Proceedings of the Thirty-Sixth International Conference on Automated Planning and Scheduling (ICAPS 2026)

Memory Assignment for Finite-Memory Strategies in Adversarial Patrolling Games

Vojtěch Kůr, Vít Musil, Vojtěch Řehák

Masaryk University, Brno, Czechia
vojtech.kur@mail.muni.cz, {musil, rehak}@fi.muni.cz

Abstract

Adversarial Patrolling games form a subclass of Security games where a *Defender* moves between locations, guarding vulnerable targets. The main algorithmic problem is constructing a strategy for the Defender that *minimizes* the worst damage an *Attacker* can cause. We focus on the class of *finite-memory* (also known as regular) Defender's strategies that experimentally outperform other competing classes. A finite-memory strategy can be seen as a positional strategy on a finite set of states. Each state consists of a pair of a location and a certain integer value–called memory. Existing algorithms improve the transitional probabilities between the states but require that the available memory size itself is assigned at each location *manually*. Choosing the right memory assignment is a well-known open and hard problem that hinders the usability of finite-memory strategies. We solve this issue by developing a general method that iteratively changes the memory assignment. Our algorithm can be used in connection with *any* black-box strategy optimization tool. We evaluate our method on various experiments and show its robustness by solving instances of various patrolling models.

Code — https://gitlab.fi.muni.cz/formela/regstar
Extended version — https://arxiv.org/abs/2505.14137

Introduction

This work follows the *patrolling games* line of work studying non–cooperative games where a mobile Defender guards resources against an Attacker. In *adversarial* patrolling (Vorobeychik, An, and Tambe 2012; Agmon, Kraus, and Kaminka 2008; Agmon et al. 2009; Basilico, Gatti, and Amigoni 2012, 2009; de Cote et al. 2013; Sless, Agmon, and Kraus 2019), the Attacker *knows* the Defender's strategy and observes his locations. The solution concept is based on *Stackelberg equilibrium*. The Defender commits to a strategy σ, and the Attacker chooses a counter-strategy π that maximizes the expected Attacker's utility against σ. Intuitively, the value $\mathrm{Val}(\sigma)$ denotes the damage caused by the best Attacker, and the precise definition of Val depends on the underlying *patrolling model*.

The environment where the agents play can be described by a Markov decision process. If the environment is not fully known, the problem becomes one of reinforcement learning, where the agent learns the strategy through interaction with the environment, using either model-free (e.g., Q-learning) or model-based (e.g., using a learned model to simulate outcomes) approaches. We focus on the cases when the *environment is fully known* both to the Defender and the Attacker. An example of such an environment is an oriented graph that typically describes the topology of the terrain. This concept is also used in the domain of planning.

Another aspect is the time frame in which the game is evaluated, which plays a critical role in shaping the agent's behavior. We focus on an *infinite-horizon model*, where the Attacker can wait as long as needed to exploit the best opportunity that maximizes the targets' damage. This is especially suitable for uninterrupted services where the Defender represents many consecutive patrol shifts or a robot that is expected to run for a long time.

For general topologies, deciding whether there is a Defender's strategy with null expected damage is PSPACE-complete(Ho and Ouaknine 2015). Additionally, deciding whether there is a strategy with at most ε expected damage is NP-hard (Klaška, Kučera, and Řehák 2020) (where $\varepsilon \leq 1/2n$ and n is the number of vertices). Therefore, no polynomial algorithm can guarantee (sub)optimality in general (unless $\mathrm{P} = \mathrm{NP}$). Hence, finding suitable heuristics applicable to real-world scenarios is of great importance.

Prior works focus on constructing *positional strategies* where the Defender makes a randomized choice of the next location based only on the current location (Basilico, Gatti, and Amigoni 2012; Alpern et al. 2022; Basilico, Nittis, and Gatti 2017; Collins et al. 2013). However, positional strategies fail to provide optimal protection even in a simple setting; see Figure 3 or (Kučera and Lamser 2016).

To increase the strategy expressivity, Kučera and Lamser (2016) introduced a class of *regular strategies*, where the Defender is equipped with a suitable finite-state automaton. In each location, the distribution over the next locations is determined by the state of the automaton after feeding it with the current history of locations. The authors proved that regular strategies offer better protection than window-based strategies, which generalize positional strategies by considering the last k locations. They also present a strategy synthesis algorithm for regular strategies. However, this approach's main limitation is that the finite-state automaton

must be supplied manually to achieve strong strategies.

The idea of regular strategies was followed by *finite-memory* strategies where the finite-state automaton is abstracted into finite integer memory denoted M (Klaška et al. 2018). The Defender chooses the new location based on the current location and memory value and also selects (possibly randomly) the new memory value.

Finite-memory strategies can also be seen as positional strategies on the set of states consisting of the pairs of the location and a memory integer between 1 and M. In this view, the state space increases dramatically, limiting the usability and performance of optimization algorithms. In fact, the size of the memory does not necessarily need to be the same in each location, i.e., the states can be set as $C = \{(v, i) \mid v \in V, 1 \leq i \leq \mathrm{mem}(v)\}$, where V is a set of locations and $\mathrm{mem}\colon V \to \mathbb{N}$ is certain *memory-assignment* function.

Recent works provide well-performing optimization algorithms for finite-memory strategies and various patrolling models (Klaška et al. 2018, 2021; Brázdil et al. 2022; Klaška et al. 2022). However, they all assume that a suitable memory assignment is given as an input. Currently, this is the main bottleneck of these algorithms, and the results depend heavily on the choice of mem.

All the published experiments either use uniform memory assignment with increasing M (that hits the limits of the algorithms already for $M \lesssim 10$) or use carefully handcrafted assignment mem based on the expert knowledge of the problem at hand. To the best of our knowledge, *no approach automates memory assignment* for general topologies. Klaška et al. (2021) state that choosing the optimal set of states is an open and hard problem. We demonstrate the problem of optimal memory assignment in Example 1 below.

Example 1 Consider a graph where three target vertices v_1, v_2, v_3 are connected to a central vertex X, forming a star topology.

Assume that moving along any edge takes 1 time unit, and each target requires 6 time units to complete an attack. The Defender uses a patrolling strategy that cycles through the targets in the sequence:

$$v_1 \to X \to v_2 \to X \to v_3 \to X \to v_1.$$

Since the length of this cycle is 6, the patrol takes 6 time units to complete. This way, the Defender reaches each target within the attack duration and can thus intercept any ongoing attack in time.

However, this strategy is not positional since the Defender's decision at X depends on the history. We can model this behavior as a finite-memory strategy, where the Defender must maintain three distinct internal states at X, denoted X_1, X_2, X_3, resulting in the sequence:

$$v_1 \to X_1 \to v_2 \to X_2 \to v_3 \to X_3 \to v_1.$$

Now suppose that the attack duration of v_1 is 4 and the attack durations of v_2 and v_3 are 8. The original strategy is no longer good, since an observing and patient Attacker can wait for the moment when the Attacker moves from v_1 to X and initiate an attack on v_1. Since the Defender will return to v_1 after 6 time units, the attack will be completed.

