Artificial Intelligence 184–185 (2012) 78–123

Contents lists available at SciVerse ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint

Patrolling security games: Definition and algorithms for solving large instances with single patroller and single intruder

Nicola Basilico, Nicola Gatti *, Francesco Amigoni

Artificial Intelligence and Robotics Laboratory, Dipartimento di Elettronica e Informazione, Politecnico di Milano, Piazza Leonardo da Vinci 32, 20133 Milano, Italy

ARTICLE INFO

Article history:
Received 29 July 2010
Received in revised form 22 February 2012
Accepted 8 March 2012
Available online 12 March 2012

Keywords:
Security games
Adversarial patrolling
Algorithmic game theory

ABSTRACT

Security games are gaining significant interest in artificial intelligence. They are characterized by two players (a *defender* and an *attacker*) and by a set of targets the defender tries to protect from the attacker's intrusions by *committing* to a strategy. To reach their goals, players use *resources* such as patrollers and intruders. Security games are *Stackelberg games* where the appropriate solution concept is the *leader–follower equilibrium*. Current algorithms for solving these games are applicable when the *underlying game* is in *normal form* (i.e., each player has a single decision node). In this paper, we define and study security games with an *extensive-form infinite-horizon* underlying game, where decision nodes are potentially infinite. We introduce a novel scenario where the attacker can undertake actions during the execution of the defender's strategy. We call this new game class *patrolling security games* (PSGs), since its most prominent application is patrolling environments against intruders. We show that PSGs cannot be reduced to security games studied so far and we highlight their generality in tackling adversarial patrolling on arbitrary graphs. We then design algorithms to solve large instances with single patroller and single intruder.

1. Introduction

Security applications for transportation, shipping, airports, ports, and other infrastructures have recently received an increasing interest in the artificial intelligence literature [39,40,52,62]. The mainstream approach models a security problem as a two-player *non-cooperative game* [27] between a *defender* and an *attacker* with the aim to find effective strategies for the defender [51]. The basic ingredients are a number of *targets*, each with a value (possibly different for the two players), and a number of *resources* available to the defender to protect the targets and to the attacker to intrude them. In most situations of interest, the resources available to the defender are not enough to protect all the targets at once. This induces the defender to randomize over the possible assignments of resources to targets to maximize her expected utility. Furthermore, while the defender continuously and repeatedly protects the targets, the attacker is assumed to be in the position to observe the defender and derive a correct belief over her strategy. This last assumption pushes the defender to commit to a strategy and places security games in the more general class of *leader–follower* (also said *Stackelberg*) *games* where the leader is the defender and the follower is the attacker [64].

A leader–follower game is characterized by an *underlying game* and by the property that the leader can commit to a strategy. Von Stengel and Zamir studied this class of games in [64]. They show that, by committing to a particular strategy in a two-player normal-form underlying game, the leader cannot receive a utility worse than that she would receive when

* Corresponding author. Tel.: +39 02 2399 3658; fax: +39 02 2399 3411.
E-mail address: ngatti@elet.polimi.it (N. Gatti).

0004-3702/$ – see front matter
doi:10.1016/j.artint.2012.03.003

playing a Nash equilibrium. The *leader–follower equilibrium* is thus proposed as the appropriate solution concept. Letchford and Conitzer recently showed that the same result holds also when the underlying game is in extensive form [45].

In the last few years, several works have addressed the field of security games. For example, one of the most influential works is [51], where the problem of placing checkpoints to protect some targets from intrusions is studied (the approach has been practically applied at the Los Angeles International Airport [52]). The works proposed in this field are based on an underlying game in *normal form* (both players have a single decision node) and focus on the equilibrium computation problem, i.e., on the development of efficient algorithms to compute a leader–follower equilibrium.

The currently available models are not applicable when the attacker has the *option* to exploit the observation of the execution of the defender's actions to decide when to attack during the realization of the defender's plan without being subject to any temporal deadline (namely, with the possibility of waiting indefinitely for attacking). In [29], the author shows that by exploiting this option an attacker can drastically improve her expected utility.

