Using One-Sided Partially Observable Stochastic Games for Solving Zero-Sum Security Games with Sequential Attacks*

Petr Tomášek[1], Branislav Bošanský[1], and Thanh H. Nguyen[2]

[1] Artificial Intelligence Center, Dept. of Computer Science
Faculty of Electrical Engineering, Czech Technical University in Prague
{petr.tomasek, branislav.bosansky}@fel.cvut.cz
[2] Department of Computer and Information Science
University of Oregon
tnguye11@uoregon.edu

Abstract. Security games are a defender-attacker game-theoretic model where the defender determines how to allocate scarce resources to protect valuable targets against the attacker. A majority of existing work has focused on the one-shot game setting in which the attacker only attacks once. However, in many real-world scenarios, the attacker can perform multiple attacks in a sequential manner and leverage observable effects of these attacks for better attack decisions in the future. Recent work shows that in order to provide effective protection over targets, the defender has to take the prospect of sequential attacks into consideration. The algorithm proposed by existing work to handle sequential attacks, however, can only scale up to two attacks at most. We extend this line of work and focus on developing new *scalable* algorithms for solving the zero-sum variant of security games. We formulate security games with sequential attacks as a one-sided partially observable stochastic games. We show that the uncertainty about the state in the game can be modeled compactly and we can use variants of heuristic search value iteration algorithm for solving these games. We give two variants of the algorithm – an exact one and a heuristic formulation where the resource reallocation possibilities of the defender are simplified. We experimentally compare these two variants of the algorithm and show that the heuristic variant is typically capable of finding high-quality strategies while scaling to larger scenarios compared to the exact variant.

Keywords: Security games · Sequential attacks · Partially observable stochastic games · Zero-sum games.

* This research was supported by the Czech Science Foundation (no. 19-24384Y) and by the OP VVV MEYS funded project CZ.02.1.01/0.0/0.0/16 019/0000765 "Research Center for Informatics".

1 Introduction

Defender-attacker security games are a well-known class of resource allocation games where a defender has to protect a set of targets against an attacker. The defender chooses how to allocate his limited resources to these targets while the attacker chooses which target(s) to attack. In practice, the *one-shot* security game setting was used in several successful applications, in which the attacker is assumed to attack only *once* [5, 13, 8, 6, 1, 9]. However, in many real-world security domains, the attacks might occur *sequentially* – the attacker can choose to attack targets in a sequence while observing the results of executed attacks. Performing a sequential attack is beneficial for the attacker due to discovered knowledge (by attacking a target, the attacker can partially discover the current allocation of the defending units). Only recently, a new model of *security games with sequential attacks* (`SGSA`) has been introduced [7] showing that it is indeed necessary for the defender to be prepared for the sequential attacks.

In `SGSA`, both players choose their actions *simultaneously* over several rounds — each round corresponds to a simple security game (in which the defender allocates the resources to the targets and the attacker chooses one target to attack). Afterwards, the outcome of the actions is determined — if the attacked target has been unprotected (protected, respectively), the attack is successful (unsuccessful). The game then enters into the next round while assuming that the attacked target is no longer available for protection/attack. Moreover, if the attack was unsuccessful, the defending unit that was present at the target cannot be reallocated to protect other targets. The initial work [7] introduced several variants of `SGSA` and showed that it is indeed better for the attacker to attack in sequence. Therefore, the defender must take the possibility of sequential attacks into consideration and provided an algorithm for computing Strong Stackelberg equilibrium for selected variants. However, the general algorithms for solving `SGSA` are missing. The existing algorithms for computing a Strong Stackelberg equilibrium for `SGSA` are restricted to two rounds only (the attacker can perform two attacks) if the defender is able to reallocate the units and it is not clear whether a generalization to multiple rounds is possible.

In this work, we attempt to address this computation limitation of the previous work, with the following main contributions. First, we leverage recent advancement in solving sub-classes of zero-sum *partially observable stochastic games (`POSGs`)* in which one player has perfect information and the other player has partial information, termed *one-sided `POSGs` (`OS-POSGs`)* [4, 3]. Algorithms for solving `OS-POSG`s are based on a heuristic search value iteration (`HSVI`) algorithm and are capable of handling very long horizons. We show that zero-sum `SGSA` can be formulated as a `OS-POSG`s, thus allowing us to use the existing algorithms of solving `OS-POSG`s. Second, we develop a new compact representation for `SGSA` to avoid the exploration of an exponential number of states (due to exponentially many possible subsets of protected targets) involved in the computation of the original `HSVI` algorithm. While the idea behind using the compact representation in `HSVI` has been introduced for a lateral-movement game in computer networks [2, 3], the technical realization of this idea in the domain of security

games is non-trivial and novel. Third, in order to further improve the scalability, we introduce a heuristic variant of the game where we introduce a mild restriction for the defending units — each target can be in one stage protected only by one unit, and this allocation of units is determined heuristically. While this heuristic partitioning of targets among the defending units can negatively affect the quality of defending strategies, our experimental evaluation shows that with an increasing number of targets, the quality of strategies is very close to the exact formulation. Moreover, the heuristic variant scales to larger scenarios. Finally, we conduct extensive experiments to evaluate proposed methods. We show that (1) ignoring the sequential aspect and solving each round separately results in strategies with poor quality and that (2) our methods can solve larger `SGSA` with multiple rounds (which the existing algorithm cannot handle) while maintaining high-quality strategies for the players in the game.

2 Technical Background

In this section, we first provide the basic definitions for one-sided partially observable stochastic games (`OS-POSG`s) and describe the ideas behind the heuristic search value iteration (`HSVI`) algorithm. We then formally define security games with sequential attacks (`SGSA`s).

2.1 One-sided Partially Observable Stochastic Games (`OS-POSG`)

`OS-POSG` [4] is an imperfect-information two-player zero-sum infinite-horizon game with perfect recall, formally defined as a tuple $G = \langle S, A_1, A_2, O, \tau, \rho \rangle$. The game evolves in rounds, where in each round a *stage game* is played. At each stage, the game is in one of the states $s \in S$ and players simultaneously pick their actions $a_1 \in A_1$ and $a_2 \in A_2$. The initial state of the game is drawn from a probability distribution $b^0 \in \Delta(S)$ over the set of states S, which is treated as a parameter of the game and termed *the initial belief*. The one-sided nature of the game results in the fact that while player 2 can observe the game perfectly (i.e., his only uncertainty is the action a_1 player 1 decided to take in the current stage), player 1 lacks detailed information about the course of the game (i.e., he is uncertain not only about the action a_2 for the current stage but also about the current state of the game).