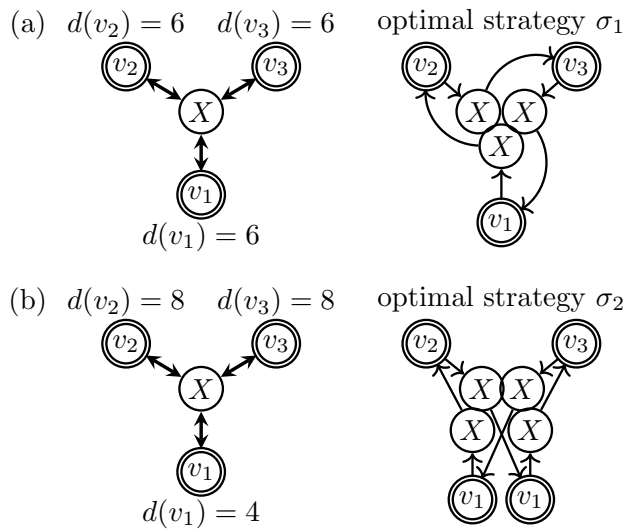

Figure 1: The optimal deterministic strategies require different memory allocations. The goal is to patrol a graph (a) with three targets and an internal location M. It takes $d = 6$ time units to complete an attack on each target. The optimal strategy is to cycle around the graph since the cycle's length is 6. This is represented as σ_1, which requires 3 different states at M. In (b), the attack durations change. The strategy σ_2 achieves perfect protection since v_1 is always visited every 4 steps, and both v_2 and v_3 are visited once on the cycle of length 8. Strategy σ_2 needs 4 states in M and 2 states in v_1.

In this case, the optimal strategy returns to v_1 after each visit of v_2 and v_3, resulting in a sequence:

$$v_1 \to X \to v_2 \to X \to v_1 \to X \to v_3 \to X \to v_1.$$

Here, the Defender needs two different states for v_1 and 4 for X. Both settings and their respective optimal strategies are visualized in Figure 1.

Discussion. This example demonstrates the difficulty of choosing the optimal memory assignment. It shows that the minimal optimal set of states does not solely depend on the topology but differs based on particular properties of the targets. Moreover, memory may serve purposes beyond merely remembering the last visited vertex; it can capture more complex patterns in the Defender's behavior.

Our Contribution

We propose a smart general heuristic algorithm that dynamically adjusts the memory assignment for finite-memory strategies without any expert knowledge. Our method works with any differentiable value function and any black-box strategy optimization tool. In combination, our memory assignment algorithm, together with any strategy synthesis algorithm, yields a general and fully automated method for a strong finite-memory Defender's strategy.

We evaluate our method on several benchmarks and for various patrolling models. Our automated approach outperforms the manual memory assignment when mem is chosen

to be uniform with increasing M in nearly all cases. Furthermore, our method is on par with expertly handcrafted memory assignments.

Related Work

This work contributes to the field of security games, which focuses on the optimal allocation of limited security resources to achieve effective target coverage, see monograph (Tambe 2012). Patrolling games are a specialized type of security games where the Defender is mobile; see survey works (Huang et al. 2019; Almeida et al. 2004; Portugal and Rocha 2011; Basilico 2022). Most existing patrolling models can be categorized as either *regular* or *adversarial*.

Regular patrolling (Afshani et al. 2021; Hari et al. 2021; Farinelli, Iocchi, and Nardi 2017) is akin to surveillance, where the Defender's goal is to quickly discover incidents by minimizing the time between consecutive visits to each target. In this model, the Defender typically employs a strategy involving a single path or cycle that visits all targets.

In contrast, *adversarial patrolling* (Vorobeychik, An, and Tambe 2012; Agmon, Kraus, and Kaminka 2008; Agmon et al. 2009; Basilico, Gatti, and Amigoni 2012, 2009; de Cote et al. 2013; Sless, Agmon, and Kraus 2019; Basilico 2022) focuses on protecting targets from an Attacker who seeks to exploit the best opportunities to maximize damage. This model is generally framed within the context of Stackelberg equilibrium (Yin et al. 2010; Sinha et al. 2018). The Defender's strategies are often *randomized* to prevent the Attacker from predicting future moves, and the Defender aims to maximize the probability of detecting an attack. The adversarial approach is particularly relevant in scenarios where a certain level of protection must be maintained, even if incidents occur at the most inconvenient times.

For patrolling scenarios of bounded duration, such as an eight-hour shift of a human ranger, *finite-horizon* models are sufficient. In contrast, *infinite-horizon* models are used when the duration is potentially unbounded or the bound is large, like 24/7 surveillance.

The model can also be distinguished by prior knowledge about the environment, which can be either fully known to all players or needs to be discovered during patrolling.

The difference between models has a substantial impact on strategy synthesis techniques. For finite horizon models with known environments, mathematical programming is the primary technique (Basilico, Gatti, and Amigoni 2012, 2009). Reinforcement learning has been largely successful for patrolling scenarios with a finite horizon and unknown environment, such as green security games (Wang et al. 2019; Biswas et al. 2021; Xu 2021; Karwowski et al. 2019). Gradient descent methods for finite-memory strategies are used extensively for infinite horizon models with known environments (Klaška et al. 2018, 2021; Brázdil et al. 2022; Klaška et al. 2022).

Practical applications of security games include the deployment of police checkpoints at Los Angeles International Airport (Pita et al. 2008), the scheduling of federal air marshals on domestic airline flights across the U.S. (Tsai et al. 2011), and the strategic arrangement of city guards in the Los Angeles Metro (Fave et al. 2014), positioning of U.S. Coast Guard patrols to secure critical locations (An et al. 2012), as well as the initiatives for wildlife protection in Uganda (Ford et al. 2014).

Background

First, we recall the standard notions of a patrolling graph, the Defender's and Attacker's strategy, and their value, and we fix the notation. Then, we formulate the requirements of our algorithm for the underlying patrolling model and show how existing patrolling models fit our framework. We assume familiarity with basic notions of calculus, linear algebra, probability theory, and finite-state discrete-time Markov chains.

Patrolling Graph

A *patrolling graph* is a tuple $G = (V, T, E, time)$, where V is a non-empty set of *locations* (admissible Defender's positions), $T \subseteq V$ is a non-empty set of *targets*, $E \subseteq V \times V$ is a set of *edges* (admissible Defender's moves) and $time\colon E \to \mathbb{N}$ specifies the traversal time of an edge.

