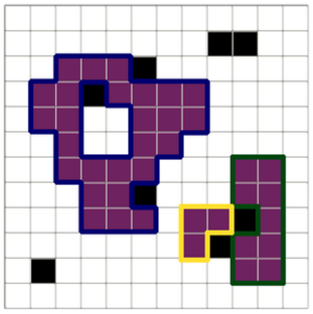
Given the proven hardness of the problem, it is natural to look for polynomial-time approximate solution algorithms. Two such algorithms are presented in sections IV and V.

IV. STAC Algorithm

The Spanning Tree Adversarial Coverage algorithm (STAC for short) uses a layered 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 all the dangerous cells, also 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.

Let us define a **safe area** as a connected subset of safe cells in the grid, while a **dangerous area** is defined as a connected subset of dangerous cells. Figure 1 shows an example of a grid containing two safe areas and three dangerous areas.

[1]Full proofs can be found in [9].


1. An example of a grid containing two safe areas and three dangerous areas. Obstacles are colored in black, dangerous cells are colored purple, and safe cells are colored white. The dangerous areas are outlined by blue, green and yellow thick lines, respectively.

For the coverage of each connected area, we use the Spiral-STC algorithm as described in [3], with a few changes. 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 area are covered, the robot can proceed to the next one. In addition, we addressed cases where the given area cannot be decomposed into 2D-size cells (for example, if the area is just a row of consecutive cells).

STAC uses a helper procedure called Create_Safe_Coverage_Path (CSCP for short) to create a safe coverage path for a specific type of cells (in our case, safe cells or dangerous cells). This procedure is composed of four main stages. First, it identifies the connected areas of the given cell type by running Depth First Search (DFS) on the graph induced from its input cells. Second, it finds a coverage path for each connected area by running the modified Spiral-STC algorithm. Third, it finds the safest route between each pair of connected areas. Finally, it finds a minimum risk route that connects all areas. The final coverage path returned by CSCP consists of the coverage paths for each area and the route that connects them. CSCP is fully described in [9].

The safest route between each pair of connected areas is found by running Dijkstra's shortest paths algorithm on the graph induced from the entire grid cells with the following edge weights:

$$w_{ij} = \begin{cases} 1 & \text{if cell } j \text{ contains a threat} \\ 1/n & \text{otherwise} \end{cases} \tag{3}$$

The weights represent the risks of traversing each edge in the graph. However, since we also want to take into account the path length, we set a small constant weight to edges that connect safe cells. This weight is chosen small enough so that only if there are two equally safe paths connecting two nodes, we will prefer the shorter one.

The minimum-risk route that connects all the areas is determined as follows. The algorithm creates a graph whose nodes represent the connected areas. The weight of the edge between two areas is determined by the length of their connecting path. We then run an approximation algorithm that solves TSP (Traveling Salesman Problem) on this graph 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 [2]).

STAC uses the CSCP procedure twice: to create a coverage path of all the safe cells in the grid, and then to create a coverage path of all the dangerous cells. In addition, it needs to find a connecting route between the last visited safe cell and the first cell in the dangerous areas, which is performed in step 7 of the algorithm.