The choice of actions determines the outcomes for the current stage: player 1 gets an observation $o \in O$ and the game transitions to a state $s' \in S$ with transition probability $\tau(o, s' \mid s, a_1, a_2)$, where s is the current state of the game. Furthermore, player 1 gets a reward $\rho(s, a_1, a_2)$ for this transition, and player 2 receives $-\rho(s, a_1, a_2)$ (the rewards are not directly observable by player 1). Note that the next stage of the game s' is a result of joint action (a_1, a_2) and actions of player 2 (who has perfect information) directly affects the observations received by player 1 and thus his belief as well. The rewards are discounted over time with discount factor γ, $0 < \gamma < 1$.

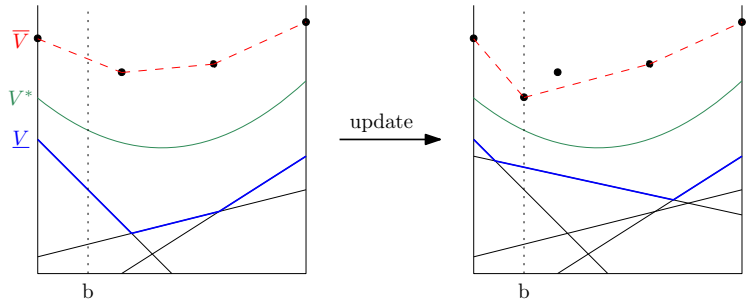

Fig. 1: HSVI local update in the belief b [11]

2.2 Heuristic Search Value Iteration (HSVI)

State of the art method for solving OS-POSGs [4] is a modification of the HSVI algorithm for Partially Observable Markov Decision Processes [10, 12] that combines heuristic search techniques with piecewise linear convex value function representations. The goal of HSVI is to approximate the optimal value function $V^* : \Delta(S) \to \mathbb{R}$ that maps each belief point to a value of the game (should the players follow optimal strategies) using a pair of value functions $\underline{V}$ (lower bound on V^*) and $\overline{V}$ (upper bound on V^*) – see Figure 1. HSVI refines these bounds by solving a sequence of stage games. In each of these stage games, the algorithm searches for the optimal strategies of both players (i.e., $\pi_1 \in \Delta(A_1)$ for player 1 and $\pi_2(s) \in \Delta(A_2)$ for player 2) while assuming that the play in the subsequent stages yields values represented by value functions $\underline{V}$ or $\overline{V}$, respectively. When moving to the next stage in sequence, the stage with maximum excess approximation error between corresponding upper and lower bound weighted by reach probability is selected. The key advantage of this approach is that in practice, we do not need to solve the whole game tree but only a smaller portion of it. Furthermore, the algorithm uses two approximations on the optimal value function V^* ($\underline{V}$ and $\overline{V}$). By further refining, these approximations are converging to the optimal value and the margin by which the approximated solution is worse than the optimal one has guaranteed bounds (unlike in other value iteration methods).[3]

The lower bound $\underline{V}$ (blue line in Figure 1) on V^* is represented by commonly used vector representation as a finite set Γ of linear functions $\alpha_i : \Delta(S) \to \mathbb{R}$. Where the value at a belief state b is the maximum projection of b onto the set Γ. The value of $\underline{V}(b)$ is point-wise maximum over this set

$$\underline{V}(b) = \max_{\alpha \in \Gamma} (\alpha \cdot b)$$

These linear functions are termed *alpha vectors* and represent expected outcomes of strategies found by the algorithm (black lines in the bottom part of Figure 1).

[3] For theoretical results and proofs refer to [11].

Since the HSVI algorithm relies on the local updates (i.e., updating bounds for specific belief point) and improving bounds in the neighborhood of the local update (as shown in Figure 1 local update results not only in improving bounds in particular belief point but also in its neighborhood), the upper bound cannot be represented by a vector set. Therefore, the upper bound $\overline{V}$ on V^* is expressed using a set $\Upsilon = \{(b^{(i)}, y^{(i)}) \,|\, 1 \leq i \leq |\Upsilon|\}$ of belief/value points $(b^{(i)}, y^{(i)})$ (dots in the upper part of Figure 1). The lower convex hull of this set of points is then used to obtain the value of $\overline{V}(b)$ (red dashed line in Figure 1).

$$\overline{V}(b) = \min_{\lambda \in \mathbb{R}^{|\Upsilon|}_{\geq 0}} \left\{ \sum_{1 \leq i \leq |\Upsilon|} \lambda_i y^{(i)} \mid \mathbf{1}^T \lambda = 1, \sum_{1 \leq i \leq |\Upsilon|} \lambda_i b^{(i)} = b \right\}$$

Finally, HSVI local updates are performed by adding a new vector (for the lower bound) or a point (for the upper bound) to the current sets Γ and Υ, respectively.

Compact representation HSVI The dimension of the value function V^* depends on the number of states, which can be potentially very large. Therefore an abstraction scheme was proposed [3]. This abstraction scheme reduces the dimensionality of the problem by creating a simplified representation of the beliefs over the state space. This means that each belief $b \in \Delta(S)$ in the game is associated with a *characteristic vector* $\chi^{(b)} = \mathbf{A} \cdot b$ (for some matrix $\mathbf{A} \in \mathbb{R}^{k \times |S|}$ where $k \ll |S|$). The characteristic vector corresponding to the initial belief b^0 is denoted as χ^0. It was proved that value function V^s computed using compact representation is valid lower bound on the solution of the original game (value function V^*) – [3, Theorem 1]. And also that the value function V^s is convex – [3, Theorem 2]. Compact representation HSVI was shown to outperform the current state of the art algorithms for solving large OS-POSGs [2, 3] in terms of scalability with only negligible loss in quality. In this work, we aim to modify this method for a different domain of games and achieve similar results.

2.3 Security Games with Sequential Attacks

Security games with sequential attacks (SGSA) [7] are an extension to the classical Stackelberg security games (SSG) model. The defender has to perpetually defend a set of targets T using a limited number of resources R. The attacker is able to surveil the defender's strategy and adjust his attack based on the surveillance. An action of the defender is deploying his limited set of resources R^s to protect targets from T^s in each game state $s \in S$. Similarly, an action of an attacker represents attacking one of the targets from T^s in each game state $s \in S$. The mixed strategy of the defender in each state $s \in S$ then corresponds to a probability distribution over pure strategies in that state. Finally, each target has associated a set of payoff values that define the utilities for both players. Since we restrict to the zero-sum case, we assume that the payoff values correspond to the perspective of the attacker receiving in case of a successful or failed attack,

respectively. SGSAs further extend this model by incorporating sequential attacks allowing an attacker to attack multiple times during one game.

The SGSA dynamic works as follows. Initially, the resources of the defender are randomly allocated to targets according to a mixed strategy of the defender in the initial stage. During the execution time, the defender samples a specific allocation of resources from his mixed strategy. The attacker is aware of defender's mixed strategy, but he lacks the information which targets are protected at the execution time. By attacking targets sequentially, the attacker is able to obtain additional information about a state of the game through observations from previous attacks. Based on the observation, the attacker can update his belief about the strategy of the defender and decide on targets to attack next that would benefit the attacker the most. After each attack, the game moves to the next stage and the defender decides whether to move security resources to any other target or not (by sampling from his mixed strategy for that particular stage of the game).