In this paper, we propose a variant of security games that accounts for such an option. To do so, we consider a security game with an underlying game in *extensive form* with *infinite horizon*, players having multiple (potentially infinite) decision nodes. This contribution constitutes, to the best of our knowledge, an extension to the state of the art in security games.

The main *theoretical* motivation behind our work is that the currently available techniques are not efficient with this variant of security games. Indeed, their resolution requires techniques, largely unexplored in the security games literature, to reduce the size of the game instances. The main *practical* motivation behind our work is that the above option is available to the attacker in many scenarios, among which the most studied is probably *adversarial patrolling*, where one or more *patrollers* (the resources controlled by the defender, usually consisting in autonomous mobile robots) move within an environment to protect it and one or more *intruders* (the resources controlled by the attacker) wait outside the environment for the best time to attack. We focus on *patrolling* as reference scenario for our game models and we call them *patrolling security games* (PSGs). Formulating the adversarial patrolling problem as a PSG allows us to deal with environments represented as arbitrary graphs with targets. The drawback is that the needed computational effort is much larger than that required to solve settings with special topologies without targets (e.g., with closed perimeters [3]).

Our main original contributions, aiming at addressing the equilibrium computation problem for PSGs, follow.

(i) We model a PSG as a two-player multi-stage game with infinite horizon, where the defender moves a single resource on the vertices of an arbitrary graph environment to protect the targets while the attacker intrudes the environment by placing, for some time, a resource on a selected target vertex. We show that the equilibrium computation problem is a multi-quadratic mathematical programming problem that does not scale to realistically large settings. To tackle these limitations, we propose the following techniques.

(ii) We study the problem to find, when it exists, an equilibrium in pure strategies (namely, *deterministic patrolling strategies*). We show that this problem is a currently unexplored variant of the travel salesman problem (TSP) and that, although NP-complete, it can be efficiently solved by a constraint satisfaction programming algorithm, that solves with high success rate ($\geqslant 90\%$) significantly large instances ($\geqslant 500$ targets) in short time ($\leqslant 10$ s).

(iii) We develop reduction techniques to find a mixed strategy equilibrium (namely, *non-deterministic patrolling strategies*) in large game instances when no pure strategy equilibrium exists. We provide some reduction algorithms based on the combination of *removal of dominated actions* and *abstractions* and we show that no further general reductions *ex ante* the actual resolution can be provided. We show that with first-order Markovian strategies (that depend only on the vertex visited last by the patroller) our algorithm optimally solves medium-size game instances (up to 75 vertices and 15 targets) and sub-optimally solves large-size game instances (up to 166 vertices and 30 targets). We show that the quality of optimal and sub-optimal first-order Markovian solutions is at least 99% and 86%, respectively, of the quality of the optimal high-order Markovian solutions.

The structure of the paper follows. In Section 2, we survey the related works on security games and on robotic patrolling. In Section 3, we describe our game model and we extend the known techniques to solve it, showing their limitations. In Section 4, we discuss how a pure strategy equilibrium can be found when it exists. In Section 5, we provide techniques to reduce game instances and speed up the (mixed strategy) equilibrium computation. Our algorithm is summarized in Section 6 and experimentally evaluated in Section 7. Section 8 concludes the paper. Appendices A, B, and C report extensions, proofs, and complete experimental data, respectively.

2. Related works

We review security games and leader–follower equilibrium computation in Section 2.1. Next, we survey the main works on robotic patrolling in Section 2.2 and on other related fields in Section 2.3.

2.1. Security games and leader–follower equilibrium computation

The seminal work on security games is probably the von Neumann's *hide-and-seek game* [24]. It is a strategic-form zero-sum game played in grid environments where the hider chooses a location wherein to hide and the seeker chooses a set of locations wherein to seek. Starting from this work, several significant variations have been proposed in the literature.

A number of works study the problem where some *pursuers* search for *evaders* [1]. Game models in which both players are mobile are said *infiltration games* [8]. When the pursuer is mobile and the evader is immobile, we have *search games* [28]. When the situation is the reverse, we have *ambush games* [56]. A variation is the *interdiction game* [65] where the evader moves from a source to a sink (target), while the pursuer acts to prevent the evader to reach the target. All these works are defined on arbitrary graphs, but they do not consider targets with different importance.