Algorithm 1 Spanning_Tree_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: Create a new coverage path $P$
2: $S \leftarrow$ all the safe cells in $G$ that are reachable from $s$
3: $P_S \leftarrow$ Create_Safe_Coverage_Path($G$,$S$)
4: Add $P_S$ to $P$
5: $D \leftarrow$ all the dangerous cells in $G$ that are reachable from $s$
   and are not included in $P_S$
6: $P_D \leftarrow$ Create_Safe_Coverage_Path($G$,$D$)
7: Find a safest route between the last visited cell in $P_S$ and the
   first visited cell in $P_D$ by using Dijkstra's algorithm, and add
   this route to $P$
8: Add $P_D$ to $P$
9: return $P$
```

Analysis of the STAC algorithm

Lemma 4.1 (completeness): STAC creates a path that covers every free cell accessible from the starting cell s.

The next lemma gives the run-time guarantees of STAC.

Lemma 4.2: Let n be the total number of free cells accessible from the starting cell S, and a be the number of connected areas. Then STAC covers the given area in $O(a^2 n \lg n + a^3)$ time.

In the worst case, the number of connected areas is $a = \Theta(n)$, and in such case the time complexity of the algorithm is $\Theta(n^3 \lg n)$. However, in practice, $a \ll n$, and in such environments the algorithm's run-time is $O(n \lg n)$.

The following theorem establishes a bound on the number of dangerous cells in the coverage path generated by the STAC algorithm. We start with the following definitions.

Definition 4.1: **Boundary cells** are cells that share either a point or a segment with a cell containing an obstacle, or with the boundary of the area which they belong to.

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

Theorem 3: Let d be the total number of dangerous cells in the accessible grid. Let $b \leq d$ be the total number of dangerous boundary cells and $c \leq d$ the number of dangerous connecting cells. Then STAC covers the given grid using a path that contains $x \leq d + b + 2c$ dangerous cells.

Proof: (Sketch.) By theorem 1 in [3], the total number of cells revisits in Spiral-STC is bounded by the number of boundary cells in the work-area grid. Thus, the total number of visits to dangerous cells during the coverage of the dangerous areas is at most $d+b$. In addition, dangerous cells may be visited along the connecting route between different areas. The connecting route is found by the Christofides approximation to TSP [2]. This route does not traverse the same edge more than twice, thus the connecting dangerous cells cannot be visited more than twice. Hence, the coverage path generated by STAC contains at most $d+b+2c$ dangerous cells, where $b \leq d$ and $c \leq d$. ■

In practice, $b \ll d$ and $c \ll d$, and in such environments STAC generates paths whose number of dangerous cells is close to d.

The next theorem establishes a bound on the length of the coverage path generated by the STAC algorithm. Its proof is similar to the proof of Theorem 3.

Theorem 4: Let n be the total number of cells in the accessible grid. Let $m \leq n$ be the total number of boundary cells and $r \leq n$ the number of connecting cells. Then STAC covers the given grid using a path that contains $y \leq n + m + 2r$ cells.

V. GSAC Algorithm

The Greedy Safest Adversarial Coverage algorithm (GSAC for short) follows a greedy approach, where in each step it searches for the next unvisited cell, which has the safest route from the robot's current location. For that purpose, the algorithm runs Dijkstra's algorithm on the graph induced from the grid cells, in order to find a minimum weighted path between the robot's current position and all the other cells in the grid (see Algorithm 2).

The weight function used for the graph edges is the same as in Eq. (3). These weights represent the risks of traversing each edge, while also considering the path length.

Algorithm 2 Greedy_Safest_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: Create a new coverage path $P$
2: Build the graph $G_C$ whose nodes are the cells in $G$ and its edges connect neighboring cells in the grid $G$ with the weights as defined by formula 3
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 $s$ to each node in $G_C$
7:     $v \leftarrow$ the node in $G_C$ with the shortest 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 $P$
```

Analysis of the GSAC Algorithm

Lemma 5.1 (completeness): GSAC creates a path that covers every free cell accessible from the starting cell s.

Lemma 5.2: Let n be the total number of free cells accessible from the starting cell S. Then GSAC covers the given area in $O(n^2 \lg n)$ time.

The following theorem establishes a bound on the number of dangerous cells in the coverage path generated by GSAC.

Theorem 5: Let d be the total number of dangerous cells in the accessible grid. Then GSAC covers the given grid using a path that contains at most $2d$ dangerous cells.

Proof: Consider the outgoing edges from a dangerous cell c. The first time the algorithm traverses such an edge, it is guaranteed that it will cover all the cells that are accessible from that edge before it returns back to c, since all the other cells in the grid will have longer paths from the current position of the robot. Thus, each edge connected to c is traversed at most once in each direction. Consequently, there are at most $2d$ edges that touch dangerous cells along the coverage path. Hence, the coverage path contains at most $2d$ dangerous cells. ■

Note that in some environments the bound of $2d$ is tight. However, as we will demonstrate in the next section, in practice GSAC generates paths whose number of dangerous cells is close to d. The next theorem establishes a bound on the length of the coverage path generated by the GSAC algorithm. Its proof is similar to the proof of Theorem 5.

Theorem 6: Let n be the total number of cells in the accessible grid. Then GSAC covers the given grid using a path that contains at most $4n$ cells.

VI. Experiment Results

We use a specific map to illustrate the operation of both algorithms and compare their performance (Section VI-A). Then, we report on the statistical analysis based on multiple randomly generated maps with varying parameters, such as map size, number of obstacles, number of threat points, etc. (Section VI-B).