This paper focuses on solving SGSAs in the resource-movement setting under the following assumptions. First, we assume that the attacker can carry out $K > 1$ rounds of attacks and attack one target per round. Furthermore, the attacker can discover whether target t_i was protected after attacking that particular target. Note that this observation reveals only protection status for target t_i and the attacker is still unaware of the current protection status of remaining targets. The defender has to move security resources among targets in response to each attack, and there is a constant reallocation cost $c \geq 0$ for moving a resource from one target to another one[4]. Further, we assume that when a target t_i is attacked, the damage caused by the attack (if any) to t_i is already done. Therefore, that target will not be considered in future rounds. In addition, if there is a security resource protecting the attacked target, the resource has to resolve that attack. Thus the defender can no longer use that resource for future defense.

3 Using OS-POSGs for Sequential Attacks

In this section, we first represent our SGSA modeled game as OS-POSG. Then we present an HSVI-inspired algorithm for solving such games and discuss two variants of it — an exact one and a simplified heuristic formulation.

3.1 Representing SGSA as OS-POSG

Since the attacker can attack multiple times in SGSA, the game itself is divided into several rounds (stage games). Each of these stage games is equivalent to a state of the game we are trying to solve forming a set of states S. These states are described by a set of remaining security resources R, set of remaining targets

[4] Note that this can be generalized even further so that costs correspond to, for example, distances between the targets in a graph.

T, the number of remaining attacks K and initial allocation of security resources χ (based on the final allocation in the previous state). As mentioned above, in SGSA the defender has perfect information about the current situation in the game (current state of the game) and is only uncertain about the attack that will be performed. On the other hand, the attacker has only partial information since he knows only the set of remaining targets and the number of remaining resources. Therefore we can easily represent SGSA as OS-POSG. The defender from SGSA corresponds to the perfect-info player in OS-POSG (player 2 in the definition) and the attacker corresponds to the imperfect-info player (player 1 in the definition). Observation sent to the attacker contains information about whether there was a security resource on the attacked target or not. Reward function ρ returns utility of attack based on whether it succeeded or not plus the cost for reallocating security resources (if the reallocation cost $c > 0$). Finally, transition function τ determines the set of remaining resources R' and targets T' and initial allocation χ' for state s' based on taken actions. Each state $s \in S$ has its own specific value function V^s (note that this value function is equal to the value function of a subgame rooted in the state s) with corresponding upper ($\overline{V}^s$) and lower ($\underline{V}^s$) bound.

The initial allocation χ consists of a set of marginal distributions over targets—one for each resource $r \in R$—stating what is the probability that resource r is protecting target i. As the following example demonstrates, we cannot use aggregated marginal coverage ignoring the resources. In this case, the transition function τ could not uniquely define the next state of the game – the rules of SGSA require that we can identify which resource was protecting a target in case of an unsuccessful attack (that resource is removed for next stages and the allocation of other resources has to be rescaled appropriately).

Example: Let's consider instance of SGSA presented in Figure 2. This instance corresponds to stage game with 2 security resources and 4 targets and possible transitions to future stage games after target t_1 is attacked. Note that the final allocations in the root game are represented by marginal probabilities x^r per resource r. In this representation, we can easily determine the initial allocation in future stage after attacker being caught either by resource r_1 or r_2 (the initial allocation is normalized distribution consisting of probabilities $x^r[i]$ that are not crossed out). Let's assume that we will use marginal probabilities over targets ($x[i] = \sum_{r \in R} x^r[i]; \forall i \in T$) instead. In such a case, we will be still able to compute coverage of targets that will ensure the same immediate reward in first stage game as marginal probabilities per resource representation. After successful defense of a target i, security resource r protecting i is removed from the game and all contributions of r to marginal probabilities over targets must be deleted. However, when representing initial allocations by marginal probabilities over targets, we do not know the exact contributions of individual resources. Therefore, we cannot compute the exact initial allocation for sub-games after catching the attacker. To handle this issue, we have to use marginal probabilities per resource.

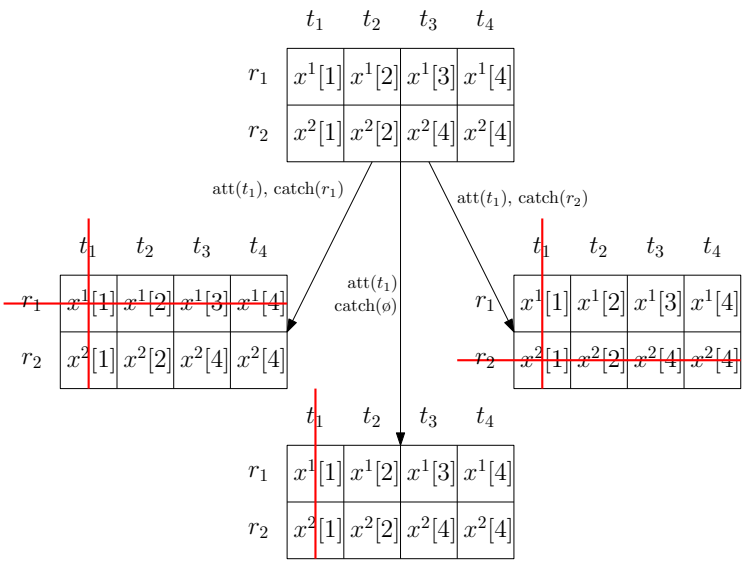

Fig. 2: Example of SGSA

3.2 HSVI-Inspired Algorithm

Our definition of SGSA uses a compact representation of states and uncertainty. Our algorithm (the pseudocode is shown in Algorithm 1) is based on the HSVI algorithm for compactly-represented lateral-movement game in computer networks [2, 3]. While the overall schema of the algorithm is similar and follows the steps of the original HSVI as described in Section 2.2, the main technical difference is in the algorithms for solving stage games and thus updating the lower and upper bound functions (described in Sections 3.3 and 3.4).

Besides this, there are two minor changes to the structure of the algorithm itself. One of the differences is that SGSA is finite horizon game (the number of rounds is limited by the number of attacks K) while the OS-POSG is infinite horizon thus we can omit discount factor γ. Another difference between Algorithm 1 and HSVI for abstracted OS-POSGs is that we do not explore only state s'_{max} with maximum weighted gap $p(s'_{max}) * (\overline{V}^{s'_{max}}(\chi') - \underline{V}^{s'_{max}}(\chi')$ but instead we explore each possible state s' for which holds $p(s') * (\overline{V}^{s'}(\chi') - \underline{V}^{s'}(\chi') > 0$. This decision is based on the experimental evaluation, where we achieved significantly better runtime when exploring all possible states with a non-zero weighted gap.

