Automatic Synthesis of Efficient Regular Strategies in Adversarial Patrolling Games

David Klaška
Faculty of Informatics, Masaryk University
Brno, Czech Republic
xklaska1@fi.muni.cz

Antonín Kučera
Faculty of Informatics, Masaryk University
Brno, Czech Republic
tony@fi.muni.cz

Tomáš Lamser
Faculty of Informatics, Masaryk University
Brno, Czech Republic
xlamser@fi.muni.cz

Vojtěch Řehák
Faculty of Informatics, Masaryk University
Brno, Czech Republic
rehak@fi.muni.cz

ABSTRACT

We give a polynomial-time algorithm for synthesizing efficient regular strategies in adversarial patrolling games with general topology. Regular strategies use finite memory to gather some relevant information about the history of Defender's moves which results in substantially better protection of the targets. So far, the scope of automatic strategy synthesis was limited to positional strategies (which ignore the history) or to regular strategies where the underlying finite-memory observer had to be supplied *manually*. Furthermore, the existing methods do not give any information on how far are the constructed strategies from being optimal. In this paper, we try to overcome these limitations. We develop a novel *gradient-based* algorithm for synthesizing regular strategies where the underlying finite-memory observers are constructed *algorithmically*. The running time of our algorithm is *polynomial* which makes the algorithm applicable to instances of *realistic size*. Furthermore, we develop an algorithm for computing an *upper bound* on the best achievable protection, and compare the quality of the constructed strategies against this bound. Thus, we can effectively measure the "distance" of the constructed strategies from optimal strategies, and our experiments show that this distance is often quite small.

KEYWORDS

Single and multi-agent planning and scheduling, patrolling games.

ACM Reference Format:
David Klaška, Antonín Kučera, Tomáš Lamser, and Vojtěch Řehák. 2018. Automatic Synthesis of Efficient Regular Strategies in Adversarial Patrolling Games. In *Proc. of the 17th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2018), Stockholm, Sweden, July 10–15, 2018,* IFAAMAS, 8 pages.

1 INTRODUCTION

Security games are non-cooperative games where the Defender strives to protect a given set of vulnerable targets against the Attacker using the available security forces. A special variant of security games are *patrolling games*, where the Defender (e.g., a police patrol) moves among the targets and tries to discover possible intrusions. The Defender's moves are constrained by the external environment (streets, passages, etc.) specified as a directed graph, where the targets form a subset of the vertices. The task is to construct a strategy determining the moves of the Defender so that the overall protection is maximized. In *adversarial* patrolling, it is further assumed that the Attacker knows the Defender's strategy and can observe his moves, which leads to *Stackelberg equilibrium* as a natural solution concept.

Solving patrolling games of realistic size is challenging, because even a special variant of this problem is PSPACE-hard [16]. The existing methods for computing Defender's strategies are mostly based on mathematical programming involving *non-linear* constraints, which limits the applicability of these methods only to small instances of patrolling games. Furthermore, the constructed strategies are mostly *positional*, which does not suffice for achieving optimality. A more general class of *regular* strategies has recently been considered in [17]. Here, the history of Defender's moves is observed by a suitable finite-state automaton, and the Defender's strategy depends on the state of the automaton entered after reading the history of the moves performed so far. In [17], it was shown that regular strategies are more powerful than positional strategies, but no *automatic* method for constructing the finite-state automaton is given; it must be supplied manually, which substantially limits the effectiveness of the whole approach. Furthermore, the existing works do not provide any tools allowing to estimate how far are the constructed strategies from being optimal. This is also challenging, because even approximating the best achievable protection up to a non-trivial precision is computationally hard (see Section 5 for more details).

Our contribution. The main result of this paper is a novel gradient-based algorithm for constructing regular Defender's strategies in patrolling games. The algorithm automatically computes a suitable finite-memory structure observing the history. These finite-memory structures are more flexible than finite-state automata used in [17], but retain the same expressive power. Instead of employing mathematical programming with non-linear constraints, our algorithm is based on utilizing symbolic derivatives and proceeds as follows: First, the vertices of a given patrolling game are assigned a suitable number of memory elements based on their significance (which depends on the number of outgoing edges). Then, a suitable

Proc. of the 17th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2018), M. Dastani, G. Sukthankar, E. André, S. Koenig (eds.), July 10–15, 2018, Stockholm, Sweden.

initial regular strategy is chosen, and repeatedly *improved* in a number of rounds. In each round, the most promising "direction" for modifying the current strategy is computed by a certain gradient-based technique, and the parameters are changed slightly in this direction. Performing one such round takes only *polynomial time*, and hence our algorithm is *guaranteed* to terminate in polynomial time if the number of rounds is bounded. This makes the algorithm applicable to patrolling graphs of *realistic size*.

The second contribution of our paper is a general algorithm for computing an upper bound on the maximal protection achievable for a given patrolling game. The algorithm is parameterized by a maximal admissible "attack delay" ℓ, and the computed upper bound is guaranteed to converge to the maximal achievable protection as ℓ increases. The running time is *exponential* in ℓ, which is inevitable because no polynomial-time algorithm for approximating the maximal achievable level of protection can exist unless P=NP (see Section 5). Still, the algorithm often terminates quickly, particularly in situations when the underlying graph of the patrolling game is sparse. For example, the graphs modeling the structure of passages or streets satisfy this property.

We implemented our strategy synthesis algorithm, and the preliminary results reported in Section 6 show that high-quality strategies for realistic instances are computable in reasonable time. The quality is justified *rigorously* by running our second algorithm for estimating the achievable protection. To the best of our knowledge, this is the first framework for solving general patrolling problem which is *guaranteed* to produce a solution quickly and can also *evaluate* the quality of the computed solution.

2 RELATED WORK

Game-theoretic approaches to security problems based on the leader-follower (i.e., Stackelberg) game model have been intensively studied in recent years. Notable practical applications include the deployment of police checkpoints at the Los Angeles International Airport [18], the scheduling of federal air marshals over the U.S. domestic airline flights [19], the arrangement of city guards in Los Angeles Metro [14], the positioning of U.S. Coast Guard patrols to secure selected locations [4], and recently also applications to wildlife protection in Uganda [15].

