Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI-16)

A Security Game Combining Patrolling and Alarm–Triggered Responses Under Spatial and Detection Uncertainties

Nicola Basilico
University of Milan
Via Comelico, 39/41
Milano, Italy
nicola.basilico@unimi.it

Giuseppe De Nittis and **Nicola Gatti**
Politecnico di Milano
Piazza Leonardo da Vinci, 32
Milano, Italy
{giuseppe.denittis, nicola.gatti}@polimi.it

Abstract

Motivated by a number of security applications, among which *border patrolling*, we study, to the best of our knowledge, the first *Security Game* model in which patrolling strategies need to be combined with responses to signals raised by an alarm system, which is *spatially uncertain* (i.e., it is uncertain over the exact location the attack is ongoing) and is affected by *false negatives* (i.e., the missed detection rate of an attack may be positive). Ours is an infinite–horizon patrolling scenario on a graph where a single patroller moves. We study the properties of the game model in terms of computational issues and form of the optimal strategies and we provide an approach to solve it. Finally, we provide an experimental analysis of our techniques.

Introduction

The paradigm of Security Games, modeling the problem of protecting physical environments as a non–cooperative game between a *Defender* and an *Attacker*, is one of the successful applications of game theory in the real world (Jain, An, and Tambe 2012). Such paradigm aims at maximizing the protection of targets by scheduling the Defender's resources (like canine units or guards) during time. A long–standing literature focuses on a number of issues involved in Security Games such as studying different observation models for the Attacker (An et al. 2013; Yang et al. 2014) and incorporating realistic aspects of infrastructures to be protected (Blum, Haghtalab, and Procaccia 2014).

Besides scheduling mobile resources (a.k.a. *patrolling*), real security settings can adopt *sensors*, capable to trigger *alarms* when attacks are detected. This approach is of high prominence in practice, since sensors can be cheap devices that are deployed over the environment with high capillarity, allowing remarkable improvements of the protecting task. Indeed, it can be showed that patrolling strategies exploiting an alarm system may be arbitrarily better than strategies that do not and this pushed for considering this feature in Security Games. With an alarm system, two questions must be answered simultaneously: *which is the best patrolling strategy in absence of any alarm signal?* and, *which is the best strategy to respond to an attack once an alarm signal has been raised?* (Munoz de Cote et al. 2013) is the first attempt to tackle this problem. Each sensor is affected by *detection uncertainty* (i.e., *false negatives* and *false positives*) and covers exactly one single target. The best strategy to respond to an alarm signal is easily computable, reducing to the rush to a target once its sensor detected an attack. On the other hand, finding the best patrolling strategy is hard and the proposed techniques to approximate the best patrolling strategy are strongly intractable even with few targets (e.g., even with 5 targets). (Basilico and Gatti 2014) studies sensors without detection uncertainty, but the single–target deployment assumption is relaxed allowing sensors to be *spatially imperfect*. This means that an alarm signal is raised if and only if an attack is performed, but the Defender is uncertain on the actual attacked location, as in border patrolling (Agmon, Kraus, and Kaminka 2008). In this case, the best patrolling strategy is easily tractable, being *placement based*, i.e., place a guard in a location, wait for an alarm signal, and then respond to it at best. Instead, finding the best response to an alarm signal is computationally hard. However, the proposed techniques scale with a high number of targets (i.e., beyond 50). In the present paper, we bridge together the above two approaches towards a more general framework and propose the first security game that combines sensors detection uncertainty together with spatial imperfection, tackling the challenge of designing tractable algorithms for real–life scenarios.

Original contributions We formalize our game model, whose resolution's complexity is $\mathcal{APX}$–hard in the case without false negatives (Basilico, De Nittis, and Gatti 2015) and $\mathcal{PSPACE}$–hard in the case without alarm systems (Ho and Ouaknine 2015), and we prove that placement–based strategies may be arbitrarily worse than *patrolling–based* ones. This motivates the need for strategies in which the Defender both patrols over some targets and responds at best once an alarm signal is raised, thus requiring innovative and more involved techniques w.r.t. the case without false negatives. We formulate the problem of searching for the best Defender's strategy as a mixed–integer–non–linear mathematical program whose resolution can be done in practice only by means of local–search techniques based on non–linear mathematical programming. Furthermore, if non–Markovian patrolling strategies are used, the size of the mathematical program explodes, while if Markovian patrolling strategies are used, the quality of the solution can be very low (we prove indeed that Markovian strategies may be arbitrarily worse than non–Markovian strategies). Motivated by the hardness of the problem, we study its properties in the attempt to design algorithms as efficient as it is possible. Based on our study, we propose an *anytime* approach based on two *oracles* in which we consider only deterministic (potentially non–Markovian) patrolling strategies for the case in which no alarm signal is present. We design an algorithm searching for the optimal subset of targets over which the patroller moves and, for each subset, we use an oracle to verify whether there exists a deterministic patrolling strategy. If such strategy exists, we use an oracle to determine the best response to the alarm signals. We propose three *heuristics* to guide the search of our algorithm. Two heuristics are *static*, enumerating the subsets of targets at time zero and keeping such an order during all the search, and one heuristic is *dynamic*, changing the order on the basis of the information got by the evaluation of the previous subsets. Finally, we provide an experimental evaluation to assess the quality and the scalability of our algorithms.