The algorithm (Algorithm 1) works as follows. First, for each state $s \in S$ we initialize bounds $\overline{V}^s$ and $\underline{V}^s$ (line 1) to valid piecewise linear and convex lower and upper bound on V^s. During the initialization phase, we initialize sets Γ and Υ for each state of the game. Every set Γ is initialized by one linear function representing the value of corresponding game state assuming that reallocation cost $c = 0$ and in the future states the defender will always catch the attacker on such a target from the remaining ones where the attacker

has the highest penalty for being caught. Similarly, every set Υ is initialized by points representing all possible pure strategies of the defender in that particular state and value achieved when playing according to that strategy assuming that all resources have to be reallocated, and in the future states, the defender will never catch the attacker while the attacker always attacks the most valuable target from the set of remaining targets. After the initialization, we perform a sequence of trials (lines 3-5) from initial characteristic vector χ^0 until the desired precision $\epsilon > 0$ (determined on line 2) is reached.

In each of the trials, we first compute the optimal *optimistic* strategy of player 2 (line 7) and update sets Γ and Υ based on the solutions of $\underline{V}^s(\chi)$ and $\overline{V}^s(\chi)$ (line 8). Next, we iterate over each pair of action a_1 of player 1 and observation o leading to next state s' with a non-zero weighted gap (lines 9-12). For each of these states we update Γ' and Υ' based on the solutions of $\underline{V}^{s'}(\chi')$ and $\overline{V}^{s'}(\chi')$ (line 13). If the gap $\overline{V}^{s'}(\chi') - \underline{V}^{s'}(\chi')$ is greater than desired ϵ, we recurse to the characteristic vector χ' (lines 14- 15). Finally, the update of sets Γ and Υ is done by adding a new alpha vector or point to the corresponding set.

```
1  Initialization
2  Set ϵ = (V̄^{s^0}(χ^0) − V̲^{s^0}(χ^0)) ∗ 10^{−2}
3  while V̄^{s^0}(χ^0) − V̲^{s^0}(χ^0) > ϵ do
4  |  Explore(s^0, χ^0, ϵ)
5  |  Update Γ and Υ based on the solutions of V̲^{s^0}(χ^0) and V̄^{s^0}(χ^0)
6  procedure Explore(s, χ, ϵ)
7  |  (b, π_2) ← optimal belief and strategy of defender in V̲^s(χ)
8  |  Update Γ and Υ based on the solutions of V̲^s(χ) and V̄^s(χ)
9  |  for (a_1, o) ∈ A_1 × O do
10 |  |  s', χ' ← τ(χ, a_1, π_2, o)
11 |  |  Determine reach probability p(s') of state s'
12 |  |  if p(s') ∗ (V̄^{s'}(, χ') − V̲^{s'}(χ') > 0 then
13 |  |  |  Update Γ' and Υ' based on the solutions of V̲^{s'}(χ') and V̄^{s'}(χ')
14 |  |  |  if V̄^{s'}(χ') − V̲^{s'}(χ') > ϵ then
15 |  |  |  |  Explore(s', χ', ϵ)
```

Algorithm 1: `HSVI` inspired algorithm for `SGSA`

As mentioned previously, one of the key differences in our `HSVI`-inspired algorithm for solving `SGSA`s compared with the original `HSVI` is in the computation of lower bound $\underline{V}$ and upper bound $\overline{V}$, which is domain dependent. We propose two variants of `HSVI` inspired algorithm for solving our `SGSA`s. The difference between them is in the way how they approach solution (i.e., estimation of $\underline{V}$ and $\overline{V}$) of stage games (i.e., the assumed set of available actions of the defender in each stage game). The first one is exact and assumes the whole action space

consisting of all possible joint actions. The second one is a simplified heuristic formulation and reduces the size of the action space by assuming that each resource has its own set of assigned targets that can be covered by that particular resource and these sets are mutually disjoint. Therefore, it is ensured that no target can be covered by more than one resource and we do not need to use joint actions and can use separate reallocation actions for each individual resource. The different action sets used by these variants result in a different construction of linear programs used for solving stage games as well. We describe these two variants in the following.

3.3 Exact Variant of the Algorithm

As we mentioned above, the marginal probabilities of covering targets are not enough and we need probability for each target being covered by particular resource. Therefore we have to consider all possible joint reallocation actions (i.e., all possible combinations of reallocating each resource from all possible starting positions to every possible end position[5]). This means that we have to deal with extremely large action space with size exponential in the number of resources R (the size of the action space is T^{2*R}). The huge action spaces result in extremely large linear programs for computing game values.

Initializing lower bound and upper bound. The presented linear program is general for an arbitrary number of resources R. For the sake of simplicity, we show the linear program for lower bound initialization of a game with $R = 2$:

$$\min V^s \tag{1a}$$

$$\text{s.t.} \sum_{i \in N} x^r[i] = 1 \qquad \forall r \in R \tag{1b}$$

$$\sum_{i,j} m[i,k,j,k] = 0 \qquad \forall k \in T \tag{1c}$$

$$\sum_{j,k,l} m[i,k,j,l] = \chi^1[i] \qquad \forall i \in T \tag{1d}$$

$$\sum_{i,k,l} m[i,k,j,l] = \chi^2[j] \qquad \forall j \in T \tag{1e}$$

$$\sum_{i,j,l} m[i,k,j,l] = x^1[k] \qquad \forall k \in T \tag{1f}$$

$$\sum_{i,j,k} m[i,k,j,l] = x^2[l] \qquad \forall l \in T \tag{1g}$$

$$x\_[i] = \sum_{j,k \in T} \sum_{l,n \in N \setminus i} m[j,l,k,n] \qquad \forall i \in T \tag{1h}$$

[5] Note that only reallocation actions resulting in situations where no target is covered by more than one resource are assumed.

$$x\_[i] * u[i] + \sum_{r \in R} x^r[i] * p[i] \leq V^s \qquad \forall i \in T \qquad (1i)$$

$$m[i,j,k,l] \geq 0 \qquad \forall i,j,k,l \in T \qquad (1j)$$

In the above linear program, the defender is looking for a new allocation of 2 security resources in stage game without considering future stages and reallocation. The probability of executing a joint reallocation action is expressed using the variable m, where the value of $m[i,k,j,l]$ corresponds to the probability of the first resource moving from target i to target k AND the second resource moving from target j to target l. We have to ensure that probabilities m of joint actions cannot exceed the initial allocation χ^r of each resource r (constraints (1d), (1e)) and sums to the final marginal probabilities x^r per resource $r \in R$ (constraints (1f), (1g)). The final marginal probabilities x^r over targets T per resource $r \in R$ must sum to 1 (constraint (1b)). We also need to ensure that one target cannot be covered by more than 1 resource at a time (constraint (1c)). Now, in order to correctly identify the initial allocation in possibly subsequent stages of the game, we need conditional probabilities in case the attacker attacks a target i that would be protected by some resource or unprotected, respectively. The probability that no resource protects target i is represented by variable $x\_[i]$. Finally, constraints (1i) represent the best response of the attacker to the defender's strategy and ensure that the defender will minimize the reward received by the attacker.

