Learning Generalizable Patrolling Strategies through Domain Randomization of Attacker Behaviors

Carlos Diaz Alvarenga Nicola Basilico Stefano Carpin

Abstract— **Graph-patrolling problems in the adversarial domain typically embed models and assumptions about how hostile events, from which an environment must be protected, are generated at a specific time and location. Relying upon such attacker models prevents algorithms from synthesizing strategies that can generalize in different settings, providing good performance under different and uncertain scenarios. In this paper, we propose a first method to deal with adversarial patrolling using a data driven approach. We cast the problem in an RL setting where the reward function is based on the ability to neutralize attacks that can follow an unknown strategy and that, hence, can be viewed as a black box component. We apply a policy gradient framework for optimizing action probabilities under such a reward model showing how effective patrolling strategies can be obtained from repeated attack-defense interactions between a patrolling agent and an attacker. Our results show that the data driven patroller can effectively provide protection against multiple, diverse attacker behaviors.**

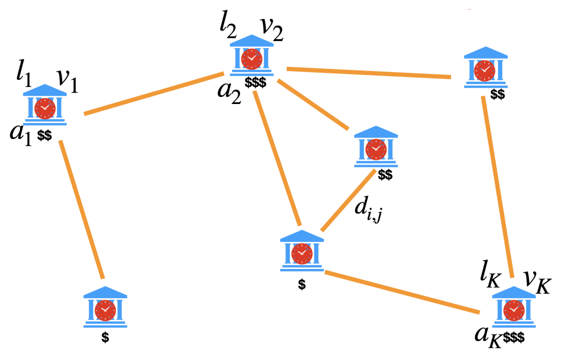

Fig. 1: Example of a patrolling instance. Each vertex has a value v and an attack time a representing the effort needed to compromise it. A patroller moving from vertex v_i to v_j will spend time $d_{i,j}$.

I. Introduction

Surveillance is an application domain that is getting notable attention from research communities and companies alike. In this domain, several solutions at the intersection between Artificial Intelligence and Robotics have been devised in the last years [21]. Robotic Patrolling is a term used to refer to a family of problems in a domain where the general objective is to deploy one or more autonomous robots in some environment to guard a set of assets against adversarial actions that can be detrimental to their integrity or value.

Many formulations exist for robotic patrolling [6], and in this work we focus on the approach where the arrangement of assets to be defended is represented by a graph. Each vertex has a value associated with the corresponding asset, and weighted edges encode the temporal cost necessary to move between locations. The agent tasked with defending the assets is dubbed *patroller*, while the agent trying to compromise the assets is called *attacker*. We consider a common set of rules (described in more detail later) where the patroller moves from vertex to vertex, and the attacker aims at entering a vertex when the patroller is not occupying it. We shall focus on the single patroller/single attacker case, although the problem with multiple agents in either category has been investigated, too [3]. Figure 1 sketches a typical instance of this setting.

Security games [26] are an example of methods where the patroller and the attacker are assumed to be rational and fully informed agents. The mathematical model takes the form of a game, where concepts like Nash or Stackelberg equilibrium are used to determine the optimal patrolling strategy. The mainstream approach to model this type of attacker is to optimize against a cost function encoding the attacker's behavior. The most popular indicators used in graph-based patrolling settings to define such cost functions are the *idleness*, namely the time elapsed since the last patroller's visits to a given vertex and the *values* of the assets present in the environment. Once a cost function is defined, deriving its exact or approximate optimal solutions provides a *patrolling strategy*.

In our past works [2], [7] we followed this approach to design strategies with the objective of defending the graph from an attacker that may be learning through observations of the patrolling activities. Key to our former works was some background knowledge about how the attacker would decide if, when, or where to attack the graph. While these patrolling strategies may be successful against the specific attacker they assume to face, they require modeling the behavior inside some optimization framework and typically fail when the attacker is different from the one considered. Being unsuitable to face novel attackers means also that these methods are not applicable in settings where the attacker's behavior is unknown or not fixed (a feature of great practical relevance). Some works in the literature studied Bayesian frameworks [17] where the attacker is known (e.g., it strikes the most valuable asset), but its parameters (e.g., its preferences) are uncertain. However, the definition of a method for patrolling that could defend against dynamic and unknown attackers is still an open problem.

