2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
Congress Center Hamburg
Sept 28 - Oct 2, 2015. Hamburg, Germany

Online Robotic Adversarial Coverage

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

Abstract— **In the robotic coverage problem, a robot is required to visit every point of a given area using the shortest possible path. In a recently introduced version of the problem, *adversarial coverage*, the covering robot operates in an environment that contains threats that might stop it. Previous studies of this problem dealt with finding optimal strategies for the coverage, that minimize both the coverage time and the probability that the robot will be stopped before completing the coverage. However, these studies assumed that a map of the environment, which includes the specific locations of the threats, is given to the robot in advance. In this paper, we deal with the *online* version of the problem, in which the covering robot has no a-priori knowledge of the environment, and thus has to use real-time sensor measurements in order to detect the threats. We employ a frontier-based coverage strategy that determines the best frontier to be visited by taking into account both the cost of moving to the frontier and the safety of the region that is reachable from it. We also examine the effect of the robot's sensing capabilities on the expected coverage percentage. Finally, we compare the performance of the online algorithm to its offline counterparts under various environmental conditions.**

I. Introduction

In robotic coverage, a robot is required to visit every part of a given area using the most efficient path possible ([2], [3], [4], [6], [8], [16]). Coverage has many applications in a multitude of domains, including search and rescue, mapping, and surveillance.

In a recently introduced version of the problem, *adversarial coverage* ([13], [14], [15], [11]), the target area contains locations with potential threats that might harm the robot, in addition to obstacles which the robot cannot go through. The robot's task is to cover the *entire* target area (including the threat locations) as quickly as possible while minimizing the probability that it will be stopped before completing the coverage. This version of the problem has many real-world applications in various domains, from performing coverage missions in hazardous environments such as nuclear power plants or the surface of Mars, to surveillance of enemy forces in the battle field and field demining.

All previous works of the adversarial coverage problem dealt with the offline version of the problem, in which a map of the environment is given to the robot in advance, therefore the coverage path of the robot can be determined prior to its movement. In this paper we discuss the *online* version of this problem, in which the robot has no map or a-priori information about the environment. Rather, the robot must collect information about obstacles and threat points in the environment during the coverage process. We will refer to this problem as the *Online Adversarial Coverage Problem*.

We describe an online sensor based algorithm for covering a target area, using the safest possible coverage path. Our approach is based on the detection of frontiers, regions on the border between covered space and unexplored space. By moving to new frontiers, a mobile robot can extend its map into new territory until the entire environment has been explored. We describe a method for choosing the next frontier to navigate to, that takes into account the risk of moving to that frontier, its distance from the robot's current location and the safety of the region that can be reached from that frontier. We provide both theoretical and empirical evaluation of the algorithm.

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, similar to what is utilized in this paper, use a representation of the environment decomposed into a collection of uniform grid cells (e.g., [3], [6]).

Online coverage algorithms known in the literature include the Spiral-STC (Spiral Spanning Tree Coverage) [3], which provides close-to-optimal coverage paths in a uniform grid based terrain. Shivashankar et al. [8] describe four strategies for generating cost-effective coverage plans in real time for unknown environments. Their best strategy returned plans with less than 6% redundant coverage in real environments, however without any theoretical guarantees.

A related problem to online coverage is the problem of exploring an unknown territory. In the exploration problem an autonomous robot has to completely scan an unknown environment with its sensors. In contrast, area coverage requires physical sweep of a tool (or the robot itself) over every point of a given work-area.

The most common approach to exploration is based on *frontiers* (e.g., [10], [1], [5]), which are segments that separate known (explored) regions from unknown regions. The frontier-based approach incrementally constructs a global occupancy map of the environment. The map is analyzed to locate the frontiers between the free and unknown space, and exploration proceeds in the direction of the closest frontier. Here, we also use frontiers but as segments that separate

978-1-4799-9993-4/15/$31.00 ©2015 IEEE

covered regions (i.e., regions that have been physically visited by the robot and not only detected by its sensors) from uncovered regions.

The offline adversarial coverage problem was formally defined in our recent study [13], in which a simplistic heuristic algorithm was proposed for generating a coverage path aiming at minimizing 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. In a follow-up paper [14] 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 while STAC tends to achieve higher expected coverage, GSAC produces shorter coverage paths with lower accumulated risk. In [11] we have built a more sophisticated model of the adversary, in which it can choose the best locations of the threat points, such that the probability of stopping the covering robot is maximized. All these previous works dealt with the offline version of the problem, in contrast to this paper.