Updating lower bound and upper bound. During `HSVI` inspired computation, we need to solve linear programs for lower and upper bound and based on the solutions of these programs update set of alpha vectors Γ and set of points Υ respectively. The linear programs for computing value of lower and upper bound look almost the same as for the initialization. The only difference is that these linear programs will take into account future stages and reallocation cost.

First, we add constraints defining the reallocation costs for all actions:

$$C[k,l,m,n] = 2 * c \qquad \forall k,m \in R, \forall l \in R \setminus k, \forall n \in R \setminus m \qquad (2a)$$

$$C[k,l,m,m] = c \qquad \forall k,m \in R, \forall l \in R \setminus k \qquad (2b)$$

$$C[k,k,m,n] = c \qquad \forall k,m \in R, \forall n \in R \setminus m \qquad (2c)$$

$$C[k,k,m,n] = 0 \qquad \forall k,m \in R \qquad (2d)$$

$$\mathbb{C} = \sum_{i,j,k,l \in T} m[i,j,k,l] * C[i,j,k,l] \qquad (2e)$$

Where constraints (2a) - (2d) ensure that the reallocation cost C for each joint action corresponds to the number of resources reallocated by that joint action. Variable $\mathbb{C}$ represents a reallocation cost of defenders mixed strategy.

Second, we must add constraints for values propagating from future stage games. In the stage game with 2 resources, there are three possible next stages

reachable after an attack on target i is performed. The defender either did not catch the attacker or the attacker was caught by either resource r_1 or r_2.

We need two components to correctly compute the values of future stage games: (1) alpha vectors representing the value function of each particular sub-game and (2) initial allocation of security resources in those sub-games. Note that the future initial allocations must correspond to the final allocation in the current stage game and that the future initial allocations are already weighted by probabilities of reaching corresponding sub-games (initial allocation of individual resources in a sub-game sums to the reach probability of that sub-game). For the sub-game reachable when the attacker was not caught on target i, we will use set of alpha vectors $\mathbb{A}_-[i]$ (stands for set Γ in lower bound linear program and for the lower convex hull of set Υ in upper bound linear program) and initial allocation $b_-[i]$ which consist of initial allocation $b^1_-[i]$ of resource r_1 and initial allocation $b^2_-[i]$ of resource r_2 — constraints (3a) and (3b). When the attacker was caught on target i we will use set of alpha vector $\mathbb{A}_+[i]$ (corresponding to set Γ or lower convex hull of set Υ respectively). As initial allocation we will either use $b^1_+[i]$ (when caught by r_1) or $b^2_+[i]$ (when caught by r_2) — constraints (3e) and (3f).

$$b^1_-[att] = [\sum_{j,k\in T, l\in T\setminus att} m[j,i,k,l]; \forall i \in R \setminus att] \qquad \forall att \in T \tag{3a}$$

$$b^2_-[att] = [\sum_{j,k\in T, l\in T\setminus att} m[j,l,k,i]; \forall i \in R \setminus att] \qquad \forall att \in T \tag{3b}$$

$$b_-[i] = [b^1_-[i], b^2_-[i]] \qquad \forall i \in T \tag{3c}$$

$$\sum_{\alpha\in\mathbb{A}_-[i]} \alpha * b_-[i] \leq V^-[i] \qquad \forall i \in T \tag{3d}$$

$$b^1_+[i] = [\sum_{j,k\in T} m[j,i,k,l]; \forall l \in T \setminus i] \qquad \forall i \in T; \tag{3e}$$

$$b^2_+[i] = [\sum_{j,k\in T} m[j,l,k,i]; \forall l \in T \setminus i] \qquad \forall i \in T \tag{3f}$$

$$\sum_{\alpha\in\mathbb{A}_+[i]} \alpha * b^1_+[i] \leq V_{+,1}[i] \qquad \forall i \in T \tag{3g}$$

$$\sum_{\alpha\in\mathbb{A}_+[i]} \alpha * b^2_+[i] \leq V_{+,2}[i] \qquad \forall i \in T \tag{3h}$$

Constraint (3d) stands for expected future value if no resource is present at target i. Constraints (3g) and (3h) represent expected future values if resource r_1 or r_2 is protecting target i.

Finally, we need to modify constraints (1i) to take into account reallocation cost and values of future states

$$x_-[i] * u[i] + V_-[i] + \sum_{r\in R}(x^r[i] * p[i] + V_{+,r}[i]) + \mathbb{C} \leq V^s \tag{4}$$

3.4 Heuristic Variant of the Algorithm

To tackle the issue with large action space needed for the exact variant of our algorithm, we devised a simplified heuristic formulation of stage games we need to solve. The heuristic formulation assumes that each resource has its own set of assigned targets that can be covered by that particular resource and these sets are mutually disjoint. Such distribution ensures that every target can be covered by only one resource and therefore we can have separate reallocation actions for each resource r. This means that the size of the action space is significantly reduced since it is no longer exponential but linear.

Initializing lower bound and upper bound. The smaller number of actions in the game results in less variables in the linear program and easier construction of the linear program as well. In general, the linear program for initialization of lower bound looks as follows:

$$\min V^s \tag{5a}$$

$$\text{s.t.} \sum_{i,j \in T} m^r[i,j] = 1 \qquad \forall r \in R \tag{5b}$$

$$\sum_{j \in T} m^r[j,i] = x^r[i] \qquad \forall r \in R, \forall i \in T \tag{5c}$$

$$\sum_{j \in T_r} m^r[i,j] = \chi^r[i] \qquad \forall i \in T \tag{5d}$$

$$\sum_{j \in T_r \setminus i} x^r[j] * u[i] + x^r[i] * p[i] \leq V^s \qquad \forall i \in T, r \in R; T_r \ni i \tag{5e}$$

$$m^r[i,j] \geq 0 \qquad \forall i,j \in T \tag{5f}$$

Where $m^r[i,j]$ stands for the probability of executing a reallocation action of resource r from target i to target j. As in the linear program in the exact variant of algorithm, we have to ensure that the probabilities of reallocation actions m^r of resource r sums to 1 (constraints (5b)) and do not exceed the initial allocation χ^r (constraints (5d)) and sums to final marginal probabilities x^r per resource r (constraints (5c)). Finally, we represent best response of the attacker by the constraints (5e).