Starting from these observations, we address the above

N. Basilico is with the Department of Computer Science, University of Milan, Milan, Italy. C. Diaz Alvarenga and S. Carpin are with the Department of Computer Science and Engineering, University of California, Merced, CA, USA.

challenges by exploiting a reinforcement learning (RL) approach [14] where the patrolling strategy is learned by repeated interaction with what, from the point of view of the patroller, is a black box running some arbitrary, non–stationary, and randomized adversarial behavior. Our findings show how exploiting a technique based on Proximal Policy Optimization (PPO) [24] a single patrolling algorithm can learn to deliver effective strategies against several heterogeneous attackers. As a key contribution, such a method can provide fair performance not only against single arbitrary attackers but also against many attackers at the same time – a feature not addressed by other approaches. In our patrolling RL setting, we achieve this by training the PPO agent with Domain Randomization (DR) [30] over different attacker behaviors.

II. Related Work

A multi-agent formulation of the patrolling problem has been introduced in [9] where *idleness* is proposed as a cost metric. It measures the time passed since the last surveillance inspection of any given location. Idleness is a natural proxy for the effectiveness of a patrolling strategy since it is inversely proportional to the frequency of visits. Approaches built around the optimization of idleness (typically the average or the maximum over the environment) are typically combined with other metrics and domain-dependent constraints. In [27], for example, robots' sensing capabilities are confronted with a cost function that grows in each node if this is not falling in the range of any robot and decreases otherwise. The approaches presented in [18] and [23] are examples where idleness is coupled with the cost of establishing situational awareness between the robots by communication, either with teammates or with a base station. Optimizing this metric is typically hard. Other works explored the adoption of optimization paradigms for dealing with maximum idleness constraints. For example, [16] proposed approximated and heuristic algorithms for the problem of computing the minimum number of robots under idleness constraints [4], and devised approximated methods [1].

Supervised learning has been evaluated to synthesize decentralized multi-agent patrolling strategies by leveraging historical data of surveillance tours computed by a reference ideal method [15]. Despite showing promise, the need for a training dataset makes this type of approach difficult to deploy. Reinforcement learning (RL) provides a more natural environment-driven paradigm. A first RL formulation has been proposed in [22] where idleness plays a central role in shaping the reward function providing feedback to an agent moving in the environment. This idea has been reused under the latest advances in RL by some recent works. In [20] and [13] deep reinforcement learning is adopted to learn patrolling policies that maximize a reward defined as a variable interest metric on each node. Such an interest increases, at a node-dependent rate, as long as the node is not patrolled by the agent. Deep R-learning is exploited to solve a persistent area coverage problem in [25]. The task is characterized by the need to sense stochastic events in the environment as soon as they appear. This problem, sharing in its definition some of the features of the patrolling problem, is solved by adopting a reward function based on the average detection rate of the events. The main difference with our work is that events are not modeled as the product of some adversarial process, a defining feature for the class of adversarial patrolling problems called Security Games [8].

Works in this last field use attacker models best responding to patrolling strategies [12], [19]. This task might be cast in an online framework where the patroller executes a strategy for some time and then receives feedback [5]. When compared to ours, this class of approaches, despite adopting an online setting, models the problem under a game theoretical perspective according to which attackers often comply with some level of rationality. Recent advancements in so-called "green security games" proposed the use of deep Q-learning for computing optimal strategies. In [31] deep RL is exploited to approximate best responses in an iterative resolution of a patrolling game where real-time observations suggesting how the attacker is operating are exploited. In our work, we do not necessarily assume rationality for the attackers and hence we do not rely on an underlying game-theoretical formulation.

III. Patrolling Setting

We adopt a graph-based representation of the environment where K target locations must be protected through visits by a single patrolling robot. Targets are denoted as $\{l_1, ..., l_k\}$. Their topological layout is described by a weighted undirected graph $G = (V, E, d)$ where $V = \{l_1, ..., l_k\}$ and $(l_i, l_j) \in E$ indicates that the patroller can move from l_i to l_j (or vice versa) in a time equal to d_{ij}. We assume that this graph is connected and we will always work on its transitive closure, that is if $(l_i, l_j) \notin E$ we add it and set the corresponding d_{ij} to the length of the shortest path between l_i and l_j in the original graph. Each target l_i is characterized by a *value* v_i measuring its importance and an *attack time* a_i expressing the temporal cost needed to compromise it. In our experiments, values are drawn uniformly within the range [1, 2]. Attack times are assumed to be derived as follows:

$$a_i = k \cdot U(lb, ub)$$
$$lb = m - \left(\frac{m}{L}\right) \qquad ub = m + \left(\frac{m}{L}\right)$$

where: L denotes a positive real number, m denotes the average travel distance on the undirected graph, the function $U(\cdot)$ denotes a random uniform sample on the interval $[lb, ub]$, and k scales the resulting random sample by a positive real number. To generate non-trivial instances, attack times cannot be either too large (capture would be too easy) or too small (attacks would be too hard to defend against.) The above formulas capture this requirement by uniformly drawing their values from an interval (whose width is controlled by the parameter L) centered at the patroller's expected distance to be traveled by any two targets.