Defender's Strategy

In general, the Defender can choose the next location randomly based on the whole history of previously visited locations. However, general strategies may not be finitely representable. We focus on finite-memory Defender's strategies, which were shown to achieve the same limit protection as general strategies in two different patrolling models; see (Klaška et al. 2021) and (Klaška et al. 2022).

In finite-memory strategies, the Defender is equipped with an integer variable M called *memory*. A *state* of the Defender is a pair $(v, i) \in V \times \mathbb{N}$. A finite-memory strategy of the Defender can be seen as positional on a fixed finite set of states, i.e., it forms a discrete-time Markov chain on $V \times \mathbb{N}$.

To prevent the state space from blowing up, different amounts of memory can be allocated to each location. Hence, we define the *memory assignment* as a function $\mathrm{mem}\colon V \to \mathbb{N}$ and a set of Defender's states

$$C = \{(v, i) \mid v \in V, 1 \leq i \leq \mathrm{mem}(v)\}. \tag{1}$$

A finite-memory strategy of the Defender on V is a positional strategy on C, i.e., a function $\sigma\colon C \times C \to [0, 1]$ satisfying $\sum_{y \in C} \sigma(x, y) = 1$, for each $x \in C$, and $(u, v) \in E$, whenever $\sigma((u, i), (v, j)) > 0$. Intuitively, if the Defender is currently in a location u with a memory value i, the next state (v, j) is chosen with probability $\sigma((u, i), (v, j))$, which means that the Defender changes the memory value to j and starts traveling from u to v. The second condition ensures that the strategy respects the edges of the graph.

For every finite sequence of states $h = (c_1, \ldots, c_n)$, we use $\mathbb{P}^{\sigma,c}[h]$ to denote the probability of executing h when the Defender starts patrolling in the state $c \in C$ and follows the strategy σ. That is, $\mathbb{P}^{\sigma,c}[h] = 0$ if $c_1 \neq c$, and $\mathbb{P}^{\sigma,c}[h] = \prod_{i=1}^{n-1} \sigma(c_i, c_{i+1})$ otherwise.

Strategy Types If $\mathrm{mem} \equiv 1$, then we say that σ is positional on V or *memoryless*. If for every state x there is a state y such that $\sigma(x, y) = 1$, then we say that σ is *deterministic*. Note that σ can be both memoryless and deterministic.

Attacker's Strategy

In the patrolling graph, the time is spent traversing the edges. In adversarial models, the Attacker is assumed to perfectly observe the Defender's moves and can determine the next edge taken by the Defender immediately after its departure. For the Attacker, this is the best moment to attack because delaying the attack could only lower the attack's gain. Furthermore, the Attacker is allowed to attack only once in a single run.

An *observation* is a sequence of states $o = (c_1, \cdots, c_n, c_{n+1})$. Intuitively, c_1 is the initial state of the Defender, c_n is the current state of the Defender, and c_{n+1} is the state chosen as the next one according to the Defender's strategy. The set of all observations is denoted by Ω. Formally, an *Attacker's* strategy is a function $\pi \colon \Omega \to T \cup \{wait\}$. Since the Attacker is allowed to attack only once, it is required that if $\pi(o) \in T$, then $\pi(o') = wait$ for every prefix o' of o.

Protection Value and Target Types

Let σ be a finite-memory Defender's strategy and π an Attacker's strategy. Let us fix an initial state c where the Defender starts patrolling. The expected damage the Attacker causes by attacking τ after observing o is denoted as $\mathcal{D}(o, \tau \mid \sigma)$. We will now present some examples of the definition of the value $\mathcal{D}(o, \tau \mid \sigma)$.

Hard-constrained targets One way of defining the damage on targets is to model a scenario where the Attacker needs to perform some action (e.g. picking a lock) to successfully complete an attack. If the target is visited in time, the Defender 'catches' the Attacker, and the damage is zero. Otherwise, the Attacker 'steals' the cost of the target. Here, $d(\tau)$ denotes the time it takes to complete an attack on τ and $\alpha(\tau)$ its cost. The value $\mathcal{D}(o, \tau \mid \sigma)$ is then defined as the probability that the Defender does not visit τ in the next $d(\tau)$ time units multiplied by $\alpha(\tau)$. We call this type of target *hard-constrained*. Within the context of finite-memory strategies, this model was first introduced by (Klaška et al. 2018) and later used in (Klaška et al. 2021; Brázdil et al. 2022).

Blind targets The hard-constrained model can be extended by introducing a level of uncertainty where the ongoing attack is discovered only with probability $\beta(\tau)$ upon each visit of the target. We call these targets *blinded* and appear in (Klaška et al. 2021; Brázdil et al. 2022).

Linear targets Another target type models a situation where the actual damage depends on the time elapsed since initiating the attack till the Defender's visit, e.g., a fire or punching a hole in a fuel tank. Here, the target τ is assigned a value $v(\tau)$, which denotes the damage caused for every time unit while the target is under attack. Hence, the expected damage $\mathcal{D}(o, \tau \mid \sigma)$ equals the expected time it takes to visit τ multiplied by $v(\tau)$. If the probability of visiting τ after o is zero, then $\mathcal{D}(o, \tau \mid \sigma) = \infty$. We call this type of targets *linear*. It was introduced in (Klaška et al. 2022).

Game Value

The *expected damage* caused by the Attacker is defined as

$$\mathrm{Val}(\sigma, c, \pi) = \sum_{o \in \Omega, \pi(o) \in T} \mathbb{P}^{\sigma,c}[o] \cdot \mathcal{D}(o, \pi(o) \mid \sigma). \tag{2}$$

The Defender/Attacker aims to minimize/maximize the damage, and hence, we define

$$\mathrm{Val}(\sigma) = \min_{c \in C} \mathrm{Val}(\sigma, c) = \min_{c \in C} \sup_{\pi} \mathrm{Val}(\sigma, c, \pi)\ . \tag{3}$$

By $\mathrm{Val}(G)$, we denote the limit value of the best Defender's strategy or formally $\mathrm{Val}(G) = \inf_\sigma \mathrm{Val}(\sigma)$.

Requirements on the Patrolling Model

For our algorithm, we require two properties of the underlying patrolling model.

Let σ be a Defender's strategy and let $BSCC(\sigma)$ denote all subsets of states corresponding to bottom strongly connected components of the Markov chain determined by σ. For every $B \in BSCC(\sigma)$, let us denote by $\mathrm{Val}(\sigma \mid B)$ the value

$$\max_{\tau \in T, e \in B \times B, \sigma(e) > 0} \mathcal{D}(e, \tau \mid \sigma).$$

Our first requirement is that $\mathrm{Val}(\sigma)$ can be expressed as

$$\mathrm{Val}(\sigma) = \min_{B \in BSCC(\sigma)} \mathrm{Val}(\sigma \mid B). \tag{4}$$

Note that if σ is irreducible, then $\mathrm{Val}(\sigma) = \mathrm{Val}(\sigma \mid C)$.