Problem formulation

We study a patrolling scenario modeled as a turn–based extensive–form game with infinite horizon and imperfect information between two agents: an *Attacker* $\mathcal{A}$ and a *Defender* $\mathcal{D}$. The environment is formally described by an undirected connected graph $G = (V, E)$. Each edge $(i, j) \in E$ requires one turn to be traversed, while we denote with $\omega^*_{i,j}$ the temporal cost (in turns) of the shortest path between any i and $j \in V$. We call targets the subset $T \subseteq V$ that are relevant to $\mathcal{A}$ and $\mathcal{D}$. Each target $t \in T$ has a value $\pi(t) \in (0, 1]$ and a penetration time $d(t) \in \mathbb{N}$ measuring the number of turns needed to successfully complete an attack over t. Finally, a *spatially uncertain alarm system* is available to $\mathcal{D}$, modeled as a pair (S, p) where $S = \{s_0, s_1, \cdots, s_m\}$ with $m \geq 1$ is a set of *signals* and $p : S \times T \to [0, 1]$ specifies the probability of having the system generating a signal s given that target t has been attacked. With s_0 we denote the null signal corresponding to the case in which no signal has been generated due to a false negative. For simplicity, we assume that $p(s_0 \mid t) = \alpha$ for every t.

At each turn of the game, agents $\mathcal{A}$ and $\mathcal{D}$ play simultaneously: if $\mathcal{A}$ has not attacked in the previous turns, it observes the position of $\mathcal{D}$ in the graph and decides whether to attack a target[1] or to wait for a turn, while $\mathcal{D}$ observes whether or not a signal has been generated by the alarm system and decides the next vertex to patrol among all those adjacent to the current one. If $\mathcal{D}$ patrols a target t that is under attack of $\mathcal{A}$ before $d(t)$ turns, $\mathcal{A}$ is captured. The game is constant sum (then equivalent to a zero–sum game): if $\mathcal{A}$ is captured, $\mathcal{D}$ receives a utility of 1 and $\mathcal{A}$ receives 0, while, if an attack over t has success, $\mathcal{D}$ receives $1 - \pi(t)$ and $\mathcal{A}$ receives $\pi(t)$; finally, if $\mathcal{A}$ waits forever, $\mathcal{D}$ receives 1 and $\mathcal{A}$ receives 0. The appropriate solution concept is the leader–follower equilibrium. The game being constant sum, the best leader's strategy is its maxmin/minmax strategy.

We denote by $\sigma^{\mathcal{A}}$ and $\sigma^{\mathcal{D}}$ the strategies of $\mathcal{A}$ and $\mathcal{D}$, respectively. For the sake of simplicity, we denote by $\sigma^{\mathcal{D}}_a$ the strategy of $\mathcal{D}$ once an alarm signal s_i with $i \geq 1$ has been raised (the response component) and by $\sigma^{\mathcal{D}}_p$ the strategy of $\mathcal{D}$ in absence of any alarm signal (the patrolling component).

Problem analysis

We start by providing a theoretical analysis that will motivate our algorithmic approach to the problem. While with $\alpha = 0$ the best patrolling strategy prescribes that the patroller places in a vertex waiting for an alarm signal and then responds at best to it (Basilico, De Nittis, and Gatti 2015), this is no longer true even when α is small, as stated by the following proposition.

Proposition 1 *There exist instances with $\alpha > 0$ in which the best placement–based strategy is not optimal.*

The weakness of placement–based strategies is even stronger, as stated in the following proposition.

Proposition 2 *There exist instances with $\alpha > 0$ in which the best placement–based strategy is arbitrarily worse w.r.t. the best patrolling–based strategy.*

The above results show that placement–based strategies do not suffice and patrolling–based strategies are necessary.

In terms of computational complexity, we already know that the problem of finding the $\mathcal{D}$'s maxmin strategy is $\mathcal{APX}$–hard from the analysis of the case in which $\alpha = 0$, now we can prove that the problem is hard also in terms of spatial complexity, as stated in the following.