The patrolling mission unfolds in discrete time steps. In each of them, the patroller protects the graph by moving from one target location to the next, but may also decide to stay at the current vertex for another time step. If the patroller does decide to stay at the current vertex, the travel time is *not* zero. Instead, the patroller travels some random time along an edge and returns to the vertex it left from. It is assumed, in the model, that if the patroller travels to a vertex where an attack is taking place then the attack is neutralized and the attacker captured.

For generality, all attackers, before a game starts and after any re-spawn, derive their deadline by randomly sampling from a pre-defined real–valued set.

IV. Deep reinforcement learning and patrolling

A. Problem Formulation

We model our robotic patrolling setting as a Markov Decision Process (MDP) and then apply Deep Reinforcement Learning (DRL) to synthesize a patrolling strategy. A Markov decision process is defined by the tuple (S, A, P, R, γ) where: S denotes the set of states, A refers to the set of available actions, P denotes the transition probabilities for the environment, R is the reward function and γ the discount factor.

Considering that in our setting the environment is modeled as an undirected graph, we introduce a state representation with the aim of encoding key aspects of the patrolling strategy realization over a finite temporal history. Specifically, we define a state as a $(M + n + 1)$-dimensional vector combining several patrolling-related aspects of the graph and the patroller's past decisions:

$$[\ t_i, p_{i-M}, ..., p_{i-1}, p_i, s_0, ..., s_n\ ]$$

The first entry in the state representation (t_i) is the current time measured as the sum of the d_{ij}'s up to the ith transition in the patrolling mission. The next M entries in the state vector represent the last M vertices that the patroller visited. Finally, the last n entries denote the current (or instantaneous) *idleness* of each vertex, a common optimization criteria used in literature [22]. Idleness is defined as the time since the patroller's last visit to a particular vertex. Hence, the currently occupied vertex has always an idleness of 0 that starts increasing as long as the patroller leaves the vertex and does not return to it. The state aims to both capture the current coverage of the graph and any cycles relevant to patrolling the graph. Each idleness value encodes information about which vertices are visited more often (resulting in a lower idleness value) and which vertices are being ignored (larger idleness value). Furthermore, the history of past vertices is included to enable the patroller to make correlations between graph cycles, defined as a sequence of traveled vertices where the first and last vertex are the same, and coverage of the graph.

The use of MDP–based representations is a widespread practice in patrolling [10]. Our formulation, however, adopts a richer state description by combining several attributes. The action set A for every state is the entire set of vertices, $\{l_1, ..., l_k\}$, denoting which target location the patroller will move to next. We assume deterministic transitions and use a discount factor of 0.99 for all experiments. Moreover, M in all experiments equals k, or the number of vertices.

We adopt a reward function that rewards the patroller *only when an attack is neutralized*. In other words, every action that the patroller takes receives a reward signal of 0, however, when the state transition results in the capture of the attacker the patroller receives some positive real reward, r. After a parameter sweep, we settled on a reward value of 5 for captures. A limitation of this reward function is the sparseness of the signal and from our experience generally requires that the patroller captures the attacker within a reasonable number of transitions, in order to learn. In our experiments, this was handled by tuning the deadline of the attackers. A modest deadline (about 10 transitions) allows for the algorithm to produce adequate results during training.

The proposed function encourages the patroller to extend the episode for as long as possible by rewarding it for repeatedly capturing the attacker indefinitely. We opted for not including a negative reward signal at the end of the episode when the patroller is unable to counter the attack because experiments showed that including the negative signal had almost no effect on the performance. We postulate this is due to the fact that as the patrolling agent begins to capture more and more attackers, the accumulation of positive rewards outweighs the single negative signal that could be sent to the agent during learning at the end of a failing episode (the patroller can capture many attackers in one episode, but can only lose the game once).

B. Resolution with Proximal Policy Optimization

Policy gradients [29] is a family of methods for finding an optimal policy of an MDP where the agent's policy itself is parameterized by a vector θ and the optimization happens in the policy space $\pi(a|s, \theta)$. To use policy gradient methods in our setting we define a policy as $\pi(l_j|s_t, \theta)$ i.e., the policy receives as input the state at time t and outputs the probability of selecting l_j as the next vertex to visit.