Another related problem to online adversarial coverage is the Canadian Traveler Problem (CTP) [7]. The goal of this problem is to minimize the cost of reaching a target in a weighted graph, where some of the edges are unreliable and may have been removed from the graph. However, that an edge has been removed is only revealed to the traveler when she/he reaches one of the edge's endpoints. In contrast, here the robot (the traveler) must visit every node in the graph (some of them may stop the robot).

III. Problem Formulation

We provide here a summary of the adversarial coverage problem formulation. Refer to [13] for a more detailed description of the problem. The target area T 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, whose size equals the size of the robot. Some cells in T contain threat points (we refer to them as dangerous cells). Each threat point i is associated with a threat probability p_i, which measures the likelihood that the threat will stop the robot. 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 dangerous cells) is visited by the robot at least once. The objective of the robot is to cover as many cells as possible before getting hit by a threat, i.e., our goal is to find a coverage path of T with maximum expected coverage. We will use the definition of expected coverage from [13]. 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:

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

In the online version of the problem, the robot has no a-priori knowledge of the environment. We assume that the robot is equipped with a sensor, such as a panoramic camera, that provides a 360-degree view of its surroundings, and can detect obstacles and threats that exist around the robot. If the given sensor has less than a 360-degree field of view, we assume that after reaching a new cell, the robot turns around in order to take a 360-degree view of its surroundings.

Let us denote by r the maximum range of the robot's sensor. Thus, by our assumption, the robot can detect all the threats and obstacles that lie inside a circle with a radius r and a center in the current position of the robot. Note that even if $r > 1$, the robot still needs to physically visit the cells it observes. The sensor just provides more information about the environment, and allows the robot to better plan its coverage path.

Denote by $\mathcal{M}_r(c)$ the set of cells that belong to the Moore neighborhood of cell c with range r. The Moore neighborhood of cell $c = (x_0, y_0)$ with range r is defined as [9]:

$$\mathcal{M}_r(x_0, y_0) = \{(x, y) : |x - x_0| \leq r, |y - y_0| \leq r\} \tag{2}$$

Figure 1 shows the Moore neighborhood with range $r = 2$ of the cell located at the center of the map (cell (5, 5)). The number of cells in a Moore neighborhood with range r is: $(2r + 1)^2 - 1$.

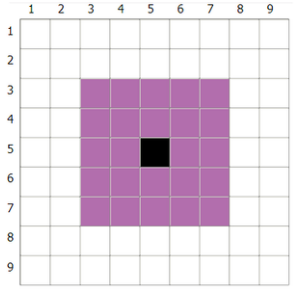

Fig. 1. An example for a Moore neighborhood with range $r = 2$.

Using this definition of Moore neighborhood, our assumption is that if the robot is located at cell c, then it can detect all the obstacles and threats that exist in the Moore neighborhood of c with range r.

Additionally, we denote by $\mathcal{V}(c)$ the set of cells that belong to the Von Neumann neighborhood of cell c. The Von Neumann neighborhood of cell c comprises the four cells orthogonally surrounding c, i.e., the four cells with Manhattan distance 1 from c. If the robot is located at cell c, then it can move to any free cell that belongs to the Von Neumann Neighborhood of c.

Now, we define a **frontier** as a segment that separates covered regions from uncovered regions. Formally, a frontier is a set of unvisited cells that each have at least one visited neighbor (in the Von Neumann neighborhood of the cell).

IV. Online Adversarial Coverage Algorithm

The Online Adversarial Coverage algorithm (OAC for short) is shown in Algorithm 1. The algorithm maintains a list of frontier cells, denoted by F. The list is initialized with the free cells in the Von Neumann's neighborhood of the robot's starting cell (lines 3–4). The main loop in OnlineCoverage continues until the frontier list is empty. In each iteration of the loop (lines 7–13), first the cost (risk and distance) of moving to each frontier and the number of safe cells that can be reached from each frontier are computed (by calling ComputeCostToFrontiers and ComputeReachableSafeCells, respectively). Next, one of the frontiers in the list is selected as the next destination for the robot to visit (by calling ChooseNextFrontier). The robot moves to the selected frontier using the safest possible path from its current location (line 10). Lastly, that frontier is removed from F, and all the unvisited free cells in its Von Neumann neighborhood are added to F (in the function UpdateFrontier). Eventually, every free cell accessible from the robot's starting cell will be added to the frontier list and thus visited by the robot (as proven by lemma **??**).