Several variations of the interdiction games where targets have varying importance have been recently proposed with the goal to design randomized policies under scheduling constraints to protect targets. We call these works *protection games*. Significant examples are [39,40,50–52,62].

The PSG model we propose in this paper considers a mobile defender and an attacker on an arbitrary graph with targets of different importance. The attacker directly appears on a target and can be detected during the intrusion at the target (this model can be extended considering movements of the attacker). This is because intruding a target requires the attacker to spend some time on it. PSGs lay at the intersection between protection games, interdiction games, and search games. As the protection games, PSGs consider different targets with different importance with the aim to prevent intrusions. As in interdiction games, the pursuer can capture the attacker during the approach to the target. However, the attacker can also be captured after she reached a target, during the time she spends there for the intrusion. As in search games, the defender is mobile and the attacker can be immobile because, while intruding a target, she stays there for some time.

A crucial point of PSGs, common with protection games, is that the appropriate solution concept is the leader–follower equilibrium. Algorithms for computing a leader–follower equilibrium constitute a recent result. The seminal work, described in [20], shows that the computation of a leader–follower equilibrium can be formulated as a multi-linear mathematical programming problem with as many linear programs as the number of the follower's actions. This result shows that a leader–follower equilibrium can be found in polynomial time. An alternative formulation is provided in [50] where the problem is formulated as a mixed-integer linear mathematical programming problem.

2.2. Robotic patrolling

A broad definition of *patrolling* is "the act of walking or traveling around an area, at regular intervals, in order to protect or supervise it" [46]. Among the many scientific aspects that are involved in developing autonomous robots for patrolling (e.g., hardware and software architectures [46]), we focus on algorithms for producing *patrolling strategies*. We classify existing algorithms for patrolling along three main dimensions.

The first dimension concerns the patrolled area *representation*. It can be *graph-based* or *continuous* (by means of geometrical primitives, e.g., lines and polygons). With graph-based representations, there are four cases: *open perimeter*, *closed perimeter*, *fully connected* (every vertex is connected to all the others), and *arbitrary*. In all these cases, an environment may have only identical vertices or special vertices of interest, called *targets*.

The second dimension is the patroller's *objective function*. It can explicitly take into account the presence of adversaries (*adversarial*) or do not (*non-adversarial*). In the non-adversarial case, objective functions are mainly related to some form of *repeated coverage*, where the aim is to repeatedly cover the locations of a given area. Frequency-based objective functions related to repeated coverage can be defined as *constraint satisfaction* functions (e.g., patrolling all locations with the same frequency) or as functions that *maximize some measure*, e.g., the *maximal average frequency of visits* (also called *average idleness*), or the *maximal minimum* frequency of visit (also called *worst idleness*). When the environment has targets, frequencies of visits are expressed relative to targets. Other cases can be encountered that refer to *ad hoc* objective functions. In the adversarial case, there are two kinds of objective functions: *expected utility with fixed adversary*, where the expected utility of the patroller is maximized given a fixed non-rational model of the adversary, and *expected utility with rational adversary*, where the adversary is modeled as a rational decision maker.

The third dimension is the *number* of adopted patrollers (i.e., available resources). It can take *single agent* or *multiagent* values.

Table 1 shows the classification of the main works on robotic patrolling according to the above dimensions. The symbols $\diamond$ and $\star$ denote the contributions we provide in Sections 4 and 5, respectively. In the following, we review the main related works on patrolling reported in the table, organizing the discussion according to the representation of the environment.

The work in [23] provides efficient algorithms to find multiagent patrolling strategies for open-ended fences (i.e., open-ended polylines) that minimize different notions of idleness for realistic models of robot motion. Patrolling open perimeters is challenging because it is intrinsically inefficient, since robots must re-visit the just visited areas when they reach an endpoint and turn back. The work in [23] divides the continuous open polylines in discrete segments and determines the actions of the robots accordingly.