Secondly, given a state space C, we require that for every $e \in C \times C$ and $\tau \in T$ the value $\mathcal{D}(e, \tau \mid \sigma)$ can be effectively calculated, and that $\mathcal{D}(e, \tau \mid \sigma)$ is a differentiable function with respect to σ whose gradient can also be evaluated.

The proofs that hard-constrained, blinded, and linear targets are eligible for our general framework are provided in the works where they were presented. In general, it can be seen that a sufficient condition for (4) is that the damage to the targets depends only on the current state and the future. That is, for every observation $o = (c_1, \ldots, c_n, c_{n+1})$ and target τ, it holds

$$\mathcal{D}(o, \tau \mid \sigma) = \mathcal{D}\big((c_n, c_{n+1}), \tau \mid \sigma\big)\ . \tag{5}$$

See the Extended Version for a proof.

The Method

Here, we describe our method for solving the problem of choosing a suitable memory assignment. Our method works with any black-box optimization tool that improves the transitional probabilities and with any patrolling model that fits our general framework. Recall that the algorithms introduced in (Klaška et al. 2021; Brázdil et al. 2022; Klaška et al. 2022; Klaška et al. 2018) are eligible instances of such black-boxes.

The algorithm is initialized with the memory assignment mem_1 that assigns 1 to each location. That is, we start with a

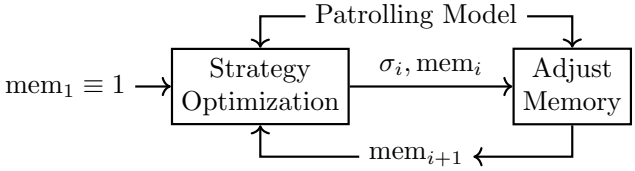

Figure 2: Diagram of our method. We iteratively run a strategy optimization black-box followed by our ADJUSTMEMORY procedure that updates the memory assignment.

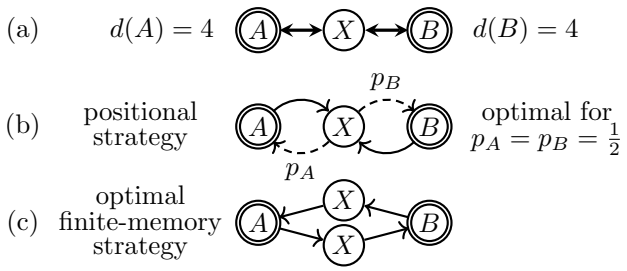

Figure 3: Patrolling on a graph (a) with three locations. End locations represent targets with costs of 1 and attack lengths equal to 4. Positional strategies (b) are parametrized by a single variable p. Due to symmetry, the optimality is reached for $p = 0.5$. The optimal strategy (b) is deterministic and needs two states in X.

memoryless strategy. Then, we run the optimization tool and collect the resulting strategy σ_1. From this, we run the core procedure ADJUSTMEMORY that outputs a new assignment mem_2. The memory assignment mem_2 is then used for initialization for the tool, and we collect a new strategy σ_3. This process continues for as long as the strategy value improves, see Figure 2.

Attacks Profiles

Now, we describe the core procedure ADJUSTMEMORY of our algorithm that decides how much memory should be added to which location. Intuitively, the procedure estimates the number of different 'behaviors' of each state that, on their own, could improve the current value.

Let us illustrate the idea with a simple example. Consider a patrolling graph depicted in Figure 3 with two targets A and B that are hard-constrained with $d = 4$ and $\alpha = 1$. We start with $\mathrm{mem} = 1$. The state space is $C = \{(A,1),(X,1),(B,1)\}$. For brevity, let us write v for $(v,1)$ since the memory value is constant. In this example, from the end locations A and B, the Defender always returns to X, and hence the strategy depends only on complementary probabilities $\sigma(X,A)$ and $\sigma(X,B)$, which we denote by p_A and p_B, for short.

The best moment to attack A is when the Defender starts traversing from X to B. Then the probability of not visiting A in $d(A) = 4$ time units is exactly p_B and hence $\mathcal{D}((X,B), A \mid \sigma) = p_B$. Analogically, the best moment to attack B is when the Defender starts traversing from X to A and $\mathcal{D}((X,A), B \mid \sigma) = p_A$. The value $\mathrm{Val}(\sigma)$ is then the maximum of both the attack values, i.e. $\mathrm{Val}(\sigma) = \max\{p_A, p_B\}$ which is minimized when $\sigma(X,A) = \sigma(X,B) = 0.5$ with $\mathrm{Val}(\sigma) = 0.5$.

We have reached the optimal value of positional strategies, and the question is whether the Defender can do better. The answer is positive since a finite deterministic strategy that moves from one end to the other and back represents a cycle of length 4. Hence, every attack is discovered in time, achieving the optimal value 0. This strategy can be represented by a finite-memory strategy on memory assignment $\mathrm{mem}_{\mathrm{det}}$ with $\mathrm{mem}_{\mathrm{det}}(X) = 2$ and $\mathrm{mem}_{\mathrm{det}}(A) = \mathrm{mem}_{\mathrm{det}}(B) = 1$.

In practice, a reasonable optimization tool should output a strategy close to σ. Now, the key question is how to infer $\mathrm{mem}_{\mathrm{det}}$ from σ. The idea is to consider gradients of all the maximal attacks w.r.t. the parameters of strategy σ. Any strategy σ can be modeled as Softmax of unconstrained parameters. In our example, we need just two real parameters, say (x_A, x_B), so that

$$\mathrm{Softmax}(x_A, x_B) = (p_A, p_B).$$

Using the chain rule, the partial derivatives of the maximal attacks w.r.t. the parameters x_A, x_B are

$$\frac{\partial \mathcal{D}\big((X,B),A|\sigma\big)}{\partial x_A} = -p_A p_B \quad \text{and} \quad \frac{\partial \mathcal{D}\big((X,B),A|\sigma\big)}{\partial x_B} = p_A p_B$$

and

$$\frac{\partial \mathcal{D}\big((X,A),B|\sigma\big)}{\partial x_A} = p_A p_B \quad \text{and} \quad \frac{\partial \mathcal{D}\big((X,A),B|\sigma\big)}{\partial x_B} = -p_A p_B.$$

We can see that the problem lies in the fact that for the attack on A, the gradient of the parameters is (up to a scale) $(-1,1)$, while for the attack on B it is $(1,-1)$, which is in the exact opposite direction. Hence, the value can no longer improve since the competing gradients 'cancel each other out'. This is exactly the place where we need to add a more memory value for X, one memory value to prioritize B (if the previous location was A) to cover the attack on B, and the second memory value to go to A (if the previous location was B) in order to cover the attack on A.