A. An Example Run

We considered a target area consisting of 30×30 square cells, out of which 20% contain obstacles, 15% contain threats and the other 65% are free and safe. The probability of being hit by a threat was set to an arbitrary value of 0.15 (changing this value does not affect the coverage paths generated by the algorithms). The obstacles are randomly scattered across the map, while the threats are confined to 8 randomly chosen contiguous areas (some of them had common borders so they merged into a single area).

See Figure 2 for the description of the map. Obstacles are represented by black cells, dangerous cells are colored purple and safe cells are colored white. The starting position of the robot is cell $(1, 1)$. We ran both STAC and GSAC on this map. The coverage path generated by STAC is denoted by a solid line, while the coverage path generated by GSAC is denoted by a dotted line.

The expected coverage obtained by STAC and GSAC was 76.68% and 55.31%, respectively. The number of times STAC has visited a dangerous cell was 210, while the number of times GAC has visited a dangerous cell was 158 (the map contained 141 dangerous cells). Thus, STAC

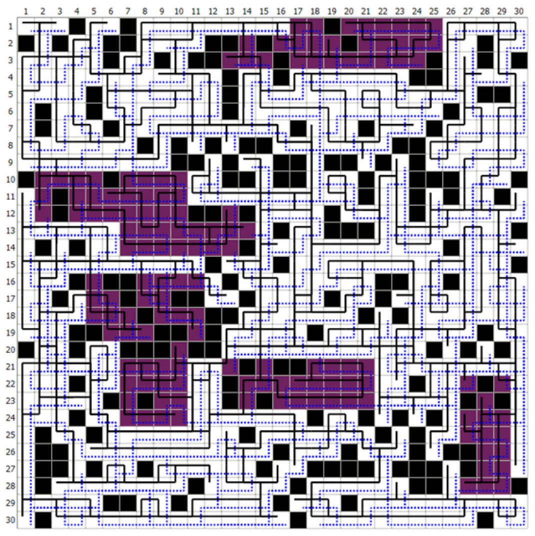


Fig. 2. An example map. STAC's coverage path is denoted by a black solid line, while GSAC's coverage path is denoted by a blue dotted line.

achieved better expected coverage, but its coverage path is less likely to complete. Note that for the uniform threats case, maximizing the coverage completion probability (Eq. (1)) is equivalent to minimizing dangerous cell visits. Thus, to make computations easier, we used the number of threat visits instead.

Examining the coverage paths created by the algorithms reveals the difference in their behavior. The grid contains two large safe areas. Both algorithms start with covering the upper large safe area. When STAC finishes covering this area, it finds the safest path to the bottom safe area, via the single dangerous cell in row 21, column 13. On the other hand, when GSAC finishes covering the upper safe area, it starts scanning the dangerous area beginning with the cell in row 10, column 7, and exits this area only when it reaches a cell on its border (the cell in (11, 1)). Thus, the greedy algorithm visits 15 dangerous cells before moving to scan the next safe area, while STAC visits only one dangerous cell.

We can also learn from these figures why GSAC suffers less from repetitive coverage of dangerous cells than STAC. Let us observe the coverage of the upper-right dangerous area, that starts in row 2, column 16. In GSAC this area is covered in two phases. First, GSAC covers cells located on the left side of this area, then it continues to the dangerous area to the left of it (that starts in row 2, column 14), and then it returns back to cover the cells on the right side of this area. This way, the number of cells revisits in this area is significantly reduced to only one repetition. On the other hand, STAC must cover this dangerous area completely before moving to the next one. Covering this area in one phase incurs a high number of repetitions (11 revisits of cells). The fact that GSAC can jump between different areas allows it to make less repetitions.

B. *Controlled Experiments*

In order to compare between the performance of the two algorithms, we examined them in various types of environments and settings. We have examined maps with randomly scattered threat points vs. maps with contiguous dangerous areas (whose locations were also randomly chosen), and the same for obstacles (maps with randomly scattered obstacles vs. maps with contiguous areas of obstacles), i.e., four types of environments. Here we report on three types of environments, which offered the most significant insights. All results are averaged on 50 randomized maps.

Randomly scattered threat points and obstacles. Figure 3 compares the expected coverage percentage, number of threats visits and the coverage path length for different threat ratios in the range between 0.0 and 0.5. In all experiments, we used a map size of 20×20 and the ratio of obstacles was 20%. The locations of the threat points and the obstacles were randomly chosen.