The work presented in [3] provides an efficient (polynomial time) algorithm to solve closed perimeter multiagent patrolling settings in a game theoretical fashion. The perimeter is continuous but is divided into segments that can be easily mapped to the vertices of a ring-like graph. The solving algorithms are referred to this discretized environment, as opposed to the algorithms that are based on continuous environments, like that in [38], for example. For this reason, we classify this work under graph-based environments. A possible intruder can enter any vertex and is required to spend a given time (measured in turns and called *penetration time*) to have success. The intruder and the patrollers have no preferences over the vertices and the intruder will enter the vertex in which the probability to be captured is minimum. The patrollers are

Table 1
Related works on robotic patrolling. The symbols ⋆ and ◇ denote the contributions of this paper.

				Graph–based				Continuous
				Open perimeter	Closed perimeter	Fully connected	Arbitrary	
Non–adversarial	Frequency–based	Constraint satisfaction	single agent				[66], ◇	[48]
			multiagent				[33,66]	[32,35]
		Maximization of a measure	single agent					
			multiagent	[23]			[7,18,22,46,59]	
	Others		single agent					
			multiagent				[47,54]	
Adversarial	Expected utility with fixed adversary		single agent					
			multiagent		[6]		[58]	
	Expected utility with rational adversary		single agent			[29]	[9], ⋆	
			multiagent		[2,3,4,5]			

synchronous. The problem is essentially a zero-sum game and the solution (i.e., the patrolling strategy) is the patrollers' maxmin strategy. In [4] and [5] the model is extended by considering realistic uncertainty over the robots' sensing and over adversary's knowledge. In [2] both the presence of events and different times of detection of the intruders, which yield different rewards to the patrollers, are considered. Finally, in [6] non-rational intruders are considered.

In [29] the author considers a fully connected topology graph where the (single) patroller and the intruder can have different preferences over the target vertices and provides an algorithm to compute a Nash equilibrium.

The approach in [66] considers single and multiagent patrolling on arbitrary graphs, but the goal is to patrol edges and not vertices. The objective function is the blanket time criterion and so the patrolling agents have to cover all the edges with the same frequency. The proposed ant-based algorithm is shown to converge to an Eulerian cycle in a finite number of steps and to re-visit edges with a finite period. A similar work is reported in [33].

Some works address the covering of environments represented as arbitrary graphs. The work in [46] deals with multiagent patrolling of vertices of graphs whose edges have unitary lengths. Several agent architectures are experimentally compared according to their effectiveness in minimizing the idleness. The approach is generalized in [7] to graphs in which edges have arbitrary lengths, and analyzed from a more theoretical perspective in [18]. Moreover, another work [59] proposes reinforcement learning as a way to coordinate patrolling agents and to drive them around the environment.

The work in [22] efficiently computes patrolling strategies for multiple agents minimizing the worst idleness in arbitrary graphs. Patrolling strategy is calculated exploiting a minimal Hamiltonian cycle.

In [47] and in [54] the authors consider multiagent patrolling settings with multi-criteria objective (e.g., idleness and distribution probabilities over the occurrence of incidents), which is pursued exploiting MDP techniques.

The work in [58] studies different adversaries: a random adversary, an adversary that always chooses to penetrate through a recently visited node, and an adversary that predicts the chances that a node will be visited. Some patrolling strategies for multiple agents are experimentally evaluated by simulation, showing that no strategy is optimal for all the possible adversaries.

In [9], the authors study arbitrary topology graphs providing an on-line heuristic approach to find the optimal strategies for a single patroller.

The contribution we provide in Section 4 (◇ in Table 1) studies a setting in which different targets of an arbitrary graph must be visited by a single agent with (at least) some frequencies, specific for each target, thus satisfying a set of constraints. The main differences with [66] are that we are patrolling vertices (and not edges) and that these vertices can have different requirements in terms of frequency of visits. Furthermore, our approach is not directly comparable with the other approaches for finding deterministic patrolling strategies, because we solve a feasibility problem (i.e., finding a patrolling strategy that satisfies some constraints), while other approaches solve an optimization problem (with some criterion).