These policy gradient methods can be extended to Actor-Critic [28] methods where not only is the agent's policy parameterized (Actor) but also a value function approximation (Critic) is used to discriminate between good and bad performing actions. By adopting Proximal Policy Optimization (PPO) [24] we approximate the objective by a first-order surrogate problem we will refer to as *Clipped–PPO* agent. The agent obtains its name from the objective function used to optimize the actor:

$$L(\theta) = \mathbf{E}[\min(f_t(\theta)A_t, clip(f_t(\theta), 1 - \varepsilon, 1 + \varepsilon)A_t)]$$

where

$$f_t = \frac{\pi(l_t|s_t, \theta_t)}{\pi_{old}(l_t|s_t, \theta_t)},$$

A_t is an estimator of the advantage function at time step t, and ε is some small real valued scalar.

In an adversarial patrolling setting where the attacker is able to make observations of the patroller's strategy, the defender must exhibit some non-determinism or else risk making the prediction problem too simple for the attacker.

This means that the optimal strategy for the patroller will be a *mixed strategy*. Stochastic policies are a natural way to model mixed strategies over an undirected graph leading to our choice of using the Clipped Proximal Policy Optimization agent (Clipped–PPO) from [24].

V. Modeling the Attacker's Behavior

An important departure in our work from others is the explicit modeling of attacker behavior. In general, one can have different attacker models, each representing a potential attacker to defend against. We propose six different attacker models and discuss their corresponding rationale. All attacker models draw their deadline randomly at the beginning of each game from the discrete set of values $\{100, 150, 200, 300\}$ in all experiments. The six attackers we consider are described in the following.

Max Idleness Attacker (MIA): Once the deadline is reached, the MIA will attack the vertex with the largest idleness. The MIA model will target locations that have been left vulnerable for more since a vertex with larger idleness means that the patroller has ignored that particular vertex for longer. Generally, a larger deadline makes this model (and other idleness models) stronger since it is able to observe the patroller for a longer period of time. This model assumes that the attacker can observe all vertices in the graph.

Average Idleness Attacker (AIA): Once the deadline is achieved this model will penetrate the vertex with the largest average idleness. Average idleness is measured as:

$$\frac{1}{t}\sum_{y=0}^{y=t} I_y$$

where I_y is the instantaneous idleness at transition y. The rationale here follows from the MIA model in that this model will penetrate more vulnerable (i.e., larger idleness) targets. However, this model will not consider instantaneous idleness but average idleness up to the current time step. This attacker observes a history of idleness and attacks the vertex that has been visited with the least regularity. Like MIA, AIA assumes full observability of the graph.

Idle Subset Attacker (MISA): This attacker model works exactly like MIA, but it only focuses on a subset of the vertices and completely ignores the vertices not in its purview. This model's rationale follows exactly from the MIA model, but also it captures scenarios where the attacker is not interested in all the target locations or cannot observe the entire graph. Whether because of preference or restriction, this attacker models intruders who have curtailed their total attention. In all subsequent experiments, the MISA attacker focuses on a fixed subset consisting of half of the target locations.

Scaled Idleness Attacker (SIA): This model attacks the vertex with the largest value of the product between the idleness vector and the vector of vertex values once the deadline is reached. This attacker uses the idleness information but also incorporates a preference for target locations with higher values. This model emulates intruders that are not interested in vertices that are being ignored by the patroller if these have low values.

Max Value Attacker (MVA): Once the deadline is reached, the Max Value Attacker always attacks the vertex with the largest value. Vertex values are generated once for a particular graph and remain static. This MVA model implements cases where the attacker is only interested in the largest payout and thus repeatedly tries for the target with the most assets. This model's observational capabilities are limited to the maximum value target location and is unaware of the patroller location when it is not at the maximum value vertex.

Preference Attacker (PA): Once the deadline is achieved this model chooses a vertex to attack according to a probability distribution over the vertices. This model does not take into consideration idleness or value. As the distribution approaches a uniform probability the problem becomes more difficult for the patroller. The PA model is a general model that can represent any preference that the attacker might have over the vertices and models an intruder who has the ability to infiltrate any target location but cannot observe the patroller's movements. All experiments presented here (where $|V| = 10$) were done with a distribution of 0.86 probability on one vertex and the remaining 0.14 distributed non-uniformly among the other vertices.

VI. Training, Domain Randomization and Results

All graphs are created by randomly placing points on a grid of size 50 by 50; It is important to note that the attackers can be divided into categories: those that make an attack based on some function of the graph idleness induced by the patroller, and those that do not. We can expect that the policies that will yield large rewards against the idleness group will not perform well when compared to the non–idleness group since the objectives are not correlated.