Formally, we fix a set of *eligible attacks* $\mathcal{E} \subseteq (C \times C) \times T$, corresponding to the attacks with close to maximal value. For an attack $(e,\tau) \in \mathcal{E}$ and a state $c \in C$, we consider the gradient of the attack value with respect to the outgoing probabilities of c

$$\nabla_\sigma \mathcal{D}(e,\tau \mid \sigma) = \left(\frac{\partial \mathcal{D}(e,\tau \mid \sigma)}{\partial \sigma(c,c_1)}, \ldots, \frac{\partial \mathcal{D}(e,\tau \mid \sigma)}{\partial \sigma(c,c_n)} \right), \quad (6)$$

where $c_1, \ldots, c_n$ are all the states with $\sigma(c,c_i) > 0$. Assuming that the strategy probabilities are parametrized by the Softmax of $x(c) = (x_1, \ldots, x_n)$, the gradient of the attack value with respect to the strategy parameters is given by

$$\nabla_{x(c)} \mathcal{D}(e,\tau \mid \sigma) = \left(\tfrac{\partial \mathcal{D}}{\partial x_1}, \ldots, \tfrac{\partial \mathcal{D}}{\partial x_n}\right) = J^T \cdot \nabla_\sigma \mathcal{D}(e,\tau \mid \sigma),$$

where J is the Jacobian of the Softmax. The *attack profile* of (e,τ) and c is then the *sign* of this vector $\nabla_{x(c)} \mathcal{D}(e,\tau \mid \sigma)$. We define $\mathrm{profiles}$ to be a function that assigns each state c

the set of all (different) attack profiles belonging to c from $\mathcal{E}$. Formally,

$$\text{profiles}(c) = \{\text{sign}(\nabla_{x(c)}\mathcal{D}(e,\tau \mid \sigma)) \mid (e,\tau) \in \mathcal{E}\}.$$

The new memory assignment mem_2 is then defined as

$$\text{mem}_2(v) = \sum_{1 \le i \le \text{mem}_1(v)} |\text{profiles}(v,i)|. \tag{7}$$

The whole process is also described in the Algorithm 1.

Theoretically, a locally optimal strategy implies the existence of multiple maximal attacks with identical values (conflicting gradients). However, since numerical optimization tools may converge to solutions where these values differ, the threshold ε ensures we capture all relevant attacks that should be driving the gradient descent. We have experimentally found that $\varepsilon = 0.25$ reliably incorporates all such maximal attacks across different patrolling graphs while significantly reducing computation time.

Bounded Number of States

We also introduce a modification of our method for the cases when the total number of states is bounded above by L. Bounding the maximal number of states of the strategy may be both desired by the user (memory of a robot) and the whole optimization process (algorithms run very slowly for strategies with a high number of states).

Assume we have an upper bound $L \ge |V|$ on the number of states for each σ_i. Let the current memory assignment be mem_1 and σ. We run ADJUSTMEMORY and collect the resulting mem_2 and attack profiles. If $\sum_v \text{mem}_2(v) \le L$, we simply output mem_2. Otherwise, for each attack profile, we sum the total value of the attacks with the given profile. That is, for each state c and an attack profile $\gamma \in \text{profile}(c)$, we define the *value* of γ as

$$\text{val}(\gamma, c) = \sum_{\substack{(e,\tau)\in\mathcal{E} \\ \text{sign}(\nabla_{x(c)}\mathcal{D}(e,\tau\mid\sigma))=\gamma}} \mathcal{D}(e,\tau \mid \sigma).$$

Then, the idea is to take L profiles with the highest value, ensuring every state has at least one profile. The memory is then calculated in the same way as in the unbounded version.

Formally, let $\text{profiles}'(c)$ be the set of profiles of c, excluding one with maximal value (it does not matter which one). This is valid since every state must have a profile with an attack of maximal value. Then we flatten all the attack profiles into a single set $\mathcal{P} = \bigcup_c \{(\gamma, c) \mid \gamma \in \text{profiles}'(c)\}$. After that, we take the $L - |C|$ highest items from $\mathcal{P}$ where the order is given first by val and secondly by the order on the states (if more profiles of the same state have the same value, the order does not matter). Let us denote by $\#(c)$ the number of occurrences of a state c among the $L - |C|$ maximal items of $\mathcal{P}$. Finally, the new memory assignment mem_2 is

$$\text{mem}_2(v) = \sum_{1 \le i \le \text{mem}_1(v)} 1 + \#(v,i).$$

Complexity Analysis

Here, we describe the complexity analysis of our method, its bounded variation and strategy expansion.

Algorithm 1: Core procedure of our method.

Function `AdjustMemory` (σ, mem, ε) **:**
- Define $x(c)(c_i)$ to be $\log(\sigma(c, c_i))$
- **foreach** c ∈ C **do**
  - profiles(c) ← ∅
- **foreach** $e \in C \times C, \tau \in T$ **do**
  - **if** $\mathcal{D}(e,\tau \mid \sigma) < (1-\varepsilon)\,\text{Val}(\sigma)$ **then**
    - **continue**
  - **foreach** c ∈ C **do**
    - profile ← $\text{sign}(\nabla_{x(c)}\mathcal{D}(e,\tau \mid \sigma))$
    - Add profile to $\text{profiles}(c)$
- **foreach** $v \in V$ **do**
  - $\text{mem}'(v) \leftarrow 0$
  - **for** $i \leftarrow 1$ **to** $\text{mem}(v)$ **do**
    - $\text{mem}'(v) \leftarrow \text{mem}'(v) + |\text{profiles}(v,i)|$
- **return** mem'

Algorithm 1 The time complexity of Algorithm 1 is in

$$\mathcal{O}(f + |\mathcal{E}|g + |\mathcal{E}|^2|C|^2)$$

where f is such that the values of all attacks are calculated in $\mathcal{O}(f)$ and g is such that for each attack, the gradients with respect to the parameters are evaluated in $\mathcal{O}(g)$. Next, $|E|$ denotes the number of state edges such that its traversal probability is positive, and finally, $|\mathcal{E}|$ denotes the number of attacks taken into account.

Bounded case Let $M = \sum_v \text{mem}_2(v)$. If $M \le L$, then the algorithm is the same as the normal case. Otherwise, we claim that it runs in

$$\mathcal{O}\left(|C||\mathcal{E}| + (M - |C|)\log(L - |C|)\right).$$

Bounded case Let $M = \sum_v \text{mem}_2(v)$. If $M \le L$, then the algorithm is the same as the normal case. Otherwise, we claim that in runs in

$$\mathcal{O}\left(|C||\mathcal{E}| + (M - |C|)\log(L - |C|)\right).$$

See the Extended Version for proofs. *In practice*, however, the evaluation for hard-constrained and blinded was faster than for linear. This is primarily because the first algorithm's core procedure is implemented in C++ with a dynamic program and uses a sparse representation of the strategy, unlike the second, which is in PyTorch and uses full matrices. In both cases, the actual bottleneck was in calculating the gradients.