Updating lower bound and upper bound. The modifications needed to obtain linear programs for computing lower and upper bound are similar to the ones used for the exact variant. Since actions in the heuristic variant correspond to the reallocation of only one resource (unlike the joint actions in the exact variant that correspond to the reallocation of multiple resources), we do not need to specifically define reallocation costs for actions. Thus first step is to add constraints for values of future states, which can be represented as follows:

$$b^r_-[i] = [\sum_{k \in T} m^r[k,j] - \lambda^r_-[i,j]; \forall j \in T_r] \qquad \forall i \in T, \forall r \in R \qquad (6a)$$

$$\sum_{j \in T} \lambda^r_-[i,j] = \sum_{r' \in R} x^r[i] \qquad \forall i \in T, \forall r \in R \qquad (6b)$$

$$b_-[i] = [b^r_-[i]; \forall r \in R] \qquad \forall i \in T \qquad (6c)$$

$$\sum_{\alpha \in \mathbb{A}_-[i]} \alpha * b_-[i] \leq V^-[i] \qquad \forall i \in T \qquad (6d)$$

$$b^r_+[i] = [\sum_{k \in T} m^r[k,j] - \lambda^r_+[i,j]; \forall j \in T_r] \qquad \forall i \in T, \forall r \in R \qquad (6e)$$

$$\sum_{j \in T} \lambda^r_+[i,j] = \sum_{j \in T_r \setminus i} x^r[j] \qquad \forall i \in T, \forall r \in R \qquad (6f)$$

$$b_+[i] = [b^r_+[i]; \forall r \in R \wedge i \notin T_r] \qquad \forall i \in T \qquad (6g)$$

$$\sum_{\alpha \in \mathbb{A}_+[i]} \alpha * b_+[i] \leq V_{+,r}[i] \qquad \forall i \in T, r \in R; T_r \ni i \qquad (6h)$$

$$\lambda^r_-[i,j] >= 0 \qquad \forall r \in R, \forall i,j \in T \qquad (6i)$$

$$\lambda^r_+[i,j] >= 0 \qquad \forall r \in R, \forall i,j \in T \qquad (6j)$$

Since we are not using joint actions anymore, the initial allocation for reached sub-game conditioned by final allocation in the current game can be easily obtained. However, it will not be automatically weighted by reach probability (initial allocation of individual resources will not sum to the reach probability of that stage game) like in the exact variant. To achieve that we allow the defender to modify the initial allocation of the followup stage game. Therefore we introduce slack variables λ_- and λ_+ that are used by the defender to decrease initial allocations of individual resources and make it sum to reach probability of that stage game. Value of slack variable $\lambda^r_-[i,j]$ represents how much the defender reduced initial allocation of resource r on target j if target i was attacked and the attacker was not caught. Similarly, the value of slack variable $\lambda^r_+[i,j]$ represent how much the defender reduced initial allocation of resource r on target j if target i was attacked and the attacker was caught. Constraints (6a) (equivalent to constraints (3a) and (3b)) and (6e) (equivalent to constraints (3e) and (3f)) select the initial allocations of individual resources for sub-games while constraints (6b) and (6f) ensure that selected initial allocations will remain non-negative. Constraints (6d) and (6h) represent expected future values after successful attack and after attacker being caught, respectively (equivalent to constraints (3d) and constraints (3g) and (3h), respectively).

Finally, just like in the case of exact variant, we need to modify (5e) in similar way as (1i), resulting in the following constraint:

$$\sum_{j \in T_r \setminus i} x^r[j] * u[i] + V_-[i] + x^r[i] * p[i] + V_{+,r}[i] + \sum_{l \in R, m,n \in T} m^l[m,n] * c \leq V^s \quad (7)$$

4 Experimental Evaluation

In this section we present experimental evaluation of proposed variants of our algorithm introduced in Sections 3.3 and 3.4. We compare these variants based on their runtime and solution quality.

4.1 Experiments Setting

The evaluation has been performed on sets of randomly generated games with varying parameters – the number of targets T, number of resources R and number of attacks K. Each of these games has randomly generated rewards of the attacker for successful attacks on targets (uniformly taken from interval $[0, 6]$), attacker's penalties from being caught on individual targets (uniformly taken from interval $[-6, 0]$), reallocation cost (uniformly taken from interval $[0, 1]$) and initial allocation (i.e., χ^0). In the heuristic variant of our algorithm, targets were uniformly distributed to individual resources in descending order of attacker's utility for a successful attack.

All computational results have been obtained on computers equipped with *Intel Xeon Scalable Gold 6146* processors and *32GB* of available RAM while limiting the runtime to 2 hours. We used CPLEX 12.9 to solve linear programs. The solution approaches were required to find an ϵ-optimal solution where ϵ is set to 1% of the error $(\overline{V}^{s^0}(\chi^0) - \underline{V}^{s^0}(\chi^0))$ after the initialization phase described in Section 3.2 is completed. If the algorithm failed to reach this level of precision within 2 hours, we report such instance as unsolved. The results are based on 50 randomly generated games for each parameter set.

In order to compare the quality of computed defense strategies across multiple instances of generated games, we (1) evaluate the exploitability of the strategies of the defender by computing a best response for the attacker (since we are restricted to zero-sum games) and we (2) normalize the differences between the expected outcomes against the best-responding attacker to obtain comparable relative differences across various instances of generated games. Similarly to setting the target error ϵ, we use the initial size of the interval between the upper and the lower bound for the initial belief as the normalization factor.

4.2 Comparison with State of the Art

To the best of our knowledge, right now there is no clear state of the art solution approach to compare with. Comparing to the methods proposed in [7] is not possible due to the different assumed setting. We focus on solving zero-sum `SGSA`s with reallocations costs without limiting the number of attacks. On the other hand, previous work focused on solution of general-sum `SGSA`s without reallocation cost with limiting the number of attacks $K = 2$ [7].

The solution approach closest to the state of the art is solving the game as separate `SSG`s. This method scales much better than other proposed methods; however, it significantly falls behind in the quality of the solution. As we can

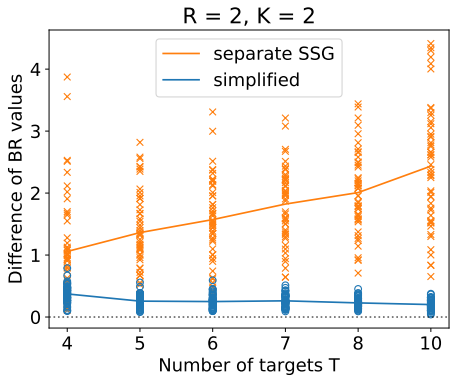

Fig. 3: Difference of best response values produced by heuristic algorithm and separate SSG approach compared to non-heuristic algorithm.

observe in Figure 3, solution quality of separate `SSG`s approach (compared to the solution found by the exact variant of our algorithm) becomes significantly worse with an increased size of the game. This is due to the fact that each stage game is solved separately without taking into account future stages. This means that we can solve each stage optimally in the sense of separate games. However, since solution in previous time step directly affects solution in the current one, these solutions are not optimal from the global point of view (e.g., the defender cannot control reach probabilities of individual stage games).