In patrolling games, the focus was primarily on finding locally optimal strategies for robotic patrolling units either on restricted graphs such as circles [2, 3], or arbitrary graphs with weighted preference on the targets [5, 7]. Alternatively, the work focused on some novel aspects of the problem, such as variants with moving targets [10, 13], multiple patrolling units [8], or movement of the Attacker on the graph [7] and reaction to alarms [9, 12]. Most of the existing literature assumes that the Defender is following a positional strategy that depends solely on the current position of the Defender in the graph and they seek for a solution using mathematical programming. Few exceptions include duplicating each node of the graph to distinguish internal states of the Defender (e.g., in [2] authors consider a direction of the patrolling robot as a specific state; in [11], this concept is further generalized), or seeking for higher-order strategies in [5] and [6]. More recently, the use of regular Defender's strategies was proposed in [17]. In [1], an algorithm for computing an ε-optimal strategy for the Defender is designed, but this algorithm is of exponential complexity.

3 PATROLLING GAMES

The model of patrolling games used in this paper is essentially the same as in [5, 17]. That is, we allow for an arbitrary topology of the game graph, we assume that the time needed to complete an intrusion can be different at each target, and the targets may have different importance. The adopted solution concept is Stackelberg equilibrium, where the Defender/Attacker play the roles of the leader/follower.

A *patrolling game* is a tuple $\mathcal{G} = (V, E, T, d, c)$ where V is a finite set of *vertices* (Defender's positions), $E \subseteq V \times V$ are *edges* (admissible moves of the Defender), $T \subseteq V$ is a set of *targets*, d assigns to every $t \in T$ the number $d(t) > 0$ of time units needed to complete an intrusion at t, and c assigns to every $t \in T$ a positive integer weight specifying its importance (where a higher value means higher importance). We use $c_{\max}$ and $d_{\max}$ to denote $\max\{c(v) \mid v \in T\}$ and $\max\{d(v) \mid v \in T\}$, respectively. We require that the directed graph (V, E) is strongly connected. It can be safely assumed that the Defender spends one unit of time in each vertex, because longer stays can be modeled by inserting auxiliary vertices and edges. A *history* of length ℓ is a finite word $v_1 \dots v_\ell$ over the alphabet V such that $(v_i, v_{i+1}) \in E$ for all $i < \ell$. The empty history is denoted by ε. The set of all histories is denoted by $\mathcal{H}$, and the set of all non-empty histories by $\mathcal{H}^+$. A *walk* is an infinite path in $\mathcal{G}$.

A *Defender's strategy* is a function σ assigning to every history h a probability distribution over the successors of the last vertex in h (if $h = \varepsilon$, then $\sigma(h)$ is a distribution over V). That is, σ specifies how the next vertex is chosen after performing a given finite history, and the choice can be randomized. The initial vertex is chosen according to $\sigma(\varepsilon)$. We say that σ is *positional* if $\sigma(h)$ depends only on the last vertex of h. Every σ determines a unique probability space over all walks in $\mathcal{G}$ in the standard way.

An *Attacker's strategy* is a function $\pi : \mathcal{H}^+ \to \{wait, enter_t \mid t \in T\}$ specifying whether the intruder should wait or try to attack a target t after observing a given history of Defender's moves. The Attacker is allowed to attack at most once along a Defender's walk, which means that if $\pi(h) = enter_t$ for some $t \in T$, then $\pi(h') = wait$ for all proper prefixes h' of h. Given an Attacker's strategy π and a Defender's walk $w = v_1, v_2, \dots$, we say the Attacker *waits along* w if $\pi(h) = wait$ for every finite prefix h of w. If there is $j \geq 1$ such that $\pi(v_1, \dots, v_j) = enter_t$, then we say that the Attacker *attacks* t *along* w, and he is either *captured* or *penetrates* t, depending on whether t appears among the vertices $v_j, \dots, v_{j+d(t)-1}$ or not, respectively. Given a Defender's strategy σ and an Attacker's strategy π, the probability of all walks w such that the Attacker penetrates t is denoted by $\mathcal{P}^\sigma(penetrate_t^\pi)$. Note that $penetrate_t^\pi$ is a measurable event, because it is the union of countably many events of the form "the Defender follows h and does not visit t within the next $d(t) - 1$ steps" over all histories h such that $\pi(h) = enter_t$ and the last vertex of h is not t.

The *expected Attacker's utility* is defined by

$$\mathrm{EU}_A^{\sigma,\pi} = \sum_{t \in T} \mathcal{P}^\sigma(penetrate_t^\pi) \cdot c(t)$$

and the *expected Defender's utility* is defined by

$$\mathrm{EU}_D^{\sigma,\pi} = c_{\max} - \sum_{t \in T} \mathcal{P}^{\sigma}(penetrate_t^{\pi}) \cdot c(t).$$

Note that maximizing the Defender's expected utility is precisely the same objective as minimizing the expected Attacker's utility[1], i.e., the players objectives are precisely opposite. Intuitively, $\mathrm{EU}_A^{\sigma,\pi}$ is the expected amount "stolen" by the Attacker, while $\mathrm{EU}_D^{\sigma,\pi}$ is the expected amount "protected" by the Defender, where $\mathrm{EU}_A^{\sigma,\pi} + \mathrm{EU}_D^{\sigma,\pi} = c_{\max}$.

The *value* of a Defender's strategy σ in $\mathcal{G}$ is defined by

$$Val_{\mathcal{G}}(\sigma) = \inf_{\pi} \mathrm{EU}_D^{\sigma,\pi}$$

where π ranges over all Attacker's strategies. Intuitively, $Val_{\mathcal{G}}(\sigma)$ is the protection guaranteed by σ against an arbitrary Attacker's strategy. The *value of* $\mathcal{G}$ is defined by

$$Val_{\mathcal{G}} = \sup_{\sigma} Val_{\mathcal{G}}(\sigma)$$

where σ ranges over all Defender's strategies, and corresponds to the best possible protection achievable by the Defender. A Defender's strategy σ is *optimal* if $Val_{\mathcal{G}}(\sigma) = Val_{\mathcal{G}}$.