The contribution we provide in Section 5 (⋆ in Table 1) generalizes both the works in [3] and [29] to arbitrary graphs with targets, but it is less computationally efficient for the settings to which both approaches are applicable. Moreover, it extends [58], capturing a rational adversary.

For completeness, we cite also some works that deal with continuous environments, even if they are not directly comparable with our graph-based approach. In [32], a system composed of multiple air vehicles that patrol a border area is presented. The environment is represented as a continuous two-dimensional region that is divided in sub-regions. Each

sub-region is assigned to an air vehicle that repeatedly patrols it with a spiral trajectory, ensuring that every point is covered. Also [35] considers multirobot patrolling of continuous environments. In this case, the environment is partitioned in sub-regions using a Voronoi tessellation, robots are assigned to sub-regions, and each robot patrols its sub-region in order to have a complete coverage. The movements of a robot are determined by a neural network model that allows to deal with dynamically varying environments. Finally, in [48], unpredictable chaotic trajectories are produced to have a robot covering a continuous environment.

2.3. Other related works

A large amount of works closely related to our patrolling problem can be found in the operational research literature as variations of the Traveling Salesman Problem (TSP). These works are close to graph-based frequency-based patrolling works, but the objective functions they adopt are not suitable for patrolling problems, as we discuss below.

In the *deadline-TSP* [63], vertices have deadlines over their first visit and some time is spent traversing arcs. Rewards are collected when a vertex is visited before its deadline, while penalties are assigned when a vertex is either visited after its deadline or not visited at all. The objective is to find a tour that visits as many vertices as possible. However, differently from what happens in patrolling, the reward/penalty is received only at the first visit.

In the *vehicle routing problem with time windows* [41], deadlines are replaced with fixed time windows, during which visits of vertices must occur. The time windows do not depend on the previous visits of the patroller, as it happens in patrolling. In the *period vehicle routing problem* [34], constraints could impose multiple visits to a same vertex in a time period.

Cyclical sequences of visits are addressed in the *period routing problem* [19,26], where vehicle routes are constructed to run for a finite period of time in which every vertex has to be visited at least a given number of times. In the *cyclic inventory routing problem* [53] vertices represent customers with a given demand rate and storage capacity. The objective is to find a tour such that a distributor can repeatedly restock customers under some constraints over visiting frequencies.

Despite these works have different similarities with the patrolling problem we consider in this paper, the application of such techniques to our setting is not straightforward and limited to very particular scenarios.

3. Game model, solution concept, and basic algorithm

In this section we introduce our approach. In particular, in Section 3.1 we describe the model of the PSGs, in Section 3.2 we discuss the appropriate solution concept, and in Section 3.3 we provide a solving algorithm inspired by the current state of the art.

3.1. Patrolling security game model

At first we describe the patrolling setting in Section 3.1.1 and then the game model in Section 3.1.2.

3.1.1. The patrolling setting

The patrolling setting is composed of an environment and of two players, an attacker **a** and a defender **d**, each with some specific resources. We assume discrete time (developing in turns) and we model the environment by means of a directed graph $G = (V, A, T, v, d)$. Set V contains vertices representing the areas of the environment. Set A contains arcs connecting vertices in V, providing the topology of the environment (graph representations can be extracted from real environments by, e.g., [43]); we represent A by a function $a : V \times V \to \{0, 1\}$, where $a(i, j) = 1$ if $(i, j) \in A$ and $a(i, j) = 0$ otherwise. Set $T \subseteq V$ contains the vertices, called *targets*, with some values for the defender and the attacker. v is defined as a pair of functions $v = (v_{\mathbf{d}}, v_{\mathbf{a}})$, where $v_{\mathbf{d}} : T \to \mathbb{R}^+$ assigns each target t the value for the defender of successfully protecting $t \in T$ and $v_{\mathbf{a}} : T \to \mathbb{R}^+$ assigns each target $t \in T$ the value for the attacker of successfully intruding t. Function $d : T \to \mathbb{N} \setminus \{0\}$ assigns each target $t \in T$ the time that the attacker needs to spend on t for completing an intrusion and getting $v_{\mathbf{a}}(t)$.