Theorem 1 *The problem of deciding whether there is a $\mathcal{D}$'s strategy giving a utility of at least k is $\mathcal{PSPACE}$–hard.*

The problem of finding the maxmin strategy can be formulated as follows. We define $\sigma^{\mathcal{D}}_p$ as a function $\sigma^{\mathcal{D}}_p : H \times V \to [0, 1]$, retuning the probability to move at the next turn to an adjacent vertex $v \in V$ given a history of visits over vertices $h \in H$, where H is the set of (potentially infinite–length) histories. We define a *route* $r \in R$ as a sequence of potentially non–adjacent vertices and targets such that $r(0) \in V$ and $r(i) \in T$ with $i > 0$. We say that a route r is *covering* if the first arrival in a target $t \in r$ starting from $r(0)$ is accomplished not later than $d(t)$. That is, if $\mathcal{D}$ starts from t and follows r, then it can cover in time all the targets $t \in r$. We define $\sigma^{\mathcal{D}}_a$ as a function $\sigma^{\mathcal{D}}_a : S \times R \to [0, 1]$, returning the probability to move along route $r \in R$ starting from $r(0)$ once alarm signal $s \in S$ has been raised. The actions available to $\mathcal{A}$ are attack–when(t, h), meaning that $\mathcal{A}$ attacks target t after observing history h of $\mathcal{D}$. For the sake of simplicity, we report the mathematical programming formulation when histories h are composed of only one vertex, corresponding to the last visited vertex by $\mathcal{D}$ (hence, the strategy is Markovian).

[1] As is customary, we assume that $\mathcal{A}$ can instantly reach the target of its attack. This assumption can be easily relaxed as shown in (Basilico, Gatti, and Rossi 2009).

$$\begin{aligned}
\min \quad & u && (1)\\
& u \geq \mathbb{E}[U(\text{attack–when}(t,v))] - 1 + I_v \qquad \forall t \in T, v \in V && (2)\\
& I_v \geq P_{\text{steady}}(v, \sigma_p^{\mathcal{D}}) \qquad \forall v \in V && (3)\\
& I_v \in \{0,1\} \qquad \forall v \in V && (4)\\
& \sigma_a^{\mathcal{D}} \text{ is well defined} && (5)\\
& \sigma_p^{\mathcal{D}} \text{ is well defined} && (6)
\end{aligned}$$

where $P_{\text{steady}}(v, \sigma_p^{\mathcal{D}})$ is the steady state probability that $\mathcal{D}$ is at v given strategy $\sigma_p^{\mathcal{D}}$ and

$$\mathbb{E}[U(\text{attack–when}(t,v))] = \Bigg[\alpha\pi(t)\Big(1 - P_{\text{capture}}(t, v, \sigma_p^{\mathcal{D}})\Big) + \pi(t) \sum_{s \in S \setminus \{s_0\}} p(s \mid t)\Big(1 - \sum_{r: r(0)=v, t \in r} \sigma_a^{\mathcal{D}}(r, s)\Big)\Bigg] \qquad (7)$$

with $P_{\text{capture}}(t, v, \sigma_p^{\mathcal{D}})$ the probability that t is visited by $d(t)$ turns starting from v given $\sigma_p^{\mathcal{D}}$. Notice that I_v is equal to 0 only if $P_{\text{steady}}(v, \sigma_p^{\mathcal{D}}) = 0$ and it is necessary in the formulation to enable/disable Constraints (2)—the rationale is that, if a vertex $\overline{v}$ is never visited, all the actions attack–when$(t, \overline{v})$ must be disabled. The above mathematical program is mixed integer (due to I_v) non–linear non–convex (due to $P_{\text{steady}}(v, \sigma_p^{\mathcal{D}})$ and $P_{\text{capture}}(t, v, \sigma_p^{\mathcal{D}})$). When using histories of length longer than 1, the program has the above structure except that the number of variables and constraints rises exponentially in the length.

Many issues are related to Problem (1)–(6). Specifically, the size explodes as the length of the histories is larger than 1, making *de facto* only Markovian strategies usable. However, Markovian strategies may be arbitrarily worse than non–Markovian ones, as stated in the following.

Proposition 3 *Markovian patrolling strategies are arbitrarily worse than the optimal non–Markovian ones.*

Furthermore, Problem (1)–(6) can be solved exactly only by means of global optimization tools, e.g. (Cook et al. 2011). However, these tools can deal only with extremely small programs (e.g., with about 20/30 variables) and therefore they are inapplicable even in toy instances. When non–linear mathematical programming tools (Bertsekas 1999) are adopted, two issues arise: integer variables cannot be effectively handled and only locally optimal solutions can be found. Finally, differently from the case without false negatives, Problem (1)–(6) cannot be separated into two independent programs, one searching for the best $\sigma_a^{\mathcal{D}}$ and another for the best $\sigma_p^{\mathcal{D}}$, and thus, from its resolution, we would obtain sub–optimal strategies both for responding to alarm signals and for patrolling in absence of raised signals.