Before training considering multiple attackers, we assess whether the Clipped–PPO agent is able to learn against individual attacker models. Figure 2 shows the learning curves for the Clipped–PPO for such a scenario.

A single architecture was used to generate all of the curves. It consists of an actor and value network both with two hidden layers. For both networks, the first hidden layer contains 200 units, and the second hidden layer 100 units. Training procedures and agent implementations are built upon the TF–Agents library [11]. An epoch consists of running 30 game simulations in parallel and then optimizing both actor and critic networks according to the loss functions and training loop presented in [24]. The learning rate starts at a value of $1 \cdot 10^{-4}$ and is annealed according to an exponential decay function. Rewards and observations were normalized and for Clipped–PPO an ε value of 0.01 was used. From the slope and final values of individual curves, it is evident that some of the attacker models are easier for the Clipped–PPO agent to learn against than others. For example, the Max Value Attacker curve (in purple) shows that the patroller quickly learns a good policy for defending the graph. However, the policy that the agent settles on may

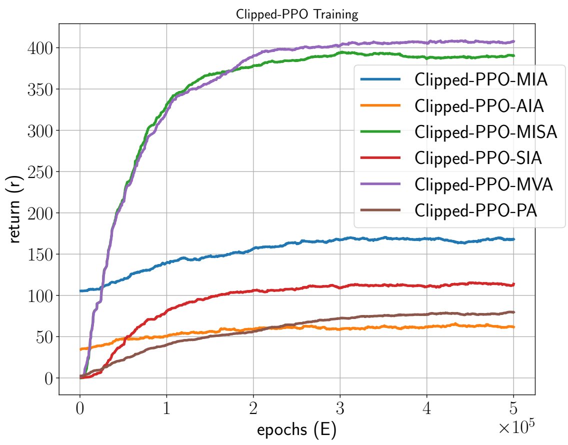

Fig. 2: Learning curves for the Clipped-PPO agent against the different attacker models. $|V| = 10$ and all curves were smoothed using a sliding window average. The x-axis represents the number of epochs trained and the y-axis shows the returned cumulative reward. Some attacker models are easier to learn against than others as evidenced by the slope and final value of the distinct curves.

not be robust to other attacker models (corroborated later empirically).

Next, we present the Domain Randomization (DR) [30] training procedure to generalize across attacker models. The approach at training time exposes the RL agent to all of the available attacker models instead of just a single model. Tables I and II show the results for the case of DR training, where before the start of any game (or episode) during training an attacker model is selected by sampling uniformly over all the available models. The first seven rows of Table I form a confusion matrix for the different Clipped–PPO agents, while the last row shows the performance of the Domain Randomization (DR) Clipped–PPO agent.

	MIA	AIA	MISA	SIA	MVA	PA
Clipped-PPO-MIA	**198**	107	98	52	1.0	3.4
Clipped-PPO-AIA	45	**101**	48	3.0	0.0	2.9
Clipped-PPO-MISA	0.3	0.2	**394**	0.9	4.3	0.3
Clipped-PPO-SIA	8.6	7.3	3.4	**83**	0.6	8.2
Clipped-PPO-MVA	0.5	0.4	1.5	0.0	**214.3**	0.2
Clipped-PPO-PA	0.3	0.3	0.0	0.0	0.0	**71.6**
MI-P	**17.3**	3.9	3.1	5.1	0.4	0.9
DR Clipped-PPO	78	79	310	**312**	247	11

TABLE I: Confusion matrix for a graph with $|V| = 10$. An entry in the table is generated by first training a Clipped-PPO agent exclusively against some attacker model then deploying it against different attackers. Results are averaged over 5,000 games.

Each entry is a testing scenario where a Clipped–PPO agent is deployed against a different attacker model. The table corroborates the expectation that an agent trained exclusively on a single attacker model will perform well against the attacker it was trained on but will also generalize badly to other attacker models. Row "Clipped–PPO–MVA" clearly shows that while the agent can learn to protect well against the MVA the resulting policy leaves the patroller vulnerable if the attacker changes strategies. DR Clipped–PPO, however, performs well across the attacker models and

	Average Performance	Worst Case
Clipped-PPO-MIA	77	1.0
Clipped-PPO-AIA	33	0.0
Clipped-PPO-MISA	67	0.2
Clipped-PPO-SIA	19	0.6
Clipped-PPO-MVA	36	0.0
Clipped-PPO-PA	12	0.0
MI-P	5	0.4
DR Clipped-PPO	**173**	**11**

TABLE II: Results presented here are some statistics gathered from Table I. Domain randomization helps the Clipped-PPO patroller generalize over the attacker models. As shown here, the DR Clipped-PPO agent performs better on average when facing any of the attacker models and at the same time has a better worst case return.