4 GRADIENT-BASED STRATEGY SYNTHESIS

In this section, we present our gradient-based strategy synthesis algorithm for patrolling games. We start by introducing regular strategies.

4.1 Regular strategies

Definition 4.1. Let $\mathcal{G} = (V, E, T, d, c)$ be a patrolling game. A *regular* Defender's strategy for $\mathcal{G}$ is a triple $\eta = (avail, v_0[i_0], \gamma)$, where

- $avail : V \to \mathbb{N}$ is a function assigning to each $v \in V$ the number of memory elements available in v. An *augmented vertex* is an expression of the form $v[i]$, where $v \in V$ and i is a positive integer bounded by $avail(v)$. The set of all augmented vertices is denoted by $\mathcal{A}$. An *augmented edge* is a pair $(v[i], u[j]) \in \mathcal{A} \times \mathcal{A}$ such that $(v, u) \in E$.
- $v_0[i_0] \in \mathcal{A}$ is the initial augmented vertex.
- γ is a *transition function* assigning to each augmented edge its probability in the interval $[0, 1]$ so that, for every $v[i] \in \mathcal{A}$, the sum of the probabilities of all augmented edges of the form $(v[i], u[j])$ is equal to 1.

Intuitively, the Defender executing a regular strategy $\eta = (avail, v_0[i_0], \gamma)$ maintains a bounded counter updated in every move. The initial vertex is v_0 and the initial counter value is i_0. If the currently visited vertex is v and the current counter value is i, the next vertex u and the new counter value j are chosen randomly according to γ, i.e., the probability of choosing given u and j is $\gamma(v[i], u[j])$.

Our definition of regular strategy is somewhat different from the one used in [17], where the finite strategy memory was formalized as a deterministic finite-state automaton over the alphabet V reading the history of visited vertices, and the Defender's moves were determined by the currently visited vertex and the current state of the automaton. Our definition is more general as it allows to simulate the execution of not only deterministic but also *probabilistic* finite-state automata reading the history (note that the counter values can be interpreted as states of a finite-state automaton).

[1] In principle, we could use any other constant instead on $c_{\max}$ (including zero) without influencing the players' rationality. By using $c_{\max}$, the Defender's expected utility becomes positive, which better corresponds to its intuitive interpretation as "achieved protection".

4.2 Strategy synthesis algorithm

For the rest of this section, we fix a patrolling game $\mathcal{G} = (V, E, T, d, c)$. Our algorithm starts by synthesizing the function *avail*. Then, an initial transition function γ_0 is chosen, and it is repeatedly "improved" until a certain termination condition is satisfied and the final γ is produced. When improving a given γ, we first compute an appropriate "direction" for modifying γ, and then "shift" γ in this direction. The direction is obtained by taking a weighted sum of gradients of certain multivariate polynomials in the "weakest points" of γ. Although our algorithm resembles the standard gradient-based approach to approximating local extreme points, it actually differs from the "straightforward" implementation of gradient descent in several important aspects, which are discussed in greater detail in subsequent paragraphs. Our previous experience with unsuccessful implementations shows that these aspects are *essential* for obtaining a working algorithm.

Computing the function *avail*. The complexity of our algorithm depends on the total number of augmented vertices $|\mathcal{A}|$. To keep this parameter under control, our algorithm inputs an integer $M \geq |V|$ specifying the total number of augmented vertices in the constructed regular strategy. Hence, the constructed *avail* must satisfy $\sum_{v \in V} avail(v) = M$.

Intuitively, *avail* should return a higher value for more important vertices, such as crossroads. At the same time, our experience shows that the values assigned by *avail* to neighboring vertices should not be too different, because then the accumulated information is not appropriately transferred when moving from vertex to vertex. Therefore, we use the following procedure for computing *avail*. Let m be the integer part of $M/|V|$, and let $k = M - m \cdot |V|$. Furthermore, let $v_1, \ldots, v_n$ be the vertices of V sorted in descending order according to the number of immediate successors. The function *avail* assigns $m + 1$ to the first k vertices, and m to the remaining vertices.

Restricting the set of augmented edges. When improving a current transition function γ, it may happen that some augmented edges are assigned very small (or even zero) probability. Our practical experience shows that these edges should better be *disregarded* in the next improvement steps, because they tend to negative values, and this tendency may "block" further shifts to better strategies. Therefore, our algorithm keeps a set $\mathcal{E}$ of *eligible* augmented edges that may still be assigned positive probability. The augmented edges *outside* $\mathcal{E}$ are assigned probability zero. The set $\mathcal{E}$ is only *reduced* as the algorithm proceeds; the exact procedure is described below.

Identifying weakest points. To improve a current γ, we need to identify its "weakest points". This is achieved by considering, for every $v[i] \in \mathcal{A}$ and every target $t \in T$, the expected Attacker's utility when the Defender starts in the vertex v, the initial counter value is i, and the Attacker enters t immediately. This expected utility is

denoted by $\mathrm{EU}^{\gamma}_{A}\langle v[i], t\rangle$, and it can be computed as follows. Consider the set $S\langle v[i], t\rangle$ of all sequences $w = u_1[i_1], \ldots, u_{d(t)}[i_{d(t)}]$ initiated in $v[i]$ such that $(u_j[i_j], u_{j+1}[i_{j+1}]) \in \mathcal{E}$ for all $j < d(t)$, and $u_j \neq t$ for all $j \leq d(t)$. Then

$$\mathrm{EU}^{\gamma}_{A}\langle v[i], t\rangle \;=\; c(t) \cdot \sum_{w \in S\langle v[i], t\rangle} \prod_{j=1}^{d(t)-1} \gamma(w_j, w_{j+1}) \qquad (1)$$

where w_j denotes the j-th augmented vertex in the sequence w. Furthermore, we put

$$\mu[\gamma] \;=\; \max\left\{\mathrm{EU}^{\gamma}_{A}\langle v[i], t\rangle \mid v[i] \in \mathcal{A}, t \in T\right\}.$$