Warm-starting

Some optimization tools enable us to warm-start the process instead of always starting from random parameters. In the Extended Version, we present a way to transform a strategy σ_i on mem_i to σ_{i+1} on mem_{i+1}. However, we conclude that this does not provide a significant advantage, and we omit this extension from the experiments presented here.

Experiments

We evaluate our memory assignment method in a series of experiments with the best-performing algorithm for hard-constrained and blind targets of (Brázdil et al. 2022) and the

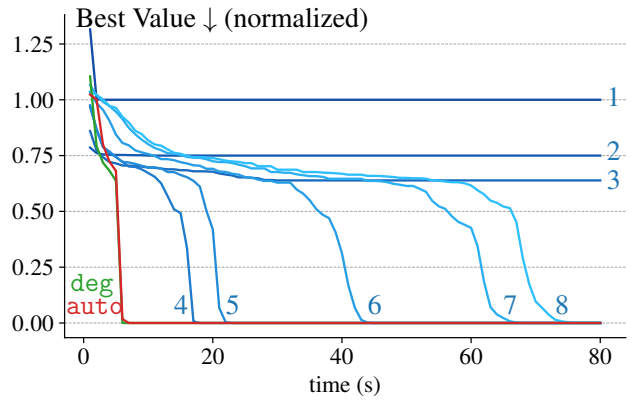

Figure 4: Values over time of the best performing strategies (out of 200 restarts) on one-floor **Offices**. The values are normalized by the value of the best memoryless strategy, optimum is zero. Uniform memory below 4 cannot achieve optimum; higher memory gets zero but significantly later as the memory increases. Both Expert (`deg`) and Automatic (`auto`) outperform the uniform assignment.

algorithm for linear targets of (Klaška et al. 2022). We created a unified Python project containing all compatible state-of-the-art algorithms with existing relevant benchmarks. The repository will be publicly released (preliminarily, see the Extended Version).

We also test the manual assignment `deg` that sets $\mathrm{mem}(v)$ to the out-degree of v. This is precisely the memory needed when an Eulerian cycle forms the optimal strategy. This memory was expertly handcrafted by the authors (Klaška et al. 2022) for their experiments.

We compare uniform memory assignments (denoted by a single number) with the degree assignment (`deg`), and our automatic method (`auto`). We ran experiments on three patrolling models/target types (hard-constrained, blind, linear) on over 50 different patrolling graphs (Offices, Building, Airports, Terrains, Stars).

For each setting, we executed 200 runs with a timeout of 180 seconds to get reliable statistical information. The number of states is bounded by 300. Recall that the whole procedure is stochastic as the optimization starts from a random init and may also contain non-deterministic updates.

Here, we present only a fragment of our experiments to highlight our key observations and limitations. The broader summary is in Section . The full report, detailed setup, and reproducibility steps can be found in the Extended Version.

Offices This benchmark comes from (Klaška et al. 2021). The goal is to patrol office buildings with 1, 2, or 3 floors. The offices are designed so that an Eulerian cycle is the optimal strategy. The maximum degree is 4; hence, the uniform memory must be at least 4 to achieve optimum. Figure 4 shows the optimization dynamics on 1 floor and Figure 5 reports the values of strategies for offices on 2 floors.

Building (Klaška et al. 2021) also used the same topology as Offices to evaluate the synthesis of blinded targets. We recreate this benchmark, and the results can be found in the Extended Version.

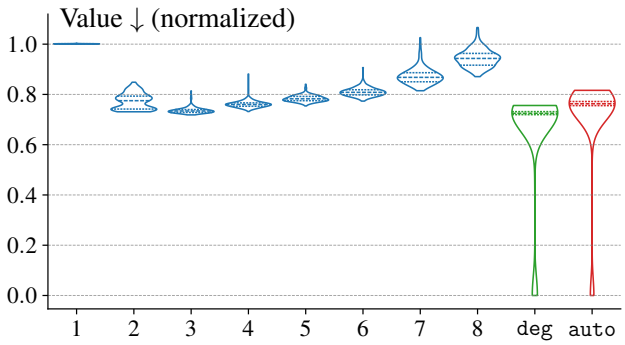

Figure 5: Comparison of memory assignments on the **Offices** benchmark (2 floors). Statistics are over 200 restarts. Uniform $\mathrm{mem} = 4$ is sufficient for the optimal strategy, but never found for any of the uniform within the timeout. Both Expert (`deg`) and Automatic (`auto`) memory assignments lead to the optimum.

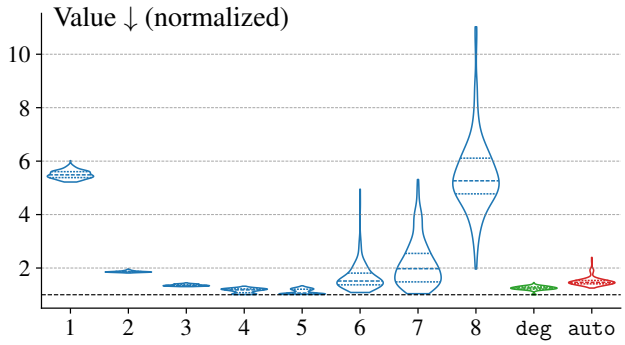

Figure 6: Comparison of memory assignments on the **Airport** with 19 halls. The values are normalized by the value of the baseline (Eulerian cycle requiring degree memory). Uniform memory (for $\mathrm{mem} = 4$) is sufficient to meet the Expert (`deg`) assignment. This is one of the rare examples where the Automatic (`auto`) assignment lags behind.

Airports This benchmark originated from (Klaška et al. 2022). The goal is to patrol airport gates. An airport with n halls has $3n + 1$ nodes, from which $2n$ are linear targets. In Figure 6, we report the distribution of strategy values for $n = 19$. We also ran the benchmark with random target values sampled uniformly and independently between 1 and 10.

Stars We introduce a new benchmark on star-shaped graphs, where the targets and the attack lengths are chosen so that the optimal strategy requires a nontrivial amount of memory. The graph is parametrized by a number of groups k ranging from 1 to 5. A graph with k groups has $k + 2$ vertices. In Figure 7, we report the statistics for $k = 3$.

Terrains Another new benchmark consists of randomly generated connected planar graphs that model open terrains.