The attacker is modeled as follows: she can wait indefinitely outside the environment observing the defender's actions and perfectly deriving the defender's strategy (as in [3,51]); she can use a single resource, called *intruder*, to attack a target directly placing it on the target; once she has attacked a target t, she cannot control the intruder for a number of turns equal to $d(t)$, after which she removes the intruder from the environment.

The defender is modeled as follows: she can move a single resource, called *patroller*, along G spending one turn to cover one arc (as in the simplest motion model adopted in [3]); the patroller can sense only (and perfectly) the area corresponding to the vertex in which she is; once the patroller has sensed the intruder, the intruder is captured.

The simplifying assumptions on players do not prevent to capture realistic applicative scenarios. For example, the fact that an attacker can directly pose its resource on a target can be encountered in situations in which a patroller can detect an intruder only when this last one is not moving. On the defender's side, the simplified movement model represented by fixed weights on the graph's arcs can model the situation in which the patroller is a software agent traveling between nodes of a sensor network deployed in the environment and performing some data analysis on the current node. Moreover, these limitations can be partially relaxed as we show in Appendix A.3 and Appendix A.4 for the attacker and the defender, respectively.

Since in our patrolling setting each player has a unique resource, in the following we will use interchangeably the terms 'patroller' and 'defender', and similarly the terms 'intruder' and 'attacker'.