Since we aim at *minimizing* the expected Attacker's utility, the weakest point of γ is a pair $\langle v[i], t\rangle$ such that $\mathrm{EU}^{\gamma}_{A}\langle v[i], t\rangle = \mu[\gamma]$. In our improvement step, we actually consider all points that are ε-close to $\mu[\gamma]$. According to our previous experience, this is necessary for achieving a good "improvement direction". Formally, we define the set

$$Wp^{\varepsilon}[\gamma] \;=\; \left\{\langle v[i], t\rangle \;\mid\; \mu[\gamma] - \mathrm{EU}^{\gamma}_{A}\langle v[i], t\rangle \;<\; \varepsilon\right\}$$

Here, $\varepsilon > 0$ is a suitable constant (in our experiments, setting $\varepsilon = 0.03 \cdot c_{\max}$ produces rather good results).

Computing the improvement direction. Now we show how to compute a suitable "direction" for improving the current γ.

For every $v[i] \in \mathcal{A}$, we fix an augmented edge $(v[i], u[j]) \in \mathcal{E}$ called a *pivot*, and define $Out(v[i])$ as the set of all augmented edges of the form $(v[i], u'[j']) \in \mathcal{E}$ such that $(v[i], u'[j'])$ is *not* a pivot. For every $v[i] \in \mathcal{A}$ and every $(v[i], u'[j']) \in Out(v[i])$, we fix a fresh variable $X(v[i], u'[j'])$.

Now, consider the *symbolic* expression $\mathrm{EU}_A\langle v[i], t\rangle$ obtained from the right-hand side of Equation (1) by substituting every $\gamma(w_j, w_{j+1})$ with either

$$1 - \sum_{(w_j, u'[j']) \in Out(w_j)} X(w_j, u'[j'])$$

or $X(w_j, w_{j+1})$, depending on whether (w_j, w_{j+1}) is a pivot or not, respectively. Observe that $\mathrm{EU}_A\langle v[i], t\rangle$ is a *multivariate polynomial*. Hence, we can easily compute all of its symbolic partial derivatives, and thus also the *gradient of* $\mathrm{EU}_A\langle v[i], t\rangle$ *at the point* γ, denoted by $Grad[\mathrm{EU}_A\langle v[i], t\rangle](\gamma)$. Strictly speaking, $Grad[\mathrm{EU}_A\langle v[i], t\rangle](\gamma)$ denotes the gradient of $\mathrm{EU}_A\langle v[i], t\rangle$ at the point obtained by *restricting* γ to eligible augmented transitions which are not pivots.

Note that $Grad[\mathrm{EU}_A\langle v[i], t\rangle](\gamma)$ can be seen as a function returning a concrete numerical value for every eligible augmented edge which is not a pivot. We extend the domain of $Grad[\mathrm{EU}_A\langle v[i], t\rangle](\gamma)$ to the set of *all* augmented edges as follows:

- For all augmented edges outside $\mathcal{E}$, the function $Grad[\mathrm{EU}_A\langle v[i], t\rangle](\gamma)$ returns zero.
- For every pivot $(v[i], u[j])$, the function $Grad[\mathrm{EU}_A\langle v[i], t\rangle](\gamma)$ returns $1 - \mathrm{Sum}$, where Sum is the sum of all values returned by $Grad[\mathrm{EU}_A\langle v[i], t\rangle](\gamma)$ for the augmented edges of $Out(v[i])$.

Now, all of the obtained gradients (extended in the above described way) are combined together by taking their weighted sum, where the weight of a given $Grad[\mathrm{EU}_A\langle v[i], t\rangle](\gamma)$ is set to $(EU^{\gamma}_{A}\langle v[i], t\rangle - \mu[\gamma] + \varepsilon)/\varepsilon$. That is, gradients computed for $\langle v[i], t\rangle$ with higher $\mathrm{EU}^{\gamma}_{A}\langle v[i], t\rangle$ have higher weight. The resulting function is denoted by $Dir[\gamma]$.

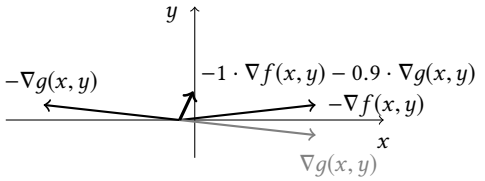

Figure 1: An illustration of why considering only the weakest point is poor.

Some notes. One may ask why is the improvement direction computed in such a "complicated way". The reasons are worth explaining, because they reflect our previous experience with unsuccessful naive implementations of gradient descent. A short summary is given in the next two paragraphs.

Intuitively, $Dir[\gamma]$ is a direction which should improve (i.e., decrease) the Attacker's expected utility for all weak points $\langle v[i], t\rangle \in Wp^{\varepsilon}[\gamma]$ *simultaneously*. If we considered only the gradient of the *weakest point*, this could lead to poor behavior. To illustrate this phenomenon, consider a simplified situation where the penetration probabilities are given by the functions $f(x, y) = -9x - y$ and $g(x, y) = 9x - y$. Then, the Attacker's expected utility is the maximum of f and g. Suppose we are currently at the point $(x, y) = (-1, 0)$, so the maximum is reached by f. The gradient of f evaluates to $(-9, -1)$, but the direction $(9, 1)$ is very close to the direction $(9, -1)$ (depicted in gray in Figure 1), along which g rises most rapidly. However, considering also the gradient of g (say, with coefficient 0.9), this yields the direction $(0.9, 1.9)$ along which both functions are decreasing.