Our approach

Tackling the resolution of Problem (1)–(6) in its original formulation would be, as discussed above, prohibitive in terms of computational effort and would require to limit the structure of the strategies by fixing a history length, incurring in potentially large losses of utility. Our objective is to address the resolution of this problem in an anytime fashion, providing an algorithmic method whose convergence to an optimal solution progresses with time. We start by introducing the notion of support graph.

Definition 1 (Support graph of a patrolling strategy) *The support graph of a patrolling strategy $\sigma_p^{\mathcal{D}}$ is a pair $G^{\sigma} = (V^{\sigma_p^{\mathcal{D}}}, E^{\sigma_p^{\mathcal{D}}})$, where $V^{\sigma_p^{\mathcal{D}}} = \{v \in V \mid P_{steady}(v, \sigma_p^{\mathcal{D}}) > 0\}$ is called support, and $E^{\sigma_p^{\mathcal{D}}} = \{(i,j) \in E \mid \exists h_i \; \sigma_p^{\mathcal{D}}(h_i, j) > 0\}$ in which h_i is a history of visits whose last vertex is i.*

In general, the support of the optimal patrolling strategy may be a strict subset of V. This is primarily due to the alarm system, whose presence enables the protection of non–patrolled targets via signal responses. We can leverage this in building a simple enumeration Multi–NLP scheme where each sub–problem is obtained from (1)–(6) by fixing I_v under a given support (this removes the integer variables from the optimization problem). Despite this would obviously require to enumerate $O(2^{|V|})$ subproblems, working in the space of supports introduces advantages. Indeed, by studying the properties of support graphs we can speedup enumeration by pruning those supports we *a priori* know not to be optimal. We start by showing that not all the supports must be enumerated in order to guarantee optimality.

Theorem 2 *The support of the optimal patrolling strategy will either be a singleton or will contain at least two targets.*

From Theorem 2 we know that we can save the time needed by enumerating all the supports with less than two targets by enumerating the $O(|V|)$ singleton ones. Let us focus our attention to the other supports to be enumerated, namely those containing at least two targets.

Consider a subset of targets T' with $|T'| \geq 2$. We drive our attention to support $V' \supseteq T'$, defined as the one among all supports covering only targets T' that provides $\mathcal{D}$ with the lowest (best) expected utility as per Problem (1)–(6). Our aim is to derive some information about the non–target vertices composing V'. The following result shows how optimality induces a structural property, which can be exploited to derive some insights on supports with two or more targets.

Theorem 3 *The support graph of the optimal patrolling strategy does not contain non–target terminal vertices (vertices with degree 1).*

The previous results convey the idea that if the optimal support does not encode a static placement, then it will be composed by two or more targets (Theorem 2) and, for each pair of them, by a number of (not necessarily shortest) acyclic paths of non–target vertices connecting them (Theorem 3). The key advantage that such properties enable is that they ease the task of guessing the vertices contained in the support. While guessing which vertices will be contained in the optimal support can be a very difficult task, guessing the non–target vertices connecting a given subset of targets in the support is relatively feasible by means of heuristics. So, if $|T|$ is the number of targets, our approach is to perform a $O(|V|-|T|-1+2^{|T|}) \approx O(2^{|T|})$ search, substantially enumerating subsets of targets instead of subset of vertices in $O(2^{|V|})$ (recall that, by definition, $|V| \geq |T|$). For each subset of targets, our approach is to guess the remaining vertices composing the support by querying an oracle that searches for a covering cycle on them.

Definition 2 ***(Covering cycle for a subset of targets)*** *Given a subset of targets $T' \subseteq T$, a covering cycle on them is the shortest finite cyclic walk on graph G such that when repeatedly following it, two subsequent visits to any $t \in T'$ have a temporal delay not greater than $d(t)$.*

Given a subset of targets, if a covering cycle on them is found, then we adopt such cycle as patrolling strategy and the covered vertices as the relative support. Searching for a covering cycle on a subset of targets seems to be a good heuristic for two main reasons. First, if found, no better patrolling strategy can be given for the same support. Indeed, a covering cycle guarantees capture for any covered target in presence of a missed detection. Second, by applying Definition 2, we are implicitly assuming to connect targets along shortest paths which, despite not being the optimal choice in general, in several realistic settings it configures as the only non–dominated scheme to connect targets.