We now explain the operation of each of the helper functions. The function ComputeCostToFrontiers finds the safest paths from the robot's current location to each of the frontiers in F. It begins by creating the graph $G = (V, E)$ induced from the visited and the frontier cells (line 2). In this graph, each visited or frontier cell is represented by a vertex in V and vertices that represent adjacent cells in the grid are connected by an edge in E. The idea is that the path to any given frontier must go through cells that have already been visited or other frontiers (since the frontiers lie on the boundary between the visited and unvisited regions). Then the algorithm runs Dijkstra's shortest paths algorithm on the graph G (line 3), using the following edge weights:

$$w_{ij} = \begin{cases} -\log(1-p_j) & \text{if cell } j \text{ contains a threat} \\ \frac{-\log(1-p_{min})}{n} & \text{otherwise} \end{cases} \quad (3)$$

This weight function ensures that edges that target more dangerous cells (i.e., cells with higher p_j) are assigned a higher weight (the reason for using the logarithmic function here will become clearer when we formally prove the correctness of this function in theorem 3). However, since we also want to take into account the path length, we set a small constant cost to edges that target safe cells. The weight assigned to an edge that targets a safe cell is equal to $1/n$ of the weight of an edge that targets the least dangerous cell on the map (n is the grid size). This way, the cost of visiting any dangerous cell is greater than the cost of visiting all the safe cells in the grid. Thus, only when there are two equally

Algorithm 1 Online_Adversarial_Coverage

Sensors: a position and orientation sensor, an obstacle and threat detection sensor with range r
Input: a starting cell s
Global data structures: F - set of frontier cells

```
1:  function OnlineCoverage(s)
2:      Mark s as visited
3:      F ← {s}
4:      UpdateFrontier(s)
5:      c ← s                          ▷ c is the robot's current cell
6:      while F ≠ ∅ do
7:          ComputeCostToFrontiers(c)
8:          ComputeReachableSafeCells()
9:          f ← ChooseNextFrontier()
10:         Move to f using f.path
11:         Mark f as visited
12:         UpdateFrontier(f)
13:         c ← f

1:  function ComputeCostToFrontiers(c)
2:      Build a graph G that consists of the visited cells and
        the frontiers, and its edge weights are defined by Eq. (3)
3:      Run Dijkstra on G starting from c
4:      for each frontier f ∈ F do
5:          f.path ← the safest path from c to f
6:          f.P ← the probability of reaching f
7:          f.L ← |f.path|

1:  function ComputeReachableSafeCells
2:      for each frontier f ∈ F do
3:          Build a graph G that consists of all the free and
            safe cells in M_r(f) and f itself
4:          Run BFS on G from f
5:          f.N ← the number of unvisited vertices in G
            that are reachable from f

1:  function ChooseNextFrontier
2:      P_max ← 0
3:      for each frontier f ∈ F do
4:          if f.P > P_max then
5:              P_max ← f.P
6:              f* ← f
7:          else if f.P = P_max then
8:              if P_max = 1 then
9:                  if f.L < f*.L then
10:                     f* ← f
11:             else if f.N > f*.N then
12:                 f* ← f
13:             else if f.N = f*.N and f.L < f*.L then
14:                 f* ← f
15:     return f*

1:  function UpdateFrontier(f)
2:      Remove f from F
3:      for each cell c ∈ V(f) do
4:          if c is an unvisited free cell and c ∉ F then
5:              Add c to F
```

safe paths between two vertices in the graph, the algorithm will prefer the shorter one.

After running Dijkstra's shortest path algorithm on the graph, the safest path to each frontier is found by traversing the shortest paths tree from the robot's current cell. The path, its probability to complete and its length are stored as fields in the frontier's data structure.

The function COMPUTEREACHABLESAFECELLS computes the number of safe cells that can be reached from each frontier, without going through any threat point. For each frontier, it first builds a graph G that contains the frontier and all the free safe cells in its Moore's neighborhood (this neighborhood is visible to the robot's sensor when the robot is located at that frontier). Then, it runs BFS on G in order to find all the cells in the neighborhood that are accessible from that frontier (i.e., there is a path from the frontier to them that does not go through any obstacle or a threat point), and stores their count in the frontier's data structure.