Let us further note that if we fixed a fresh variable $X(v[i], u[j])$ for *all* eligible augmented edges (without introducing any pivots) and simply substituted every $\gamma(w_j, w_{j+1})$ in the right-hand side of Equation (1) with $X(w_j, w_{j+1})$, the gradient of the resulting multivariate polynomial would not reflect that, for every $v[i]$, the sum of all $X(v[i], u[j])$ must be equal to one. Ignoring this relationship among $X(v[i], u[j])$, where increasing one variable must be compensated by decreasing the others, leads to poor results in practice (although this is perhaps the most natural way of implementing gradient descent).

Improving the current transition function. Intuitively, improving the current γ simply means to add $Dir[\gamma]$ to γ. However, there are still some issues. First, all augmented edges for which $\gamma + Dir[\gamma]$ returns a value smaller than δ are removed from $\mathcal{E}$ (here, $\delta > 0$ is another parameter of our algorithm). If $\mathcal{E}$ is reduced, the current γ is modified so that it returns zero for all elements outside $\mathcal{E}$ (the complement of $\mathcal{E}$ is denoted by $\mathcal{E}^c$, and the resulting transition function by $\gamma\langle\mathcal{E}^c := 0\rangle$), and the probabilities of the remaining augmented edges are normalized so that, for every $v[i] \in \mathcal{A}$, the sum of the probabilities of all augmented edges of the form $(v[i], u[j])$ is again equal to one. Otherwise, we evaluate the protection achieved by $\gamma + Dir[\gamma]$ using the evaluation procedure described below. If $\gamma + Dir[\gamma]$ dominates γ (i.e., achieves better protection than γ), we set $\gamma = \gamma + Dir[\gamma]$ and continue with the next iteration. Otherwise,

we try to "shorten" $Dir[\gamma]$ by repeatedly setting $Dir[\gamma] = Dir[\gamma]/2$, until the shift by $Dir[\gamma]$ yields an improvement or the number of trials reaches a fixed bound *smax*. In the latter case, the algorithm terminates immediately because no substantial progress can be achieved any further.

Termination condition. As we already mentioned in the previous paragraph, our algorithm terminates if the current γ cannot be improved by a shift $Dir[\gamma]/2^{smax}$. To ensure fast termination, we also set a fixed bound *imax* on the maximal number of improvement steps. That is, when the total number of improvement steps reaches *imax*, our algorithm terminates.

Computing the initial augmented vertex. After computing the final transition function γ, we determine the initial augmented vertex $v_0[i_0]$. This is achieved as follows. We construct a directed graph where $\mathcal{A}$ is the set of vertices, and the edges $\hat{E}$ are given by $(v[i], u[j]) \in \hat{E}$ iff $\gamma(v[i], u[j]) > 0$. Then, we identify the bottom strongly connected components $B_1, \ldots, B_n$ of $(\mathcal{A}, \hat{E})$. Since the Defender visits some of these components with probability one when playing according to γ (regardless where he starts), he can only improve the achieved protection by starting directly in the most convenient B_j. Since all augmented vertices of B_j are then visited infinitely often with probability one, the achieved protection is equal to

$$c_{\max} - \max\{\mathrm{EU}_A^{\gamma}\langle v[i], t\rangle \mid v[i] \in B_j, t \in T\}\,.$$

Hence, the optimal B_j is the one maximizing the above expression, and $v_0[i_0]$ can be set to any augmented vertex of this optimal B_j. This also explains how to compute the protection achieved by a given transition function γ, which was needed in one of the previous paragraphs.

The obtained strategy synthesis algorithm is shown in Algorithm 1. Note that Algorithm 1 is parameterized by the constants ε, δ, *smax*, and *imax*, which need to be set to appropriate values (see Section 6). Furthermore, we need to determine the initial transition function γ_0 and the initial set $\mathcal{E}_0$ of eligible edges. Usually, $\mathcal{E}_0$ contains all augmented edges, and γ_0 is chosen either randomly, or set to some specific transition function (one good option is to choose γ_0 so that it selects among all augmented edges uniformly at random).

The running time of Algorithm 1 is polynomial in M, $d_{\max}$ and the size of $\mathcal{G}$. This can be shown by analyzing the time needed for computing one iteration of the main loop. It is not completely obvious that one improvement step can be completed in polynomial time, because the right-hand side of Equation 1 is of exponential size. This is overcome by using dynamic programming, and thus avoiding the explicit construction of this large expression.

5 EVALUATING DEFENDER'S STRATEGIES

As we already mentioned in Section 1, the existing literature does not offer any general method allowing to estimate how far is the constructed Defender's strategy from being optimal. Ideally, we would like to compare the value of the constructed strategy with the value of the game (i.e., the value of an optimal Defender's strategy). Unfortunately, no polynomial-time algorithm computing (or even *approximating*) the value can exist unless P = NP:

Algorithm 1: Strategy synthesis.

input : $\mathcal{G} = (V, E, T, d, c)$, M such that $M \geq |V|$
output : A regular strategy $\eta = (avail, v_0[i_0], \gamma)$

```
compute avail
E ← E_0
γ ← γ_0
i ← 0
repeat
    j ← 0
    i ← i + 1
    compute Wp^ε[γ]
    compute Dir[γ]
    E' ← {τ ∈ E | γ(τ)+Dir[γ](τ) < δ}
    if E' ≠ ∅ then
        E ← E \ E'
        γ ← γ⟨E^c := 0⟩
        γ ← normalize(γ)
    else
        while γ dominates γ + Dir[γ] and j ≤ smax do
            Dir[γ] ← Dir[γ]/2
            j ← j + 1
        end
        if γ does not dominate γ + Dir[γ] then
            γ ← γ + Dir[γ]
        end
    end
until i = imax or j > smax
compute the initial augmented vertex v_0[i_0]
return (avail, v_0[i_0], γ)
```

Theorem 5.1. *It is NP-hard to distinguish whether the value of a patrolling game is equal to* 1 *or bounded by* $1 - 1/N$*, where* N *is the number of vertices.*

To see this, consider the NP-complete *Hamiltonian cycle problem*; an instance is a directed graph $\mathcal{G}$ and the question is whether there exists a Hamiltonian cycle visiting each vertex exactly once. We can interpret $\mathcal{G}$ as a patrolling game where all vertices are targets, $c(v) = 1$, and $d(v) = N$ for all v, where N is the number of vertices. If there is a Hamiltonian cycle in $\mathcal{G}$, then the value of the associated patrolling game is equal to 1; and if there is no Hamiltonian cycle, the value is bounded by $1 - 1/N$.