Our approach can then be synthesized as follows (see Figure 1 for a graphical representation):

1. select a subset of targets T';
2. query a Covering Cycle Oracle (CCO) which returns a covering cycle C on T' if found and answers "NO" otherwise;
3. if a cycle was found, query a Signal Response Oracle (SRO) which computes the optimal signal response strategy (as per Problem (1)–(6)) assuming that C is the patrolling strategy and that the vertices covered by C constitute its support;
4. store the strategy and its value in a list;
5. repeat from Step 1 until a timeout expires and, in that case, select the strategy yielding the minimum (best for $\mathcal{D}$) value among those stored.

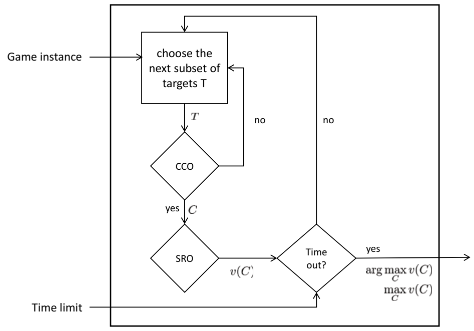


Figure 1: Algorithmic approach.

The covering cycle oracle (CCO)

The CCO takes as input a subset of targets T' and searches for a covering cycle (as per Definition 2). The oracle returns the covering cycle if found or a "NO" answer in the other case. This problem is $\mathcal{PSPACE}$–hard as we know from Theorem 1, so no exact efficient method can be designed. We resort to the approach of (Basilico, Gatti, and Amigoni 2012), where an algorithm for such problem is provided with correctness guarantees. It poses a limit to the length of the solution (say, $k|V|$ where k is a parameter) and applies a backtracking search. Although even with a fixed k the algorithm has exponential complexity, the algorithm demonstrated very good performance on realistic settings, even with a huge number of targets.

The signal response oracle (SRO)

The SRO takes as input a covering cycle C and computes the optimal signal–response strategy from every vertex in the support. We solve this problem separately for each vertex in the support since we work under the assumption that $\sigma_p^{\mathcal{D}}$ is fixed and equal to the covering cycle C. That is, by searching for the optimal response we account for the fact that each target in the support of $\sigma_p^{\mathcal{D}}$ will be fully protected upon a missed detection. This is immediately verified by the fact that Definition 2 implies the capture probability P_{capture} (see Constraint 7) equal to 1 for any target visited by the covering cycle C. The problem we obtain from (1)–(6) by casting $\sigma_p^{\mathcal{D}}$ to C is an LP, and therefore we are able to find the optimal response strategy $\sigma_a^{\mathcal{D}}$ efficiently.

Our implementation of SRO is based on the exact and approximate algorithms presented in (Basilico, De Nittis, and Gatti 2015). In particular, we will adopt two algorithms:

- Exact–DP: an exact method based on dynamic programming that runs in $O(2^{|T|})$;
- Approx–DP: an approximate method based on dynamic programming and l random restarts that runs in $O(l|T|^3)$.

Once each response strategy has been computed, the overall value of the strategy is simply given by the response that provides $\mathcal{A}$ with the highest expected utility.

Heuristics for selecting the targets in the support

As we explained in previous sections, by showing some structural properties of the optimal support, we are able to devise an algorithmic approach that requires implementation of two oracles (CCO and SRO) to map the process of computing the optimal strategy to a search in the power set of targets. The problem of refining such oracles or to devise better implementations remains open and lies outside the scope of this paper, whose aim is primarily the presentation of our analysis, approach, and experimental validation. However, we underline that our modular scheme can, in principle, integrate any implementation for such oracles leaving it open to possible future improvements. Moving to an enumeration on the power set of targets, which usually are less than the total number of vertices, we can obtain some computational advantages by means of a reduction of the search space. However, we are still dealing with an exponentially large size and the design of good heuristics to drive the search can be of critical importance. In this section, we describe the three heuristic methods we propose to effectively drive our enumeration of subsets of targets. The first one follows a dynamic approach in the sense that the ranking of possible solutions to explore changes as new subsets are explored. The last two, on the other side, follow a static approach, meaning that the order of preference between the candidate solutions is fully determined by the problem instance and does not vary during the search process.

Dynamic method based on Attacker responses (H1) The first mechanism to drive the search leverages the following rationale: if we searched subset T' and we obtained strategy σ', then constructing a T'', obtained by including in T' targets that provide $\mathcal{A}$ with large expected utilities under σ' and searching it, might result in a better strategy σ''. With this method, we progressively include in the support targets that are the most strategically appealing to the Attacker. (Notice that to assess how much a target can be appealing we cannot simply consider static parameters like its value or its penetration time, but we need to solve for a game–theoretic equilibrium. The approach is inspired to the *double–oracle method* commonly used in Operations Research and Security Games (Kiekintveld et al. 2009).)