Table II substantiates the claim that the DR training will converge to policies that are robust to changing attacker strategies.

For completeness, we also compare our method against a heuristic strategy that represents an established class of approaches to the patrolling problem (see Section II for a discussion of idleness optimization techniques). At every turn, the *Maximum Idleness Patroller* (MI-P) decides to move to the vertex with the largest instantaneous idleness. In this way, the MI-P approximates a schedule that will minimize the average idleness induced over the graph. As can be seen from Table I, the MI-P patroller performs its best when playing against the idleness–based attacker models. Our Clipped–PPO–MIA agent does better by more than a factor of 10 meaning it learns ways to manipulate the idleness so as to capture the attacker. The MI-P meanwhile does not respond or react to the attacker.

DR Clipped–PPO was able to learn a patrolling strategy that can cope against what from the perspective of the patroller is an unknown and non–stationary attack–generation process. Particularly significant is the fact that the attacker behaviors we combined exhibit substantially different dynamics. Idleness attackers (MIA, AIA, MISA) exploit observations of the patrolling strategy's realization. Value–related ones (MVA, PA) follow a set of predetermined and arbitrary preferences, which depend on the environment, not on the patroller, and the remaining one (SIA) combines idleness and values. In traditional patrolling applications, these dynamics always result in conflicting objectives to be optimized. The results obtained with DR Clipped–PPO show how our method is a first promising step to overcome this limitation and build robust patrolling agents.

A. Ablation Study

Our state representation (and input to our function approximators) can be partitioned into the tuple (t, P, S) where: t is a real–valued scalar representing the current time, P is a vector of size K denoting the last K vertices visited, and S is also a vector of size K denoting the instantaneous idleness of each vertex. Building our state representation equates to concatenating $t, P,$ and S. As discussed in Section IV, the P vector is a history of the previous actions taken by the patrolling agent which helps the agent identify cycles and relevant patrolling patterns. The S vector (also called idleness vector) while also containing information related to past actions identifies areas of the environment that are under–visited. To corroborate the individual usefulness of S and P we study the training performance of the Clipped–PPO agents: first by removing only the history vector P from the state and then by removing only the idleness vector S from the state. For all experiments, the size of vector P is equal to the number of vertices in the graph. For brevity, we focus on two attacker models that are representative of our attacker library: the Max Idleness Attacker (MIA) and the Preference Attacker (PA).

Figures 3 and 4 show the training performances under different state representations against MIA and PA, respectively. Focusing on Figure 3 (MIA) first, we see that removing the idleness component from the state diminishes the performance of the RL agent more than when solely removing the history S. Intuitively, this makes sense as the RL patroller is inferring correlations between the idleness component in the state and the attacker's decisions. Removing one or the other, however, does have a negative impact on the RL agents' training performance. Figure 4 (PA) demonstrates a similar pattern wherein removing the idleness information significantly harms the training performance of the RL patroller. From both charts, we may deduce that the idleness component in our state representation provides vital information for the patroller during the learning process. The idleness vector provides ongoing information about which vertices have been visited most recently, thus also capturing information about past actions. From both figures, we see that the actual history of actions alone is not sufficient for learning to patrol the graph, but as is the case with Figure 3 its addition can improve performance. Note that removing both the components S and P would leave the state vector as a single scalar and thus we did not perform any experiments as such.

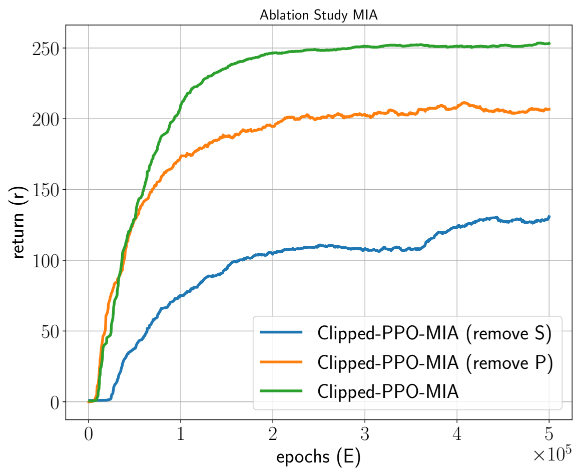

Fig. 3: Ablation study of the state representation when training Clipped–PPO against the Maximum Idleness Attacker. The orange curve represents the performance after removing the history of actions, P, from the state and the blue curve shows the performance after removing the idleness vector, S from the state. Removing either results in a decreased performance.