Consequently, there is no polynomial-time algorithm approximating the value of a given patrolling game with N vertices up to the precision $1/2N$, where N is the number of vertices, unless P = NP (if there was such an algorithm, it could be used to decide the NP-hard Hamiltonian cycle problem).

In light of Theorem 5.1, we can hardly expect to construct a polynomial-time algorithm approximating the value of a given patrolling game. In this section, we design an algorithm computing an *upper bound* for the value, which is guaranteed to converge to $Val(\mathcal{G})$ as a certain parameter ℓ increases. The running time is (unavoidably) *exponential* in ℓ, but the algorithm can produce

relatively good bounds even for small ℓ's. Furthermore, it is especially well-suited for patrolling graphs where the vertices have a small number of outgoing edges. For example, patrolling graphs modeling an ATM network in a city or the structure of passages in some building have this property.

For the rest of this section, we fix a patrolling game $\mathcal{G} = (V, E, T, d, c)$. Our algorithm (see Algorithm 2) uses the following simple observations about optimal strategies:

Lemma 5.2. *Let $\mathcal{G} = (V, E, T, d, c)$ be a patrolling game. Then the Defender can commit to an optimal strategy σ such that every vertex $v \in V$ is either not visited at all, or it is visited infinitely often with probability one. Furthermore, every target $v \in T$ such that $c(v) > Val_{\mathcal{G}}$ is visited infinitely often.*

A proof of Lemma 5.2 is simple and standard[2]. Algorithm 2 gradually computes a set U of vertices that must be visited infinitely often when the Defender commits to an optimal strategy, and terminates when U cannot be extended any further (new additions to U are first collected in an auxiliary set Aux). The set U is initialized to the set of all targets with the maximal weight (line 1). Observe that the Attacker can choose an *arbitrary* vertex $u \in U$ and wait until the Defender visits u, which happens with probability 1. Our algorithm proceeds by evaluating the expected utility of the Attacker when he decides to attack in at most ℓ steps after the Defender visits the chosen $u \in U$. This is achieved by constructing and solving a zero-sum matrix game $G_{u,\ell}$, where the maximizer/minimizer corresponds to the Attacker/Defender. The equilibrium value $Eq(G_{u,\ell})$ then corresponds to the expected utility of the Attacker. Since the Attacker can wait until the Defender visits u and then "simulate" the maximizer's strategy from the equilibrium strategy profile computed for $G_{u,\ell}$, we can conclude that $Val_{\mathcal{G}} \leq c_{\max} - Eq(G_{u,\ell})$. This explains line 4. Then, at line 5, we extend U with all targets $v \in T$ whose weight is larger than the current upper bound on the value (see the second part of Lemma 5.2). Finally, we extend U with all vertices that must *inevitably* be visited when moving among the vertices contained in U (see line 6). Formally, $Attractor(Aux)$ consists of all $v \in V$ for which there exist vertices $t, u \in Aux$ such that *every* path from t to u passes through v.

It remains to explain the construction of the zero-sum matrix game $G_{u,\ell}$. The actions of the minimizer (modeling the Defender) are all histories h of length precisely $d_{\max} + \ell$ initiated in u. The actions of the maximizer (modeling the Attacker) correspond to Attacker's strategies such that an attack is performed in at most ℓ steps after visiting u. Formally, let $\mathcal{H}^{+}_{u,\ell+1}$ be all non-empty histories of length at most ℓ+1 initiated in u. The maximizer's actions are all functions $\pi : \mathcal{H}^{+}_{u,\ell+1} \to \{wait, enter_t \mid t \in T\}$ satisfying the following conditions:

- If $\pi(h) = enter_t$, then $\pi(h') = wait$ for every proper prefix h' of h.
- For every $h \in \mathcal{H}_{u,\ell+1}$ of length ℓ+1, there exist a prefix h' of h (not necessarily proper) such that $\pi(h') \neq wait$.

Intuitively, the second condition ensures that the Attacker performs some attack in at most ℓ steps after visiting u no matter what the Defender does. For every pair of actions (h, π), there is a unique decomposition of h into $h = h_1 v h_2$ such that the length of $h_1 v$ is at most ℓ+1 and $\pi(h_1 v) = enter_t$ for some $t \in T$. The maximizer's payoff for (h, π) is either 0 or $c(t)$, depending on whether t appears among the first $d(t)$ elements of vh_2 or not, respectively.

[2]If σ is an optimal strategy visiting some vertex only finitely often with positive probability, one can easily show the existence of another optimal strategy σ' where this probability is (arbitrarily) small. Consequently, there is an optimal strategy σ'' where this probability is zero. The second part of Lemma 5.2 is trivial.

Algorithm 2: Computing an upper bound on $Val(\mathcal{G})$.

input : $\mathcal{G} = (V, E, T, d, c)$, $\ell \in \mathbb{N}_0$
output : An upper bound on $Val(\mathcal{G})$

1 $Aux \leftarrow \{v \in T \mid c(v) = c_{\max}\}$
2 **repeat**
3 $\quad U \leftarrow Aux$
4 $\quad \mu \leftarrow c_{\max} - \max\{Eq(G_{u,\ell}) \mid u \in U\}$
5 $\quad Aux \leftarrow \{v \in T \mid c(v) > \mu\}$
6 $\quad Aux \leftarrow Attractor(Aux)$
7 **until** $U = Aux$
8 **return** μ

Observe that the action spaces of the minimizer and the maximizer grow exponentially in ℓ. However, if the number of outgoing edges is small for most vertices, and $d_{\max}$ is not too large, the size of $G_{u,\ell}$ may stay reasonable and admit algorithmic analysis. For an increasing ℓ, we obtain better and better approximation of the infinite-horizon patrolling game, and hence the computed μ converges to $Val_{\mathcal{G}}$.