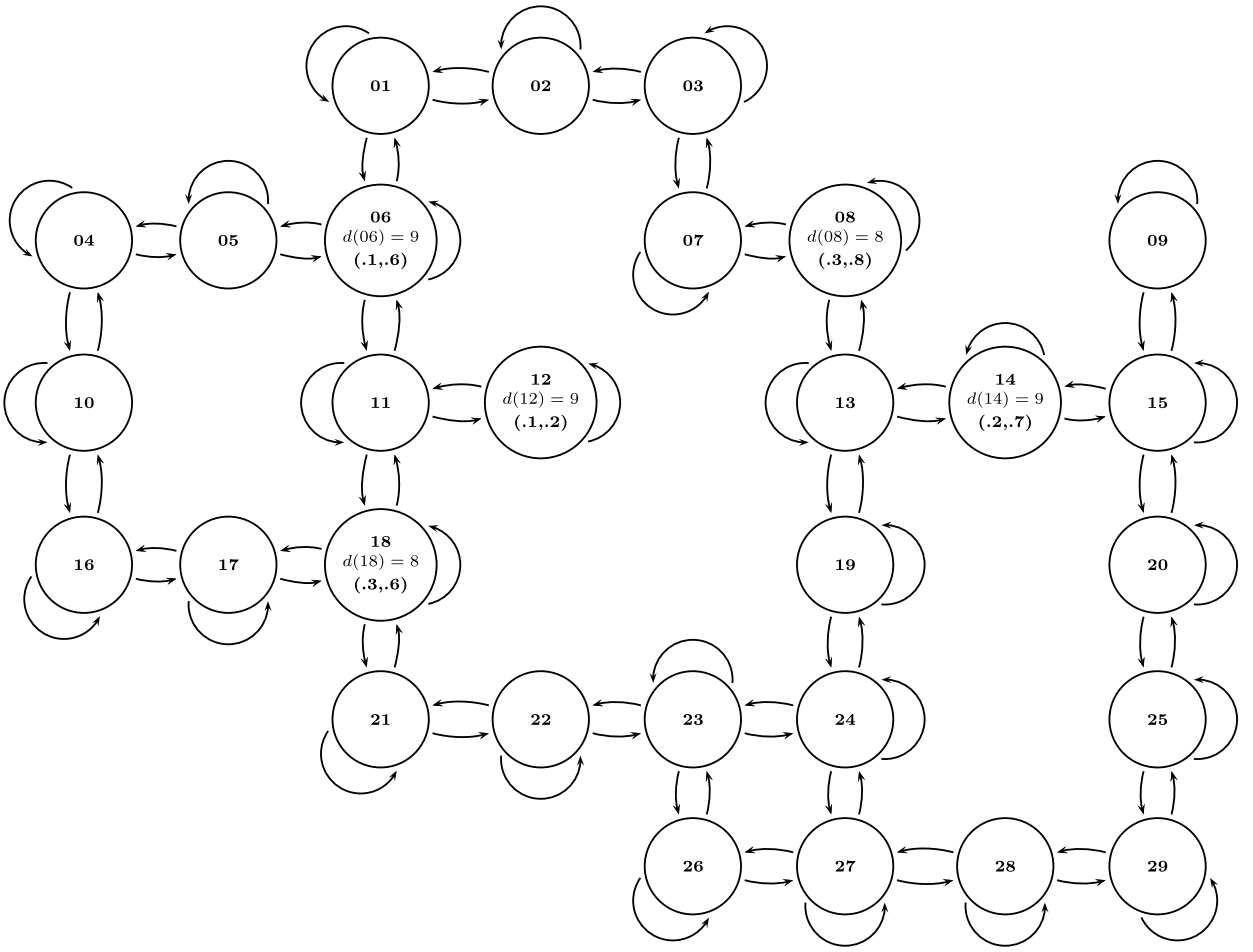


Fig. 1. The graph representing the patrolling setting used as running example. In each vertex we report: the number of the vertex, the penetration time $d(\cdot)$, and, between parentheses, the value of the defender and of the attacker, respectively.

Example 3.1. Fig. 1 depicts a patrolling setting where the circled numbers identify the vertices, arcs are depicted as arrows, and the set of targets is $T = \{06, 08, 12, 14, 18\}$; in each target t we report $d(t)$ and $(v_{\mathbf{d}}(t), v_{\mathbf{a}}(t))$.

3.1.2. The game model

PSGs are defined as two-player multi-stage games with imperfect information and infinite horizon [27]. Each stage of the game corresponds to a turn in the patrolling setting in which the defender and the attacker act simultaneously. The defender's available actions are denoted by *move*(j) where $j \in V$ is a vertex adjacent to the patroller's current one. If action *move*(j) is played at turn k, then at turn $k+1$ the patroller visits vertex j and checks it for the presence of the intruder. The attacker's available actions are denoted by *wait* and *enter*(t) with $t \in T$. Playing action *wait* at turn k means not to attempt any intrusion for that turn. Playing action *enter*(t) at turn k means to start an intrusion attempt in target t and prevents the attacker from taking any other action in the time interval $\{k+1, \ldots, k+d(t)-1\}$. Notice that playing *enter*(t) will make the game to conclude by $d(t)$ turns. The attacker's actions are not perfectly observable and thus the defender, when acting, does not know whether or not the intruder is currently within a target. The game has an infinite horizon, since the attacker is allowed to wait indefinitely for attacking.

The possible outcomes of the game are: *no-attack*: when the attacker plays *wait* at every turn k, i.e., it never attacks any target; *intruder-capture*: when the attacker plays *enter*(t) at turn k and the patroller visits target t in the time interval $I = \{k, k+1, \ldots, k+d(t)-1\}$ (and consequently detects the intruder); *penetration-t*: when the attacker plays *enter*(t) at turn k and the patroller does not visit target t in the time interval I defined above.

Example 3.2. Fig. 2 reports a portion of the game tree of the PSG for the setting of Fig. 1, given that the initial position of the patroller is vertex 01. Branches represent actions and players' information sets are depicted as dotted lines. Each turn of the game corresponds to two levels of the tree, where the defender and the attacker act simultaneously. We assume that players cannot observe each other's actions in the same turn.