The function CHOOSENEXTFRONTIER is responsible for choosing the next frontier to be visited by the robot. It evaluates each candidate frontier f, according to the following components:

1) the probability of reaching f
2) the length of the path leading to f
3) the number of unvisited safe cells that can be reached from f, without going through any threat point

First, the algorithm prefers the frontier with the safest paths from the robot's current location. If there are two frontiers with equally safe paths from the robot's current location, then we distinguish between two cases:

Case 1. The paths to both frontiers are completely safe (i.e., their probability to complete is 1). In this case, the algorithm chooses the frontier with the minimum path length. In this case we do not care about the safety of the regions that can be reached from the frontiers, because the robot can move between the frontiers safely.

Case 2. The paths to both frontiers are not completely safe. In this case, the algorithm prefers the frontier that enables the robot to reach a safer region. If the regions reachable from these frontiers are equally safe, then the algorithm again prefers the frontier with the minimum path length.

The emergent behavior of the robot following this algorithm is that it will first visit all the safe cells that can be reached from its current location. After visiting all the reachable safe cells, all the frontiers become dangerous (i.e., contain threat points). The robot then will move to the safest frontier, i.e., the frontier with minimum threat probability. If there is more than one such frontier, the robot will move to the frontier that enables it to reach the safest region, i.e., a region with the highest number of unvisited safe cells.

V. ANALYSIS OF THE OAC ALGORITHM

In this section we consider several properties of the OAC algorithm. We first prove that the algorithm creates a complete coverage path, i.e., a path that visits every free cell of the environment that is reachable from the robot's starting location.

Theorem 1 (completeness): The OAC algorithm covers every free cell accessible from the starting cell s. [1]

We now analyze the run-time and memory requirement of the OAC algorithm.

Theorem 2: Let n be the total number of free cells accessible from the starting cell s. Then OAC covers the given area in $O(n^2 \lg n)$ time using $O(n)$ memory.

We now prove the correctness of the function COMPUTECOSTTOFRONTIERS, i.e., that it finds the safest path from the robot's current location to each frontier.

Theorem 3: (correctness) The function COMPUTECOSTTOFRONTIERS finds the safest path from the robot's current location to each frontier. If there is more than one safest path to a frontier, it finds the shorter one.

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

Lemma 1: 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 OAC.

Using lemma 1, we can now prove the following theorem.

Theorem 4: 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 coverage path generated by OAC 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 $\prod_{i=1}^{l}(1-p_j)^{4n_i(l-i+1)}$.

VI. EMPIRICAL EVALUATION

We first use a specific map to illustrate the operation of the OAC algorithm. The sample map consists of 10×10 square cells, out of which 25% contain obstacles, 25% contain threats and the other 50% are free and safe. The probability of being hit by a threat was set to 10% (changing this value does not affect the coverage path generated by the algorithm; it only affects the scaling of the results). The locations of the threat points and the obstacles were randomly chosen.

Figure 2 shows the safest coverage paths found by OAC on the sample map for two different sensing ranges: $r = 1$ (upper figure) and $r = 3$ (bottom figure). The robot starts the coverage in cell (1,1). The numbers in the cells indicate their visit order (e.g., the cell with number 1 was the first cell visited by the robot).

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

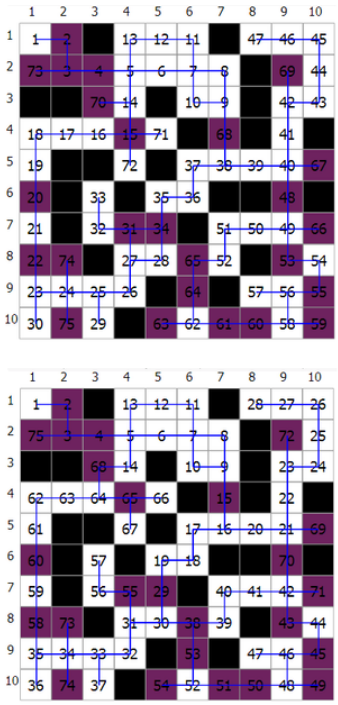

Fig. 2. OAC's generated coverage paths on a sample map. The upper figure shows OAC's coverage path when the sensing range was $r = 1$, while the bottom figure shows OAC's coverage path when $r = 3$.