Specifically, our method first searches among the singleton supports the one that represents the best static placement. Then, new targets are added to this base support by considering $\mathcal{A}$'s actions ranked according to their expected utility, from the best response to other non–optimal responses. The algorithm is essentially a greedy search. Formally:

1. compute the optimal σ prescribing a static placement w^* and set $U = \{w^*\}$;
2. sort targets not included in U according to the best expected utility value $\mathcal{A}$ can get by attacking them, calling $T^{(i)}$ the set of targets that, if attacked, will yield the i–th largest of such expected utilities;
3. for $i = 1, 2, \dots$ enumerate subsets Q_i, where $Q_i \subseteq U \cup T^{(i)}$ and $Q_i \not\subseteq U$; for each of them, compute σ by exploiting the two oracles (Figure 1);
4. repeat this procedure from Step 2 for each computed σ by setting $U = Q_i$.

This enumeration scheme is complete since, in the end, the optimal subset will be evaluated.

Static method based on targets values (H2) This method evaluates supports by considering the cumulative value of targets belonging to it. Thus, we adopt a static parameter (the value) to estimate the likelihood of having a target in the support. The rationale is that most valuable targets are more likely to be attacked. Thus, providing them with some extra protection given by the patrolling strategy can be a good choice. In practice, the method tries to search first the supports with an high cumulative value and with few targets. Formally:

1. rank targets according their value $\pi(\cdot)$, denote with $T^{(i)}$ the group of targets with the i–th highest value, set $U = T^{(1)}$ and $u = 1$;
2. enumerate subsets Q, where $Q \subseteq U$ and, if $u > 1$, $Q \not\subseteq \cup_{i \in \{1,\dots,u-1\}} U^i$; for each of them, compute σ by exploiting the two oracles (Figure 1);
3. set $U = U \cup U^{(u+1)}$, $u = u + 1$ and repeat from Step 2.

Static method based on targets distances (H3) The last method tries to privilege the subset of targets where the shortest distance between any pair of included ones in not above some given threshold. Here are the steps of the distance–based method:

1. set a threshold value $\tau = 1$;
2. enumerate all the subsets of targets T' such that for any $t_1, t_2 \in T'$ it holds that $d(t_1, t_2) \leq \tau$;
3. repeat from Step 2 after setting $\tau = \tau + 1$.

Experimental evaluation

Experimental setting We generate the worst–case instances for our problem from the instances of HAMILTONIAN PATH, with graphs where all the nodes are targets, edges are unitary, and, for each target, $\pi(t) \in (0, 1]$ and $d(t) = |T| - 1$. There is one single signal covering all the targets. We shall denote with ϵ the instance edge density defined as $\epsilon = |E| / \frac{|T|(|T|-1)}{2}$. Setting a timeout (in our case one hour) implicitly poses a limit on the maximum size (number of targets) of instances whose resolution can be completed. We implemented our algorithms in Matlab and we run them on a UNIX computer with 2.33GHz CPU and 16GB RAM.

CCO tuning One of the critical components of our resolution approach is the covering cycle oracle (CCO). As we explained above, we need to upper bound with $k|V|$ the solution length (the number of vertex visits made by the cycle) to trade off completeness for a lower computational effort (recall that the problem is $\mathcal{PSPACE}$–hard). Small values for k result in faster executions of the CCO due to the shrinkage induced on the solution space, but they could discard effective solutions. The experiment summarized in Fig. 2 shows this tradeoff with $k \in \{0.5, 1, 2\}$. For each value of k we report the number of covering cycles with at most $k|V|$ visits we are able to find in one hour (we enumerate subsets of vertices for increasing cardinalities checking for the existence of covering cycles with no more than $k|V|$ visits).

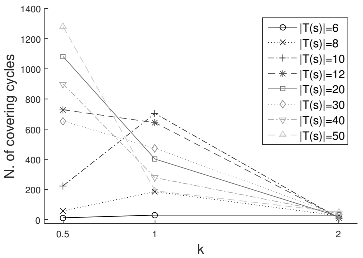
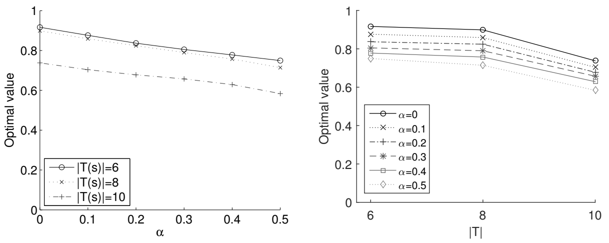


Figure 2: Number of covering cycles found in one hour with solution length upper bounded to $k|V|$.