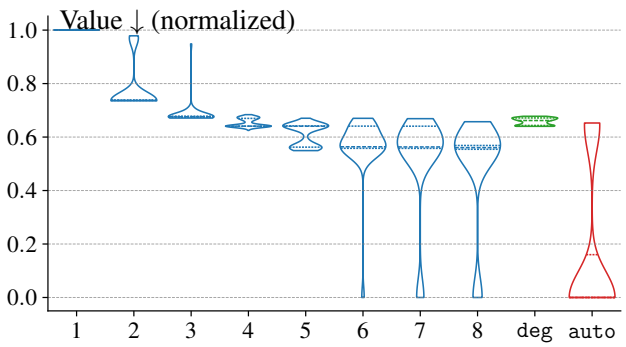

Figure 7: Comparison of memory assignments for the **Star** experiment with 3 groups. The values are normalized by the value of the best memoryless strategy found, optimum is zero. The optimal strategy requires more than a degree amount of memory. Automatic assignment achieves optimum consistently.

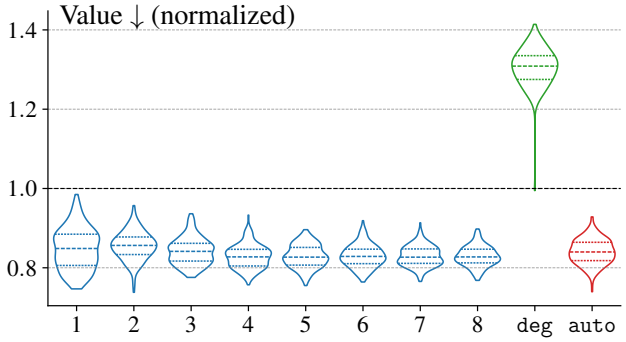

Figure 8: Comparison of memory assignments for the **Terrains** experiment with size $n = 33$. The values are normalized by a baseline (the best deterministic strategy). Larger memory does not help here. Degree assignment causes the state space to be too large, and well-performing strategies are not found.

The nodes are randomly generated on a plane, triangulated, connected by the minimum spanning tree, and each edge from triangulation is added with a probability of 0.5. We run the experiments with the odd number of nodes n ranging from 3 to 41. Figure 8 shows the statistics for $n = 33$.

Discussion

Failure of uniform We observe that uniform memory assignments are rarely optimal. Oftentimes, more memory is needed, and low values of M achieve poor results. In theory, if there is an optimal finite-memory strategy, then a high enough uniform assignment suffices. The authors of (Klaška et al. 2021) state that the probability of finding the optimal strategy increases with M in the Offices benchmark. We verify this, but higher M also significantly increases the state space and the time needed to find the optimal strategy (see Figure 4).

Therefore, within a fixed timeout, these two competing aspects (sizes of M and state space) create a convex-shaped dependency on M, where the lowest values may or may not hit the optimum (cf. Figures 6 and 5). The behavior is hard to predict without intensive testing, making the uniform assignment impractical.

We can conclude that the distribution of memory states matters. Degree assignment proved to be a simple heuristic that often yields superior results. Automatic assignment achieves better or comparable results than uniform in almost all cases.

Limitations of degree For Offices and Airport benchmarks, degree memory assignment produced the best results. However, it scales poorly when it adds too much memory (Terrains, Building), see Figure 8. On the opposite, it may provide an insufficient amount of memory, see Figure 7. While it is a good heuristic for smaller graphs with a small maximal degree, it may achieve poor results on big or highly connected graphs or in cases when the optimal strategy requires more information than remembering the next vertex.

Robustness of auto Our proposed automatic assignment achieves high robustness across all the experiments. This heuristic enables the user to fully utilize the power of the synthesis tool. On the one hand, we can push the tool to find complex strategies requiring memory (Offices, Airports, Stars). On the other hand, we keep memory low if it is not needed, enabling the tool to converge to a high-quality strategy (Terrains). Our method also scales well (Building).

A notable exception is the slight under-performance on the Airport benchmark, where for larger graphs, our automatic method achieves worse results than `deg`, see Figure 6. The reason is that this benchmark uses linear targets whose objective function is not bounded, and the convergence of positional strategies is slow.

We hypothesize that modifying the linear objective so that it is bounded would lead to better performance of `auto`, even for larger graphs. The advantage of our approach is that our heuristic would be usable for this modification as well.

Conclusion

An automatic and robust memory assignment tool was a missing part of up-to-date, promising strategy synthesis techniques. Our tool enables users and experts to fully utilize the power of the synthesis tools.

We proposed an automatic memory assignment method for patrolling problems and compared it against uniform and expert, degree-based assignments. Our experiments across diverse benchmarks showed that our automatic method outperformed uniform memory in nearly all cases, offering robust and scalable performance.

Our approach provides a versatile and novel heuristic that is independent of the underlying optimization algorithms and objectives, making it applicable to a wider class of patrolling-type problems and new optimization techniques.

Acknowledgments

Vojtěch Řehák is supported by the Czech Science Foundation, grant No. 26-23441S.

References

Afshani, P.; de Berg, M.; Buchin, K.; Gao, J.; Löffler, M.; Nayyeri, A.; Raichel, B.; Sarkar, R.; Wang, H.; and Yang, H. 2021. Approximation Algorithms for Multi-Robot Patrol-Scheduling with Min-Max Latency. In *Proceedings of WAFR 2021*, volume 17 of *Springer Proceedings in Advanced Robotics*, 107–123. Springer.

Agmon, N.; Kraus, S.; and Kaminka, G. A. 2008. Multi-robot perimeter patrol in adversarial settings. In *Proceedings of 2008 IEEE International Conference on Robotics and Automation*, 2339–2345. IEEE.

Agmon, N.; Kraus, S.; Kaminka, G. A.; and Sadov, V. 2009. Adversarial Uncertainty in Multi-Robot Patrol. In *Proceedings of IJCAI 2009*, 1811–1817.

Almeida, A.; Ramalho, G. L.; Santana, H.; Tedesco, P. A.; Menezes, T.; Corruble, V.; and Chevaleyre, Y. 2004. Recent Advances on Multi-agent Patrolling. In *Proceedings of SBIA 2004*, volume 3171 of *Lecture Notes in Computer Science*, 474–483. Springer.

Alpern, S.; Chleboun, P.; Katsikas, S.; and Lin, K. Y. 2022. Adversarial Patrolling in a Uniform. *Oper. Res.*, 70(1): 129–140.

An, B.; Shieh, E.; Tambe, M.; Yang, R.; Baldwin, C.; DiRenzo, J.; Maule, B.; and Meyer, G. 2012. PROTECT - A Deployed Game Theoretic System for Strategic Security Allocation for the United States Coast Guard. *AI Mag.*, 33(4): 96–110.

Basilico, N. 2022. Recent Trends in Robotic Patrolling. *Current Robotics Reports*, 3(2): 65–76.

Basilico, N.; Gatti, N.; and Amigoni, F. 2009. Leader-follower strategies for robotic patrolling in environments with arbitrary topologies. In *Proceedings of AAMAS 2009*, 57–64. IFAAMAS.

Basilico, N.; Gatti, N.; and Amigoni, F. 2012. Patrolling security games: Definition and algorithms for solving large instances with single patroller and single intruder. *Artif. Intell.*, 184-185: 78–123.