6 EXPERIMENTAL RESULTS

In this section we evaluate Algorithms 1 and 2 experimentally, concentrating on specific instances modeling patrolling in large buildings (such as galleries or museums), where a guard keeps visiting the individual rooms, paying more attention to those with higher value. Since the passages can be walked in both directions, the edge relation is symmetric. Furthermore, the number of passages connected to a given room is typically small.

A concrete example is shown in Figure 2. The building consists of four floors connected by three stairways (left, middle, right). The rooms at each floor are connected by a central corridor connecting the rooms. We assume that each room is a potential target, and its value depends on the total value of all items stored in the room. This determines the weights assigned to vertices in Figure 2. The time needed to complete an attack in each room is the same and set to 15 time units.

The patrolling game of Figure 2 has 28 vertices, which is far beyond the reach of the existing strategy synthesis algorithms based on mathematical programming (the number of nonlinear constraints in these programs is larger than 20^{15}). Also observe that a good Defender's strategy cannot be constructed by hand. The Defender needs to randomize to achieve an optimal protection, and he should possibly also return to previously visited rooms with some small probability (particularly if these rooms have a high weight). A naive approach cannot take into account all of these aspects, the problem is inherently complicated.

Algorithms 1 and 2 can process the instance of Figure 2 without any problems. The upper bound computed by Algorithm 2 when

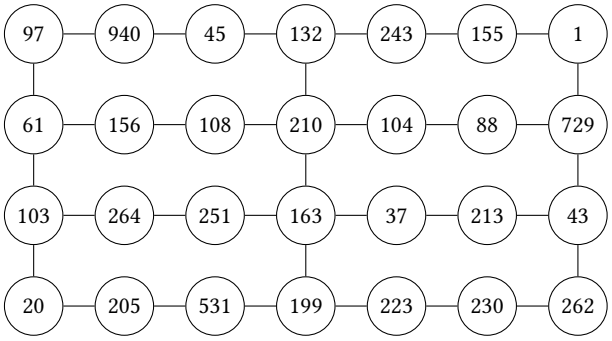

Figure 2: A patrolling game representing a building.

M	$Val_{\mathcal{G}}(\eta)$	Avg. execution time [s]
$28 \cdot 1$	321.629	0.001
$28 \cdot 2$	619.476	0.076
$28 \cdot 3$	632.721	0.142
$28 \cdot 4$	649.356	0.515
$28 \cdot 5$	649.355	0.707
$28 \cdot 6$	674.371	22.545

Table 1: Results achieved for the building of Figure 2.

setting $\ell = 3$ is equal to 679.991. For $\ell = 4$, our experimental implementation of Algorithm 2 did not terminate in reasonable time. The parameters of Algorithm 1 were set as follows: $\varepsilon = 0.03 \cdot c_{\max}$, $\delta = 0.01$, $smax = 10$, $imax = 1000$. In this setup, the number of iterations of the main loop in Algorithm 1 was always smaller than $imax$, i.e., the improvement was always terminated in situations when no further progress was achievable (i.e., the variable j reached the value $smax$). The initial transition function γ_0 was set randomly, and $\mathcal{E}_0$ was set to all augmented edges.

We experimented with a different amount of available memory elements. Initially, we set $M = 28$, which corresponds to constructing a *positional* strategy (i.e., $avail(v) = 1$ for each vertex). Then, we tried to increase M to higher multiples of 28 so that $avail(v)$ is equal to $2, \ldots, 6$ for each vertex. Since the initial γ_0 is chosen randomly, we ran Algorithm 1 repeatedly one hundred times, each time with a freshly chosen γ_0, and collected the best result.

The efficiency of the constructed regular strategies increases with increasing M. The results are summarized in Table 1, which also shows the average execution time[3] of our experimental implementation of Algorithm 1 on an average PC (that is, we report the average execution time of the 100 independent runs of Algorithm 1). Note that the computed positional strategy performs rather poorly, but for $M = 28 \cdot 6$, the value of the obtained regular strategy reaches 674.371, which is quite close to the upper bound 679.991 computed by Algorithm 2.

The behaviour of the regular strategy constructed for $M = 28 \cdot 6$ cannot be described intuitively. In practice, the strategy needs to be implemented in a small electronic device carried by a human patroller, which determines the next room to visit on-the-fly automatically. Note that the device needs to "remember" the table defining the transition function of the constructed regular strategy, and implement a random generator to perform the randomized choice.

[3]The running time data only illustrate that Algorithm 2 can indeed process instances of realistic size in reasonable time. More careful implementations would certainly terminate in much lower time.

floors	rooms on fl.	stairways	$c_{\max}$	$Val_{\mathcal{G}}(\eta)$	bound
3	5	3	886	562.538	568.547
3	5	3	992	679.995	684.940
3	7	1	855	604.364	609.948
3	8	3	635	402.893	409.388
4	7	3	940	674.371	679.991
4	7	3	824	569.397	575.641
5	6	3	996	635.702	640.809
4	10	3	930	627.643	633.746
5	9	3	878	592.211	596.384
5	9	3	996	632.857	636.012
4	12	1	820	519.670	525.428
5	10	3	973	672.765	678.489

Table 2: A summary of experiments.

To evaluate our algorithms on a wider set of instances, we experimented with a dozen of different building models. The summary of our experiments is given in Table 2. The shape of the considered buildings is always rectangular, parameterized by the number of floors, the number of rooms at each floor, and the number of stairways connecting the floors (the building of Figure 2 has three stairways). The weights of rooms are assigned randomly, we report just the maximal weight $c_{\max}$. The parameters of Algorithms 1 and 2 are set to the same values as above. The M was always set to $6 \cdot |V|$, where the number of vertices is always equal to the number of floors times the number of rooms at each floor. The results are equally satisfactory as the ones obtained for our first example of Figure 2.