When $r = 1$, the expected coverage of the path was 43.25%, while its length was 156. On the other hand, when $r = 3$, the expected coverage of the path was 46.36%, while its length was 171. The larger sensing radius enables the robot to choose frontiers that lead to larger areas of safe cells before moving to visit dangerous areas. To illustrate this point, let us examine the first frontier that was chosen differently between the two coverage paths. As can be seen in the figure, up until the 14th visited cell, both coverage paths are identical. At that point, the robot is located at cell (3, 4) and there are 4 possible frontier cells the algorithm can choose from: $f_1 = (2, 1), f_2 = (3, 3), f_3 = (4, 4)$, and $f_4 = (4, 7)$. The path to f_1 is more dangerous than the paths to the other three frontiers, since it involves passing through two other dangerous cells, thus it cannot be chosen as the next frontier for both sensing ranges. However, when the sensing range is $r = 1$, the number of unvisited safe cells that can be reached from each frontier and can be detected by the robot's sensor are: $f_2.N = 2, f_3.N = 3, f_4.N = 3$. Since $f_3.N = f_4.N$, the algorithm chooses f_3, which is the nearest frontier to the robot's location. On the other hand, when the robot's sensing range increases to $r = 3$, the number of unvisited safe cells that are reachable from each frontier and can be detected by the robot's sensor are: $f_2.N = 4, f_3.N = 5, f_4.N = 13$, thus f_4 is chosen as the next frontier to be visited by the robot. Since moving to this frontier enables the robot to visit many more safe cells before moving to a dangerous cell, the expected coverage percentage increases.

We now describe the results of running the online algorithm on 500 random maps of size 30×30, with varying sensor ranges between 1 and 15. In all the experiments, the obstacles ratio and the threats ratio were both set to 20% of the cells and the threat probability in the dangerous cells was set to 5% (changing the absolute value of this probability does not change the coverage path generated by the algorithm; it only affects the scaling of the results). The locations of the threat points and the obstacles were randomly chosen.

Figure 3 shows the results. As can be seen from the graph, the expected coverage keeps increasing as the range of the sensor increases. However, when the sensor's range reaches $r = 9$, the expected coverage does not improve any more. This can be explained by the observation that under the given map settings (20% obstacles and 20% threat points), increasing the size of the visible neighborhood of a frontier to more than 9 cells does not allow the robot to reach more safe cells from that frontier without going through any threat point. Therefore, the decision of which frontier to choose is not affected by the sensor's range.

As for the coverage path length, it initially increases from 1829 when $r = 1$ to 1884 when $r = 2$ and then remains around 1875. The initial increase in the path length can be explained by the fact that when the robot has better sensing capabilities, it occasionally prefers to move to a frontier that is further away from its current location, but enables it to reach a safer area. As we have seen, for higher values of r, the decision of which frontier to choose is not affected by r, thus the coverage path length is also not affected by the sensor's range.

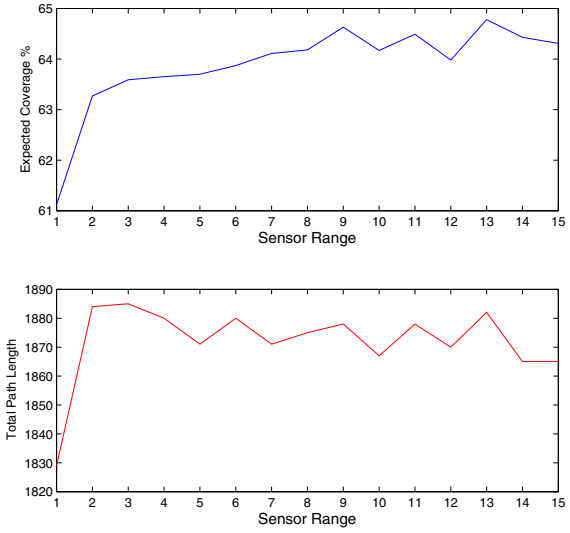

Fig. 3. Expected coverage percentage and coverage path length for different sensor ranges.

We now compare the performance of the online adversarial coverage with a sensor range that is equal to the map size (which has the same effect as providing the robot with the

entire map in advance) to the two algorithms for solving the offline version of the adversarial coverage problem mentioned in [14], namely, STAC and GSAC. Figure 4 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 these experiments, we used a map size of 20×20 and the sensor's range was set to $r = 20$ (the maximum possible sensor range for this map size). The ratio of the obstacles was set to 20%.