As anticipated, the expected coverage percentage decreases while the number of threat visits increases as we add more threats to the map. The coverage path length increases until the threat ratio reaches the level of around 27% and then it starts decreasing. The reason for this is that when the majority of the cells in the map are of the same type (either safe or dangerous), there is less repetitive coverage in the transition between different types of areas.

We see that for low numbers of threats, the two algorithms have similar results. When the number of threats in the map gets higher, STAC achieves a better expected coverage than GSAC, albeit the difference is less than 5%. In this scenario, there is no statistically significant advantage to STAC over GSAC, since when the threats are scattered across the map, most of the safe cells are connected to each other, i.e. there is no need to pass through a large dangerous area in order to move between safe cells, so both algorithms can cover most of the safe cells before covering dangerous cells.

Regarding the coverage path length, GSAC consistently produces shorter coverage paths than STAC, as it does not waste repetitive coverage on the transitions between the different areas. It just moves to the nearest unvisited cell from the robot's current position with the safest path. Moreover, the inner coverage of each area in GSAC is more efficient than STAC (at the expense of computation time).

Contiguous areas of threats and randomly scattered obstacles. When the threats are confined to contiguous areas and not scattered across the map, we can observe a clear advantage to STAC in terms of expected coverage. Figure 4 shows the expected coverage percentage and number of threats visits for a varying number of dangerous areas between 2 and 40. We used maps with size 20×20, the ratio of obstacles was 20% and the ratio of dangerous cells was 20%. In general, as there are more dangerous areas, both algorithms achieve lower expected coverage but also less repetitive visits of dangerous cells. The reason is that when there are more dangerous areas, there are more disconnected safe cells, thus the robot is able to cover less safe cells before moving to cover the dangerous areas.

When the number of dangerous areas is around 10, STAC generates coverage paths with a significantly better expected coverage than GSAC (about 10% difference, which is statistically significant; one-tailed t-test $p = 0.00047$). This

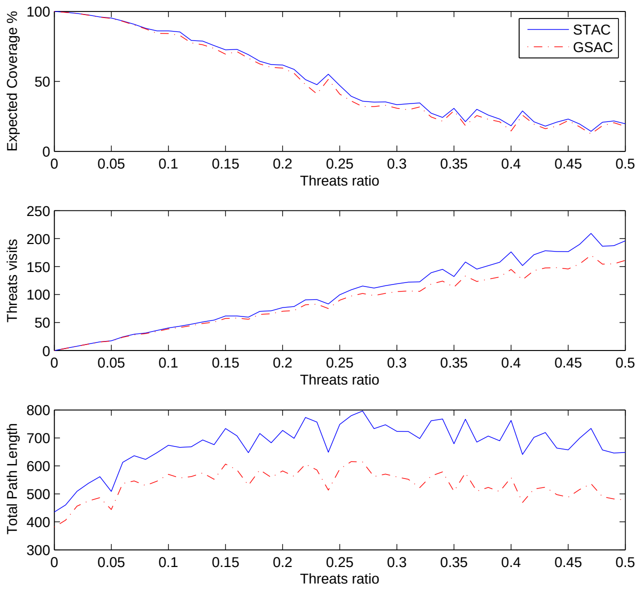


Fig. 3. Expected coverage percentage, number of threats visits and coverage path length for different threat ratios in environments with randomly scattered threat points and obstacles.

can be explained by the fact that for such a number of dangerous areas, GSAC has to pass through relatively large dangerous areas when moving between the safe areas. When the number of dangerous areas is lower, most of the safe cells stay connected so GSAC can cover them without getting stuck in a dangerous area. When the number is higher, each dangerous area contains small number of cells, so crossing a dangerous area has less impact on the expected coverage.

while the number of threat visits is always lower for GSAC, the difference in the number of threat visits between the two algorithms becomes smaller when the number of dangerous areas gets higher. This is due to the fact that STAC is less effective in covering large connected areas than smaller ones (since there are more boundary cells).