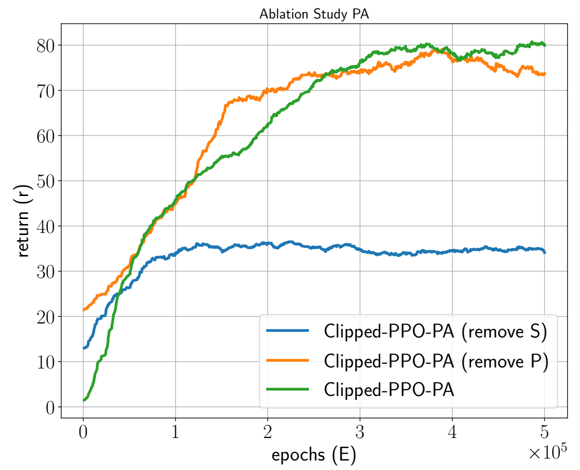

Fig. 4: Ablation study of the state representation when training Clipped–PPO against the Preference Attacker. The orange curve represents the performance after removing the history of actions, P, from the state and the blue curve shows the performance after removing the idleness vector, S from the state. PA uses the same distribution as described in section V.

VII. Conclusions

In this paper, we proposed a method to deal with adversarial patrolling using deep reinforcement learning. We cast the problem in an RL setting where the reward function is based on the realization of attacks that can follow arbitrary logic that is unknown to the patroller. Our main contribution is the combination of a Proximal Policy Optimization agent and Domain Randomization training techniques to generate patrollers that are robust to changing attacker strategies. Key to the method is changing attacker models before the start of every episode which effectively exposes the RL patroller to different MDPs during training. This method pushes the reinforcement learning algorithm (PPO) to converge to a sort of *average* policy that is able to generalize across different environments. To the best of our knowledge, this is the first work that applies a framework and presents significant results against a mixture of very distinct attacker behaviors. Our ablation study revealed that including a history of past actions in the state is less important than including the current instantaneous idleness, yet it is needed. Future directions include the application of graph neural networks to the actor and critic networks and multi-patroller settings where a team of patrollers must cooperate to patrol an environment. This involves studying different network architectures for the graph neural network and applying Centralized Training and Decentralized Execution (CTDE) methods to handle the multi-robot setting. With respect to the GNN, we could obtain a model that does not need re-training for every new graph instance since the new network could handle variable graph size input. Furthermore, the CTDE technique will encourage agents to spread out so as to cover the entire graph and avoid redundancies.

References

[1] P. Afshani, M. Berg, K. Buchin, J. Gao, M. Löffler, A. Nayyeri, B. Raichel, R. Sarkar, H. Wang, and H. Yang. Approximation algorithms for multi-robot patrol-scheduling with min-max latency. In *International Workshop on the Algorithmic Foundations of Robotics*, pages 107–123. Springer, 2020.

[2] C. Diaz Alvarenga, N. Basilico, and S. Carpin. Delayed and time-variant patrolling strategies against attackers with local observation capabilities. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4869–4876, 2019.

[3] C. Diaz Alvarenga, N. Basilico, and S. Carpin. Multirobot patrolling against adaptive opponents with limited information. In *Proceedings of the IEEE Conference on Robotics and Automation*, pages 2486–2492, 2020.

[4] A. B. Asghar, S. L. Smith, and S. Sundaram. Multi-robot routing for persistent monitoring with latency constraints. In *2019 American Control Conference (ACC)*, pages 2620–2625. IEEE, 2019.

[5] M. Balcan, A. Blum, N. Haghtalab, and A. D. Procaccia. Commitment without regrets: Online learning in stackelberg security games. In *Proceedings of the sixteenth ACM conference on economics and computation*, pages 61–78, 2015.

[6] N. Basilico. Recent trends in robotic patrolling. *Current Robotics Reports*, pages 1–12, 2022.

[7] N. Basilico and S. Carpin. Balancing unpredictability and coverage in adversarial patrolling settings. In M. Morales, L. Tapia, G. Sánchez-Ante, and S. Hutchinson, editors, *Algorithmic Foundations of Robotics XIII*, pages 762–777. Springer International Publishing, 2020.