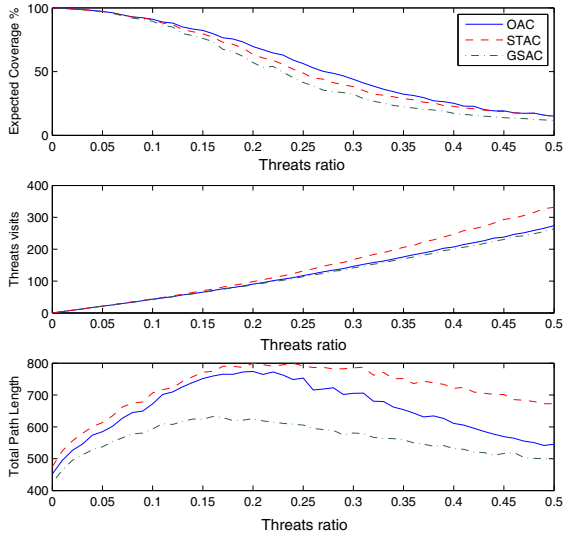

Fig. 4. Expected coverage percentage, number of threats visits and coverage path length for different threat ratios.

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 20% 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.

OAC consistently achieves higher expected coverage percentages than both STAC and GSAC (up to 8% increase relative to STAC and up to 14.8% relative to GSAC). The results are statistically significant (a one-tailed t-test between OAC and STAC yields $p = 4.19 \cdot 10^{-10}$, and between OAC and GSAC yields $p = 2.11 \cdot 10^{-14}$). OAC's number of threats visits and total path length are higher than GSAC's, but still lower than STAC's. These results can be explained by the fact that OAC's main goal is to cover as many of the safe cells before visiting dangerous cells, and this often implies repetitive coverage in the transition between different frontiers.

VII. Discussion and Future Research

In this paper we have presented the online version of the adversarial coverage problem. We have shown how a frontier-based approach can take advantage of the robot's sensing capabilities in order to select the best location to be visited by the robot. We have also shown that the same approach can significantly improve the expected coverage percentage that can be achieved in the offline version of the problem.

In the future we plan to generalize the online algorithm so that it can generate coverage paths that meet any desired levels of risk and coverage time, and not only the safest coverage path. We also intend to extend the algorithm for multi-robot systems; using multiple robots for coverage has the potential for more efficient coverage and greater robustness.

References

[1] F. Amigoni and V. Caglioti, "An information-based exploration strategy for environment mapping with mobile robots," *Robotics and Autonomous Systems*, vol. 58, no. 5, pp. 684–699, 2010.

[2] 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.

[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. Keidar and G. A. Kaminka, "Efficient frontier detection for robot exploration," *The International Journal of Robotics Research*, vol. 33, no. 2, pp. 215–236, 2013.

[6] 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.

[7] C. H. Papadimitriou and M. Yannakakis, "Shortest paths without a map," in *Automata, Languages and Programming*. Springer, 1989, pp. 610–620.

[8] V. Shivashankar, R. Jain, U. Kuter, and D. S. Nau, "Real-time planning for covering an initially-unknown spatial environment," in *Proc. of the Twenty-Fourth International Florida Artificial Intelligence Research Society Conference (FLAIRS-11)*, 2011.

[9] E. W. Weisstein, "Moore neighborhood," *From MathWorld–A Wolfram Web Resource. http://mathworld.wolfram.com/MooreNeighborhood.html*, 2005.

[10] B. Yamauchi, "A frontier-based approach for autonomous exploration," in *IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA-97)*, 1997, pp. 146–151.

[11] R. Yehoshua and N. Agmon, "Adversarial modeling in the robotic coverage problem," in *International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-15)*, 2015.

[12] ——, "Online adversarial coverage: Proofs," Bar Ilan University, Computer Science Department, SMART Group, Tech. Rep. SMART 2015/02, available at http://www.cs.biu.ac.il/~yehoshr1/, 2015.

[13] 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.

[14] ——, "Safest path adversarial coverage," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS-14)*, 2014, pp. 3027–3032.

[15] ——, "Frontier-based RTDP: A new approach to solving the robotic adversarial coverage problem," in *International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-15)*, 2015, to appear.

[16] A. Zelinsky, R. A. Jarvis, J. Byrne, and S. Yuta, "Planning paths of complete coverage of an unstructured environment by a mobile robot," in *International Conference on Advanced Robotics (ICAR)*, vol. 13, 1993, pp. 533–538.