Basilico, N.; Nittis, G. D.; and Gatti, N. 2017. Adversarial patrolling with spatially uncertain alarm signals. *Artif. Intell.*, 246: 220–257.

Biswas, A.; Aggarwal, G.; Varakantham, P.; and Tambe, M. 2021. Learn to Intervene: An Adaptive Learning Policy for Restless Bandits in Application to Preventive Healthcare. In *Proceedings of IJCAI 2021*, 4039–4046. ijcai.org.

Brázdil, T.; Klaška, D.; Kučera, A.; Musil, V.; Novotný, P.; and Řehák, V. 2022. On-the-fly adaptation of patrolling strategies in changing environments. In *Proceedings of UAI 2022*, volume 180 of *Proceedings of Machine Learning Research*, 244–254. PMLR.

Collins, A.; Czyzowicz, J.; Gasieniec, L.; Kosowski, A.; Kranakis, E.; Krizanc, D.; Martin, R.; and Morales-Ponce, O. 2013. Optimal patrolling of fragmented boundaries. In *Proceedings of SPAA 2013*, 241–250. ACM.

de Cote, E. M.; Stranders, R.; Basilico, N.; Gatti, N.; and Jennings, N. R. 2013. Introducing alarms in adversarial patrolling games: extended abstract. In *Proceedings of AAMAS 2013*, 1275–1276. IFAAMAS.

Farinelli, A.; Iocchi, L.; and Nardi, D. 2017. Distributed on-line dynamic task assignment for multi-robot patrolling. *Autonomous Robots*, 41(6): 1321–1345.

Fave, F. M. D.; Jiang, A. X.; Yin, Z.; Zhang, C.; Tambe, M.; Kraus, S.; and Sullivan, J. P. 2014. Game-Theoretic Patrolling with Dynamic Execution Uncertainty and a Case Study on a Real Transit System. *J. Artif. Intell. Res.*, 50: 321–367.

Ford, B. J.; Kar, D.; Fave, F. M. D.; Yang, R.; and Tambe, M. 2014. PAWS: adaptive game-theoretic patrolling for wildlife protection. In *Proceedings of AAMAS 2014*, 1641–1642. IFAAMAS/ACM.

Hari, S. K. K.; Rathinam, S.; Darbha, S.; Kalyanam, K.; Manyam, S. G.; and Casbeer, D. W. 2021. Optimal UAV Route Planning for Persistent Monitoring Missions. *IEEE Trans. Robotics*, 37(2): 550–566.

Ho, H.; and Ouaknine, J. 2015. The Cyclic-Routing UAV Problem is PSPACE-Complete. In *Proceedings of ETAPS 2015*, volume 9034 of *Lecture Notes in Computer Science*, 328–342. Springer.

Huang, L.; Zhou, M.; Hao, K.; and Hou, E. S. H. 2019. A survey of multi-robot regular and adversarial patrolling. *IEEE CAA J. Autom. Sinica*, 6(4): 894–903.

Karwowski, J.; Mandziuk, J.; Zychowski, A.; Grajek, F.; and An, B. 2019. A Memetic Approach for Sequential Security Games on a Plane with Moving Targets. In *Proceedings of AAAI 2019*, 970–977. AAAI Press.

Klaška, D.; Kučera, A.; Lamser, T.; and Řehák, V. 2018. Automatic Synthesis of Efficient Regular Strategies in Adversarial Patrolling Games. In *Proceedings of AAMAS 2018*, 659–666. IFAAMAS.

Klaška, D.; Kučera, A.; Musil, V.; and Řehák, V. 2021. Regstar: efficient strategy synthesis for adversarial patrolling games. In *Proceedings of UAI 2021*, volume 161 of *Proceedings of Machine Learning Research*, 471–481. AUAI Press.

Klaška, D.; Kučera, A.; and Řehák, V. 2020. Adversarial Patrolling with Drones. In *Proceedings of AAMAS 2020*, 629–637. IFAAMAS.

Klaška, D.; Kučera, A.; Musil, V.; and Řehák, V. 2022. Minimizing Expected Intrusion Detection Time in Adversarial Patrolling. In *Proceedings of AAMAS 2022*, 1660–1662. IFAAMAS.

Kučera, A.; and Lamser, T. 2016. Regular Strategies and Strategy Improvement: Efficient Tools for Solving Large Patrolling Problems. In *Proceedings of AAMAS 2016*, 1171–1179. ACM.

Pita, J.; Jain, M.; Marecki, J.; Ordóñez, F.; Portway, C.; Tambe, M.; Western, C.; Paruchuri, P.; and Kraus, S. 2008. Deployed ARMOR protection: the application of a game

theoretic model for security at the Los Angeles International Airport. In *Proceedings of AAMAS 2008*, 125–132. IFAAMAS.

Portugal, D.; and Rocha, R. P. 2011. A Survey on Multi-robot Patrolling Algorithms. In *Proceedings of DoCEIS 2011*, volume 349 of *IFIP Advances in Information and Communication Technology*, 139–146. Springer.

Sinha, A.; Fang, F.; An, B.; Kiekintveld, C.; and Tambe, M. 2018. Stackelberg Security Games: Looking Beyond a Decade of Success. In *Proceedings of IJCAI 2018*, 5494–5501. ijcai.org.

Sless, E.; Agmon, N.; and Kraus, S. 2019. Multi-robot adversarial patrolling: Handling sequential attacks. *Artif. Intell.*, 274: 1–25.

Tambe, M. 2012. *Security and Game Theory - Algorithms, Deployed Systems, Lessons Learned*. Cambridge University Press. ISBN 978-1-10-709642-4.

Tsai, J.; Rathi, S.; Kiekintveld, C.; Ordóñez, F.; Tambe, M.; and Tambe, M. 2011. *IRIS – A Tool for Strategic Security Allocation in Transportation Networks*, 88–106. Cambridge University Press.

Vorobeychik, Y.; An, B.; and Tambe, M. 2012. Adversarial patrolling games. In *Proceedings of AAMAS 2012*, 1307–1308. IFAAMAS.

Wang, Y.; Shi, Z. R.; Yu, L.; Wu, Y.; Singh, R.; Joppa, L.; and Fang, F. 2019. Deep Reinforcement Learning for Green Security Games with Real-Time Information. In *Proceedings of AAAI 2019*, 1401–1408. AAAI Press.

Xu, L. 2021. Learning and Planning Under Uncertainty for Green Security. In *Proceedings of IJCAI 2021*, 4927–4928. ijcai.org.

Yin, Z.; Korzhyk, D.; Kiekintveld, C.; Conitzer, V.; and Tambe, M. 2010. Stackelberg vs. Nash in security games: interchangeability, equivalence, and uniqueness. In *Proceedings of AAMAS 2010*, 1139–1146. IFAAMAS.