The trends suggest $k = 1$ as acceptable bounding level when $|T| \leq 10$ and $k = 0.5$ when $|T| > 10$. Indeed, when k is larger, extremely few cycles are returned since CCO spends too long time in searching for a single a covering cycle and therefore not exploring a sufficient number of cycles. Interestingly, with $k = 2$ the number of returned covering cycles is close to 0.

Optimal SRO We set k as discussed above and we start by assessing the applicability extent of the exact SRO (Exact–DP algorithm). We do this by evaluating, at the same time, the impact of the false negative rate α on the optimal game value. In Fig. 3, we report results obtained within a one hour timeout averaged over 50 instances. The graphs suggest how such limit is 10 targets when employing the exact version of SRO in the game resolution process. Due to the time limit, one hour, such approach never terminates for instances composed of 12 targets. Furthermore, as intuition suggests, for increasing values of α the game's optimal value has a negative trend (approximately) linear in α. Fig. 3(a) depicts this behavior for instances with $\epsilon = 0.25$ (the same can be observed for higher ϵ). Also, it can be noticed how with more targets, the game value decreases (about exponentially in $|T|$), confirming that more targets correspond to more difficult environments to protect. (Fig. 3(b) shows the same trends w.r.t. $|T|$). We applied this approach also to realistic instances, built taking $\pi(t)$ from a uniform random distribution on the inteval $(0, 1]$ while $d(t)$ and $\omega^*_{i,j}$ are extracted from a uniform random distribution on the interval $(0, \delta]$, where δ is the diameter of the graph. As for the hard instances, $T = V$ and there is one single signal s that covers each target. Regarding these instances, within the time limit the algorithm solves graphs with up to 12 targets.



(a) (b)

Figure 3: Optimal game value with Exact–DP SRO.

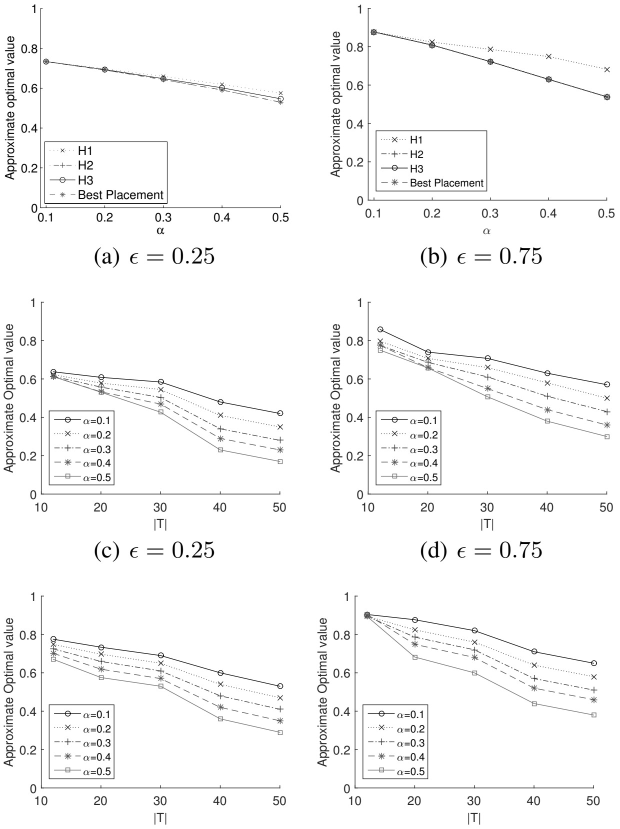
Approximate SRO Despite several common single–defender reality scenario can fit in the limit of 10 targets (e.g., when targets can be associated with large sub–areas), some others are characterized by a higher number of targets (e.g., when attackers can act on a fine discretization of the environment). This is what justified our choice to introduce an approximate method, moving the focus of the problem from scalability to solution's quality. As reported above, our SRO implementation based on the Approx–DP algorithms has a worst–case complexity of $O(|T|^3)$. This allowed us to process, within the one hour timeout, instances with up to 50 vertices (possibly scaling further for larger timeouts). The main questions here are two. First, we would like to assess the gap (in terms of utility) between the strategy we compute with our method and the best placement. Second, we want to compare the different heuristic methods we proposed in order to identify which one is the more suitable.