Contiguous areas of obstacles and randomly scattered threats. When the obstacles are concentrated in large contiguous areas, the expected coverage in both algorithms is significantly higher than in environments where the obstacles are scattered. For example, STAC obtained an expected coverage of 73.9% when there were only 4 obstacles areas, while the expected coverage was 61.62% when the obstacles were randomly scattered. This can be explained by the fact that when the threats are randomly scattered, it is easier for both algorithms to find a safe connecting route between the safe areas when the obstacles are contiguous. On the other hand, the number of threats visits is not affected by the obstacle distribution (around 75 for STAC, 70 for GSAC).

VII. Discussion and Future Research

In this paper we have presented the safest path adversarial coverage problem. First, we have suggested two optimization criteria for the evaluation of safest coverage paths. Next, we proposed two polynomial-time algorithms that try to meet these criteria, STAC and GSAC. We have provided

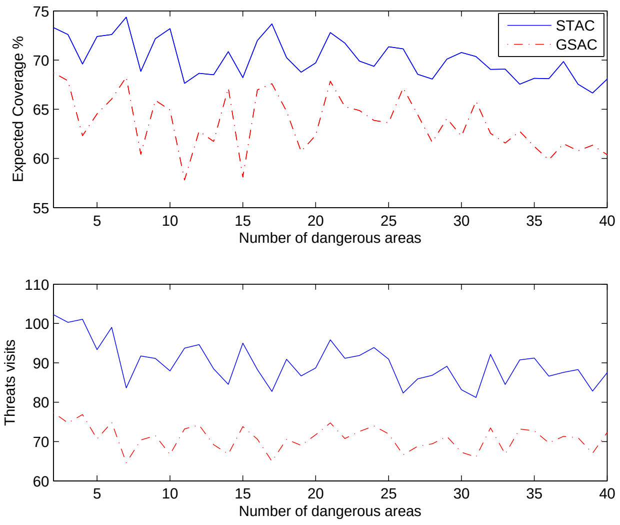


Fig. 4. Expected coverage percentage and number of threats visits for different numbers of dangerous areas in environments with contiguous areas of threats and randomly scattered obstacles.

optimality bounds on the total risk involved in the coverage paths generated by these algorithms and on their coverage time. Experiments in various environments showed that STAC tends to achieve higher expected coverage, while the coverage paths generated by GSAC have lower accumulated risk. We have also examined the coverage paths generated by the algorithms and compared their structures.

In the future we plan to consider maps with multiple threat levels (and not just safe and dangerous cells). We also would like to extend the algorithms to multi-robot systems.

References

[1] E. M. Arkin, S. P. Fekete, and J. S. Mitchell, "Approximation algorithms for lawn mowing and milling," *Computational Geometry*, vol. 17, no. 1, pp. 25–50, 2000.

[2] N. Christofides, "Worst-case analysis of a new heuristic for the travelling salesman problem," Graduate School of Industrial Administration, Carnegie-Mellon University, Tech. Rep. 388, 1976.

[3] Y. Gabriely and E. Rimon, "Competitive on-line coverage of grid environments by a mobile robot," *Computational Geometry*, vol. 24, no. 3, pp. 197–224, 2003.

[4] E. Galceran and M. Carreras, "A survey on coverage path planning for robotics," *Robotics and Autonomous Systems*, 2013.

[5] M. Grigni, E. Koutsoupias, and C. Papadimitriou, "An approximation scheme for planar graph tsp," in *36th IEEE Annual Symposium on Foundations of Computer Science*, 1995, pp. 640–645.

[6] A. Itai, C. H. Papadimitriou, and J. L. Szwarcfiter, "Hamilton paths in grid graphs," *SIAM Journal on Computing*, pp. 676–686, 1982.

[7] C. Luo, S. X. Yang, D. A. Stacey, and J. C. Jofriet, "A solution to vicinity problem of obstacles in complete coverage path planning," in *ICRA*, 2002, pp. 612–617.

[8] A. Xu, C. Viriyasuthee, and I. Rekleitis, "Optimal complete terrain coverage using an unmanned aerial vehicle," in *ICRA*, 2011.

[9] R. Yehoshua, N. Agmon, and G. A. Kaminka, "Safest path adversarial coverage: Proofs and algorithm details," Bar Ilan University, Computer Science Department, SMART Group, Tech. Rep. SMART 2014/01, available at http://www.cs.biu.ac.il/~yehoshr1/, 2014.

[10] R. Yehoshua, N. Agmon, and G. A. Kaminka, "Robotic adversarial coverage: Introduction and preliminary results," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS'13)*, 2013, pp. 6000–6005.