7 CONCLUSIONS

We presented an efficient gradient-based algorithm for computing regular Defender's strategies in adversarial patrolling games. We demonstrated the applicability of our algorithm to instances of realistic size and shown that the quality of computed strategies is *provably* close to optimal.

An interesting open question is whether regular strategies are sufficiently powerful to achieve optimality. That is, we wonder whether for each patrolling game $\mathcal{G}$ there exists a *regular* Defender's strategy η such that $Val_{\mathcal{G}}(\eta) = Val_{\mathcal{G}}$. It is known that strategies depending only on the last k visited vertices, where k is some constant, are insufficient to achieve optimality (see [17]). In particular, this applies to positional strategies. However, no example witnessing the insufficiency of regular strategies has so far been given, and we conjecture that regular strategies *are* sufficiently powerful to achieve optimal protection.

Since the preliminary experimental results reported in Section 6 are encouraging, a natural question is what are the actual application limits of the proposed framework. Given the computational hardness of the studied problems, we cannot expect that Algorithm 1 works equally well for *all* types of instances, and a more

intensive testing would probably identify some class of problematic instances. Still, we conjecture that Algorithm 1 would produce high-quality solutions after "tuning" its key parameters on small instances with the help of Algorithm 2.

Another direction for future research are patrolling problems with several patrollers, possibly under specific assumptions about their coordination.

ACKNOWLEDGEMENT

The authors are supported by the Czech Science Foundation, grant No. 18-11193S.

REFERENCES

[1] M. Abaffy, T. Brázdil, V. Řehák, B. Bošanský, A. Kučera, and J. Krčál. 2014. Solving Adversarial Patrolling Games with Bounded Error. In *Proceedings of AAMAS 2014*. 1617–1618.

[2] N. Agmon, S. Kraus, and G. Kaminka. 2008. Multi-Robot Perimeter Patrol in Adversarial Settings. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA2008)*. IEEE Computer Society Press, 2339–2345.

[3] N. Agmon, V. Sadov, G. A. Kaminka, and S. Kraus. 2008. The impact of adversarial knowledge on adversarial planning in perimeter patrol. In *Proceedings of AAMAS 2008*. 55–62.

[4] B. An, E. Shieh, R. Yang, M. Tambe, C. Baldwin, J. DiRenzo, B. Maule, and G. Meyer. 2014. Protect—A Deployed Game Theoretic System for Strategic Security Allocation for the United States Coast Guard. *AI Magazine* 33, 4 (2014), 96–110.

[5] N. Basilico, N. Gatti, and F. Amigoni. 2009. Leader-follower strategies for robotic patrolling in environments with arbitrary topologies. In *AAMAS*. 57–64. http://portal.acm.org/citation.cfm?id=1558020

[6] N. Basilico, N. Gatti, and F. Amigoni. 2012. Patrolling security games: Definition and algorithms for solving large instances with single patroller and single intruder. *Artificial Intelligence* 184–185 (2012), 78–123.

[7] N. Basilico, N. Gatti, T. Rossi, S. Ceppi, and F. Amigoni. 2009. Extending algorithms for mobile robot patrolling in the presence of adversaries to more realistic settings. In *WI-IAT*. 557–564.

[8] N. Basilico, N. Gatti, and F. Villa. 2010. Asynchronous Multi-Robot Patrolling against Intrusion in Arbitrary Topologies. In *AAAI*. http://home.dei.polimi.it/ngatti/Download/Papers/BasilicoGattiVilla-AAAI2010.pdf

[9] N. Basilico, G. De Nittis, and N. Gatti. 2016. A Security Game Combining Patrolling and Alarm-Triggered Responses Under Spatial and Detection Uncertainties. In *Proceedings of AAAI 2016*. 404–410.

[10] B. Bosansky, V. Lisy, M. Jakob, and M. Pechoucek. 2011. Computing Time-Dependent Policies for Patrolling Games with Mobile Targets. In *AAMAS*.

[11] B. Bosansky, O. Vanek, and M. Pechoucek. 2012. Strategy Representation Analysis for Patrolling Games. In *AAAI Spring Symposium*.

[12] E. Munoz de Cote, R. Stranders, N. Basilico, N. Gatti, and N. Jennings. 2013. Introducing alarms in adversarial patrolling games: extended abstract. In *AAMAS*. 1275–1276. http://dl.acm.org/citation.cfm?id=2484920.2485180

[13] F. Fang, A. X. Jiang, and M. Tambe. 2013. Optimal Patrol Strategy for Protecting Moving Targets with Multiple Mobile Resources. In *AAMAS*.

[14] F.M. Delle Fave, A.X. Jiang, Z. Yin, C. Zhang, M. Tambe, S. Kraus, and J. Sullivan. 2014. Game-Theoretic Security Patrolling with Dynamic Execution Uncertainty and a Case Study on a Real Transit System. *Journal of Artificial Intelligence Research* 50 (2014), 321–367.

[15] B. Ford, D. Kar, F.M. Delle Fave, R. Yang, and M. Tambe. 2014. PAWS: Adaptive Game-Theoretic Patrolling for Wildlife Protection. In *Proceedings of AAMAS 2014*. 1641–1642.

[16] H.-M. Ho and J. Ouaknine. 2015. The Cyclic-Routing UAV Problem is PSPACE-Complete. In *Proceedings of FoSSaCS 2015 (Lecture Notes in Computer Science)*, Vol. 9034. Springer, 328–342.

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

[18] J. Pita, M. Jain, J. Marecki, F. Ordónez, C. Portway, M. Tambe, C. Western, P. Paruchuri, and S. Kraus. 2008. Deployed ARMOR Protection: The Application of a Game Theoretic Model for Security at the Los Angeles Int. Airport. In *Proceedings of AAMAS 2008*. 125–132.

[19] J. Tsai, S. Rathi, C. Kiekintveld, F. Ordóñez, and M. Tambe. 2009. IRIS—A Tool for Strategic Security Allocation in Transportation Networks Categories and Subject Descriptors. In *Proceedings of AAMAS 2009*. 37–44.