Fig. 4.(a)–(b) depict a representative example of what we observed during our simulations with $l = 10$ random restarts. Here we report results averaged over 50 instances each with 20 targets for two density values ($\epsilon = 0.25$ can be thought as representative of indoor cluttered environment while $\epsilon = 0.75$ could correspond to outdoor, highly interconnected, areas). Among the three heuristic methods, H1 (dynamic) is the only one yielding a competitive gap within the time limit w.r.t. the best placement strategy (up to about $+10\%$ with $\epsilon = 0.25$ and $+35\%$ with $\epsilon = 0.75$). H3 (distance) provided marginal improvements while H2 (value) very rarely outperformed the best placement. Notice how, as α gets close to 0, all the heuristics tend to yield the best placement value. Such results suggest that H1 (including the strategic component in ranking targets) is right direction to obtain better results in reasonable time. Surprisingly, the heuristic based on the targets' values (H2) performed rather poorly despite often used in reality as an evaluation principle. Due to the previous analysis, we made further experiments on H1, reported in Fig. 4 (c)–(f). Specifically, Fig. 4 (c)–(d) depicts the results with $l = 1$ as the number of targets vary, while Fig. 4.(e)–(f) do the same with $l = 10$, showing that with $|T| \geq 40$ all the heuristics do not improve the quality of the best placement (requiring thus a time longer than one hour). On the other side, it can be seen that $l = 10$ are useful in one hour with $|T| \geq 20$, allowing one to find better responses to the alarms, but they degrade the quality solution when $|T| > 20$ since the SRO takes too long time.

Conclusions and future research

In real–life scenarios, alarm systems play a crucial role for protecting complex environments and infrastructures. Despite the fact that security games constitute one of the most important application of multi–agent systems in real world, very few works deal with them. In the present work, we study, to the best of our knowledge, the first Security Game model in which patrolling strategies are combined with responses to signals raised by an alarm system, which is spatially uncertain and is affected by false negatives. We study the properties of our model and we exploit them to design an anytime algorithm based on two oracles and incorporating three heuristics, one dynamic and two static. Our algorithm searches strategies in which the patroller deterministically patrols a subset of targets in absence of any alarm and then responds at best to it as soon as the alarm is triggered. Our experimental evaluation shows that the dynamic heuristic outperforms the other two heuristics in terms of game value for the Defender.

In future, a thorough experimental evaluation of our algorithms could provide the best tradeoff between the parameters of the oracles once a time limit is given. The study of false positives, multi–attacker and multi–defender scenarios, is one of the most interesting topics to investigate in future.



(a) $\epsilon = 0.25$

(b) $\epsilon = 0.75$

(c) $\epsilon = 0.25$

(d) $\epsilon = 0.75$

(e) $\epsilon = 0.25$

(f) $\epsilon = 0.75$

Figure 4: Approximate game value with different heuristics.

Acknowledgements

We would like to thank Alessandro Colombo for his contribution to the experimental evaluation of models and algorithms presented in this work.

References

Agmon, N.; Kraus, S.; and Kaminka, G. A. 2008. Multi-robot perimeter patrol in adversarial settings. In *ICRA*, 2339–2345.

An, B.; Brown, M.; Vorobeychik, Y.; and Tambe, M. 2013. Security games with surveillance cost and optimal timing of attack execution. In *AAMAS*, 223–230.

Basilico, N., and Gatti, N. 2014. Strategic guard placement for optimal response to alarms in security games. In *AAMAS*, 1481–1482.

Basilico, N.; De Nittis, G.; and Gatti, N. 2015. Adversarial patrolling with spatially uncertain alarm signals. *arXiv preprint arXiv:1506.02850*.

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

Basilico, N.; Gatti, N.; and Rossi, T. 2009. Capturing augmented sensing capabilities and intrusion delay in patrolling-intrusion games. In *CIG*, 186–193.

Bertsekas, D. P. 1999. *Nonlinear Programming: 2nd Edition*. Athena Scientific.

Blum, A.; Haghtalab, N.; and Procaccia, A. D. 2014. Lazy defenders are almost optimal against diligent attackers. In *AAAI*, 573–579.

Cook, W.; Koch, T.; Steffy, D. E.; and Wolter, K. 2011. An exact rational mixed-integer programming solver. In *IPCO*, 104–116.

Ho, H.-M., and Ouaknine, J. 2015. The cyclic-routing uav problem is pspace-complete. In *FOSSACS*, 328–342. Springer.

Jain, M.; An, B.; and Tambe, M. 2012. An overview of recent application trends at the AAMAS conference: Security, sustainability, and safety. *AI MAG* 33(3):14–28.

Kiekintveld, C.; Jain, M.; Tsai, J.; Pita, J.; Ordóñez, F.; and Tambe, M. 2009. Computing optimal randomized resource allocations for massive security games. In *AAMAS*, 689–696.

Munoz de Cote, E.; Stranders, R.; Basilico, N.; Gatti, N.; and Jennings, N. 2013. Introducing alarms in adversarial patrolling games. In *AAMAS*, 1275–1276.

Yang, R.; Ford, B.; Tambe, M.; and Lemieux, A. 2014. Adaptive resource allocation for wildlife protection against illegal poachers. In *AAMAS*.