On the other hand, for the simplified version of our algorithm holds the opposite, the difference in solution quality (compared to the solution found by the exact variant of our algorithm) decreases with an increased size of the game. Therefore we focus solely on analysis of `HSVI` solution approach.

4.3 Algorithm Scalability

First we focused on the scalability of proposed variants in the size of the game - number of targets T, number of resources R and number of attacks K (Figure 4). In Figure 4, we use two y-axes. The left y-axis represents the runtime (seconds) in a logarithm scale (upper part of the figure). The right y-axis presents the percentage of unsolved games (bottom part of the figure).

Figure 4a depicts the scalability in the number of targets T. We can observe that with a fixed small number of resources R and attacks K both variants scale quite well up to the $T = 10$ solving nearly 100% of instances for each game size. With further increasing number of targets, the percentage of unsolved instances becomes higher, especially for the exact variant.

In Figure 4b, we present the scalability in the number of resources R. The exact approach was able to solve only the smallest game instances with $R = 2$. On the other hand, the heuristic variant was capable of finishing all computations within 2 hours and achieved reasonable runtime across all sizes of game instances.

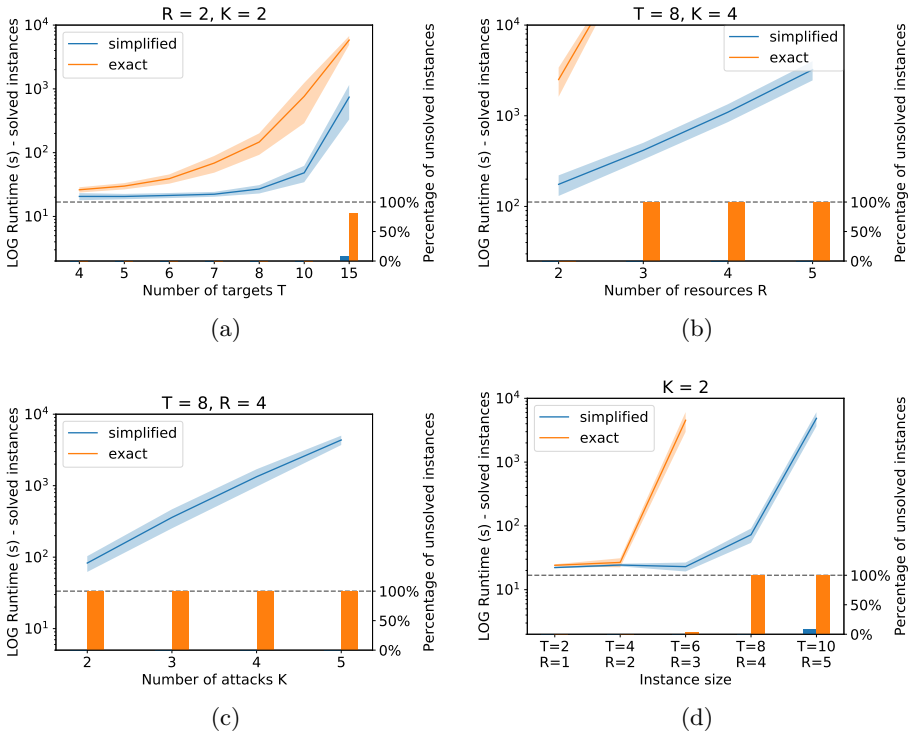

Fig. 4: Scalability in different parameters affecting the size of the game (a) number of targets T, (b) number of resources R, (c) number of attacks K and (d) number of targets T and resources R with a fixed number of attacks K. Averages based on 50 instances for each parameter set. Confidence intervals mark the standard error. The reported runtimes include only instances solved by the algorithm variants. The percentage of instances where the algorithm variants failed to terminate within 2 hours are reported separately.

Figure 4c shows the scalability in number of attacks K. Again, we can observe that the exact variant struggles when it comes to solving larger games resulting this time in no instances solved. The heuristic approach keeps its performance and solves all instances in the given time limit.

Finally, in Figure 4d, we present scalability for fixed number of attacks $K = 2$ and increasing number of targets T and resources R with fixed ratio $T : R$. These results support what we were able to observe in all previous scalability experiments. The exact variant can easily solve smaller games with runtimes not very different from those achieved by the simplified one. However, with the increasing game size, the solution speed rapidly degrades. This behaviour is closely connected to the number of actions considered by these variants. The exact one has to use joint-actions which results in T^{2*R} actions in the problem. On the

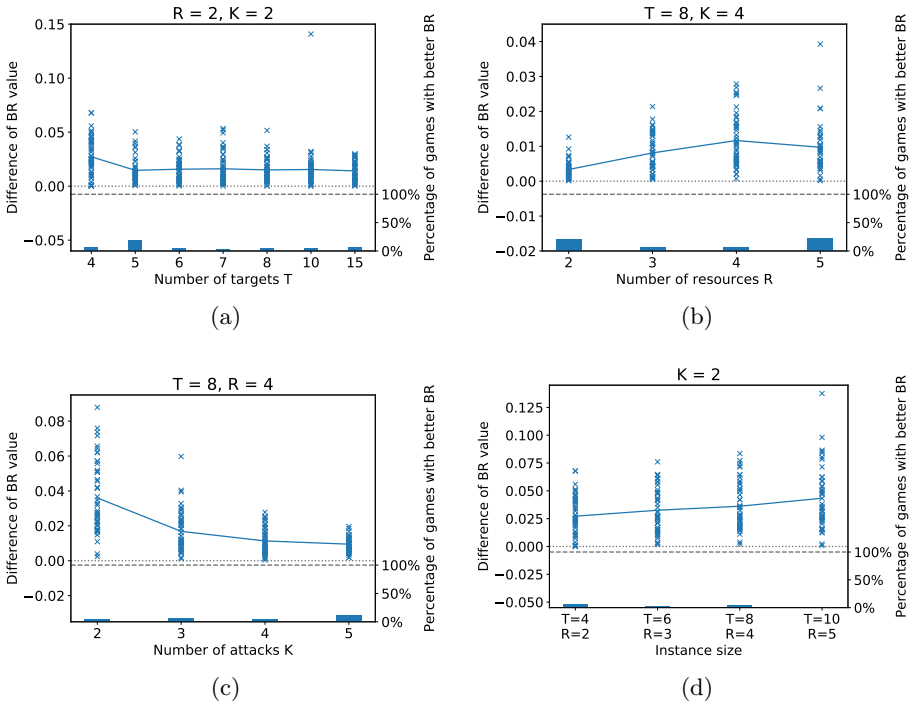

Fig. 5: Difference between computed upper bound value and best response value of the corresponding strategy depending on different parameters affecting size of the game (a) number of targets T, (b) number of resources R, (c) number of attacks K and (d) number of targets T and resources R with a fixed number of attacks K. Averages based on 50 instances for each parameter set. The reported differences include only instances where best response value was worse than upper bound value. The percentage of instances where the best response value was better than upper bound value is reported separately.

other hand, the simplified heuristic variant assumes that targets are exclusively assigned to individual resources for cover (i.e. each resource has assigned a list of targets that can be covered by this resource and these sets are mutually disjoint). Thus the heuristic approach can work with separate reallocation actions for each resource, resulting in $T^2 * R$ actions. The number of actions directly affects the size of a linear program and therefore the memory and time requirements for solving it.