[8] N. Basilico, N. Gatti, and F. Amigoni. Patrolling security games: Definition and algorithms for solving large instances with single patroller and single intruder. *Artificial intelligence*, 184:78–123, 2012.

[9] Y. Chevaleyre. Theoretical analysis of the multi-agent patrolling problem. In *Proceedings. IEEE/WIC/ACM International Conference on Intelligent Agent Technology, 2004.(IAT 2004).*, pages 302–308, 2004.

[10] X. Duan and F. Bullo. Markov chain–based stochastic strategies for robotic surveillance. *Annual Review of Control, Robotics, and Autonomous Systems*, 4:243–264, 2021.

[11] S. Guadarrama, A. Korattikara, O. Ramirez, P. Castro, E. Holly, S. Fishman, K. Wang, E. Gonina, N. Wu, E. Kokiopoulou, L. Sbaiz, J. Smith, G. Bartok, J. Berent, C. Harris, V. Vanhoucke, and E. Brevdo. TF-Agents: A library for reinforcement learning in tensorflow. https://github.com/tensorflow/agents, 2018. [Online; accessed 25-June-2021].

[12] N. Haghtalab, F. Fang, T. H. Nguyen, A. Sinha, A. D. Procaccia, and M. Tambe. Three strategies to success: Learning adversary models in security games. 2016.

[13] S. Y. Luis, D. G. Reina, and S. L. Marín. A deep reinforcement learning approach for the patrolling problem of water resources through autonomous surface vehicles: The ypacarai lake case. *IEEE Access*, 8:204076–204093, 2020.

[14] K. Mnih, V.and Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[15] M. Othmani-Guibourg, A. El Fallah-Seghrouchni, and J. Farges. Path generation with lstm recurrent neural networks in the context of the multi-agent patrolling. In *2018 IEEE 30th International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 430–437. IEEE, 2018.

[16] J. M. Palacios-Gasós, E. Montijano, C. Sagues, and S. Llorente. Multi-robot persistent coverage using branch and bound. In *2016 American Control Conference (ACC)*, pages 5697–5702. IEEE, 2016.

[17] P. Paruchuri, J. P. Pearce, J. Marecki, M. Tambe, F. Ordonez, and S. Kraus. Playing games for security: An efficient exact algorithm for solving bayesian stackelberg games. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 2*, pages 895–902, 2008.

[18] F. Pasqualetti, A. Franchi, and F. Bullo. On cooperative patrolling: Optimal trajectories, complexity analysis, and approximation algorithms. *IEEE Transactions on Robotics*, 28(3):592–606, 2012.

[19] A. Perrault, B. Wilder, E. Ewing, A. Mate, B. Dilkina, and M. Tambe. End-to-end game-focused learning of adversary behavior in security games. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 1378–1386, 2020.

[20] C. Piciarelli and G. L. Foresti. Drone patrolling with reinforcement learning. In *Proceedings of the 13th International Conference on Distributed Smart Cameras*, pages 1–6, 2019.

[21] Y. Rizk, M. Awad, and E. W. Tunstel. Cooperative heterogeneous multi-robot systems: A survey. *ACM Computing Surveys (CSUR)*, 52(2):1–31, 2019.

[22] H. Santana, G. Ramalho, V. Corruble, and B. Ratitch. Multi-agent patrolling with reinforcement learning. In *Autonomous Agents and Multiagent Systems, International Joint Conference on*, volume 4, pages 1122–1129. IEEE Computer Society, 2004.

[23] J. Scherer and B. Rinner. Multi-uav surveillance with minimum information idleness and latency constraints. *IEEE Robotics and Automation Letters*, 5(3):4812–4819, 2020.

[24] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[25] R. Shah, Y. Jiang, J. Hart, and P. Stone. Deep r-learning for continual area sweeping. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5542–5547. IEEE, 2020.

[26] A. Sinha, F. Fang, B. An, C. Kiekintveld, and M. Tambe. Stackelberg security games: Looking beyond a decade of success. IJCAI, 2018.

[27] S. L. Smith, M. Schwager, and D. Rus. Persistent robotic tasks: Monitoring and sweeping in changing environments. *IEEE Transactions on Robotics*, 28(2):410–426, 2011.

[28] R.S. Sutton and A.G. Barto. *Reinforcement Learning – An Introduction.* MIT Press, 2018.

[29] R.S. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999.

[30] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 23–30. IEEE, 2017.

[31] Y. Wang, Z. R. Shi, L. Yu, Y. Wu, R. Singh, L. Joppa, and F. Fang. Deep reinforcement learning for green security games with real-time information. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 1401–1408, 2019.