4.4 Solution Quality

In this section, we focus on the solution quality of our proposed approaches. First, we compare the exact version of our algorithm with its heuristic variant.

We observe that the solution quality of the simplified heuristic approach highly depends on the size of the game, and the bigger the game is the closer the heuristic best response gets to the exact one (Figure 3). The worse values achieved by the heuristic approach are due to the exclusive target assignment used in the heuristic.

The following example demonstrates the key limitation of the heuristic approach and the reason for not good quality of the strategies found by the heuristic variant in games with small number of targets (see the difference for $T = 4$ in Figure 3 that is 0.373 on average). Without loss of generality, let's assume we want to solve a game with $T = 3$, $R = 2$, attacker's rewards for attacking non-covered target $u = [3, 3, 3]$ and no penalties for the attacker when being caught or reallocation cost. In the exact variant, we are able to achieve the game value of 1 since it is possible to cover all three targets with uniform probability $\frac{2}{3}$. On the other hand, in the heuristic approach two targets will be assigned to one resource and one target to the other resource. Because of this, we are no longer able to achieve the same coverage as in the exact approach and the best we can do is to cover that single-assigned target with probability 1 and the remaining two targets (those assigned to the same resource) with uniform probability $\frac{1}{2}$ resulting in the game value of 1.5. The actual difference in expected outcomes between the optimal strategy and the heuristic strategy can be even higher if the rewards of the attacker for a successful attack are higher or if the reallocation costs are considered. However, with the increasing number of targets T (or the number of resources R), the impact of this limitation decreases (to 0.201 for $T = 10$).

Since we cannot compute optimal strategies using the exact variant for larger instances, we evaluate the robustness of strategies computed by the heuristic variant of our algorithm as the difference between the computed upper bound value and the best response value of the corresponding strategy (normalized by initial gaps – Figure 5). In this figure, we use two y-axes. The left one represents the difference between the upper bound and best response values (upper part of the figure) and the right one presents the percentage of games in which the best response value was strictly better than the computed upper bound value (bottom part of the figure). As Figure 5 shows, the heuristic algorithm was capable of retaining its properties across all instances and keep the average exploitability of computed strategies below 5%.

5 Conclusion

In this work, we study the problem of sequential attacks in security games. We introduce a new formulation of zero-sum security games that consider such sequential attacks, and we use the formalism of one-sided partially observable stochastic games. This allows us to use existing algorithms developed for this class of games. We exploit compact representation of uncertainty and design a heuristic variant of the problem that, for larger games, achieves very similar quality of strategies compared to the exact formulation, while scaling to greater

depths and the number of resources. Our paper opens a new possible direction for studying security games with sequential attacks. Key components of the algorithm can be improved to achieve even better scalability. The second important direction is a modification of the algorithm to support also general-sum security games and computation of Strong Stackelberg equilibria.

References

1. Fang, F., Nguyen, T.H., Pickles, R., Lam, W.Y., Clements, G.R., An, B., Singh, A., Tambe, M., Lemieux, A.: Deploying PAWS: Field optimization of the protection assistant for wildlife security. In: IAAI (2016)
2. Horák, K., Bošanský, B., Kiekintveld, C., Kamhoua, C.: Compact representation of value function in partially observable stochastic games. IJCAI (2019)
3. Horák, K., Bošanský, B., Tomášek, P., Kiekintveld, C., Kamhoua, C.: Optimizing honeypot strategies against dynamic lateral movement using partially observable stochastic games. Computers & Security **87**, 101579 (07 2019). https://doi.org/10.1016/j.cose.2019.101579
4. Horák, K., Bošanský, B., Pěchouček, M.: Heuristic Search Value Iteration for One-Sided Partially Observable Stochastic Games. In: 31st AAAI Conference on Artificial Intelligence. pp. 558–564 (2017)
5. Kiekintveld, C., Jain, M., Tsai, J., Pita, J., Ordóñez, F., Tambe, M.: Computing optimal randomized resource allocations for massive security games. In: Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems. pp. 689–696 (2009), `http://portal.acm.org/citation.cfm?id=1558013.1558108`
6. Nguyen, T.H., Sinha, A., Gholami, S., Plumptre, A., Joppa, L., Tambe, M., Driciru, M., Wanyama, F., Rwetsiba, A., Critchlow, R., Beale, C.M.: CAPTURE: A New Predictive Anti-Poaching Tool for Wildlife Protection. In: Proceedings of the 2016 International Conference on Autonomous Agents and Multiagent Systems. pp. 767–775. AAMAS, Richland, SC (2016), `http://dl.acm.org/citation.cfm?id=2937029.2937037`
7. Nguyen, T.H., Yadav, A., Bošanský, B., Liang, Y.: Tackling sequential attacks in security games. In: Alpcan, T., Vorobeychik, Y., Baras, J.S., Dán, G. (eds.) Decision and Game Theory for Security. pp. 331–351. Springer International Publishing, Cham (2019)
8. Shieh, E., An, B., Yang, R., Tambe, M., Baldwin, C., DiRenzo, J., Maule, B., Meyer, G.: PROTECT: A deployed game theoretic system to protect the ports of the United States. In: AAMAS (2012)
9. Sinha, A., Fang, F., An, B., Kiekintveld, C., Tambe, M.: Stackelberg security games: Looking beyond a decade of success. In: IJCAI. pp. 5494–5501 (2018)
10. Smith, T., Simmons, R.: Heuristic search value iteration for POMDPs. In: 20th Conference on Uncertainty in Artificial Intelligence (UAI). pp. 520–527 (2004)
11. Smith, T., Simmons, R.: Heuristic search value iteration for pomdps: Detailed theory and results. Technical report, Robotics Institute, Carnegie Mellon University (2004)
12. Smith, T., Simmons, R.: Point-based POMDP algorithms: improved analysis and implementation. In: 21st Conference on Uncertainty in Artificial Intelligence (UAI). pp. 542–549 (2005)
13. Tambe, M. (ed.): Security and Game Theory: Algorithms, Deployed Systems, Lessons Learned. Cambridge University Press (2011)