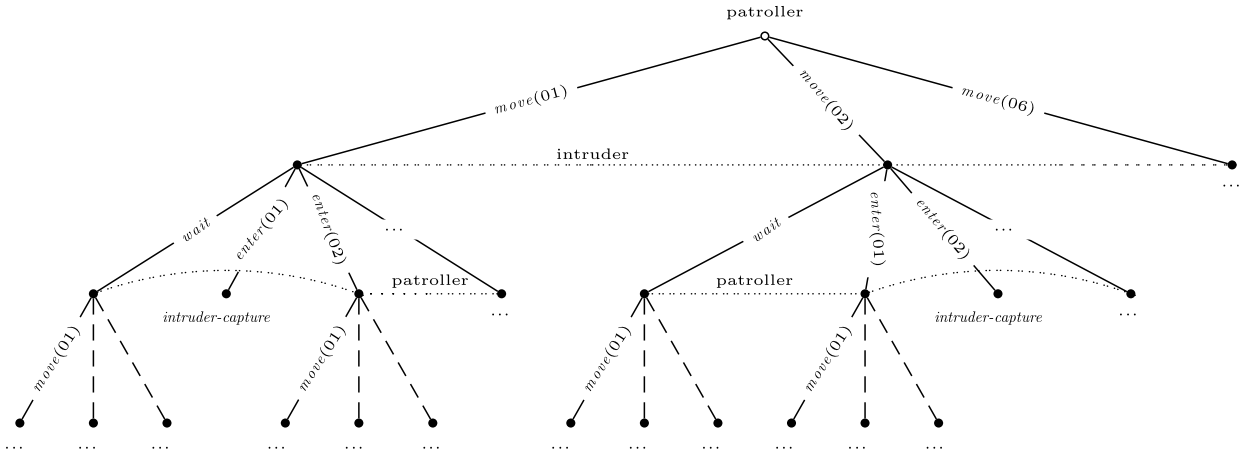


Fig. 2. A portion of the game tree for the setting of Fig. 1 (patroller is initially in 01).

Agents' utility functions are defined as follows. The defender's utility function $u_{\mathbf{d}}$ returns the total amount of preserved targets' value:

$$u_{\mathbf{d}}(x) = \begin{cases} \sum_{i \in T} v_{\mathbf{d}}(i), & x = \textit{intruder-capture} \text{ or } \textit{no-attack} \\ \sum_{i \in T \setminus \{t\}} v_{\mathbf{d}}(i), & x = \textit{penetration-t} \end{cases}$$

Notice that the defender gets the same utility when the intruder is captured and when the intruder never enters. This is because, in the case a utility surplus is given for capture, the defender could prefer a lottery between *intruder-capture* and *penetration-t* to *no-attack*. This behavior is not reasonable, since the defender's primary purpose in a typical patrolling setting is to preserve as much value as possible and not necessarily capture the intruder.

The attacker's utility function $u_{\mathbf{a}}$ returns a penalty in case the intruder is captured, otherwise it returns the value of the attacked target:

$$u_{\mathbf{a}}(x) = \begin{cases} 0, & x = \textit{no-attack} \\ v_{\mathbf{a}}(t), & x = \textit{penetration-t} \\ -\epsilon, & x = \textit{intruder-capture} \end{cases}$$

where $\epsilon \in \mathbb{R}^+$ is the penalty. In words, the *status quo* (i.e., *no-attack*) is better than *intruder-capture* for the attacker.

We denote by H the space of all the possible histories h of vertices visited by the patroller (or, equivalently, actions undertaken by the defender). For example, in Fig. 1, given that the patroller starts from vertex 01, a possible history is $h = \langle 01, 02, 03, 07, 08 \rangle$. We define the *defender's strategy* (also called *patrolling strategy*) as $\sigma_{\mathbf{d}} : H \to \Delta(V)$ where $\Delta(V)$ is a probability distribution over the vertices V. Given a history $h \in H$, the strategy $\sigma_{\mathbf{d}}$ gives the probability with which the patroller will move to vertices at the next turn. The defender's strategy does not depend on the actions undertaken by the attacker, these being unobservable for the defender.

We distinguish between *deterministic* and *non-deterministic* patrolling strategies. When $\sigma_{\mathbf{d}}$ is in pure strategies, assigning a probability of one to a single vertex for each possible history h, we say that the patrolling strategy is deterministic. Otherwise, the patrolling strategy is non-deterministic.

We define the *attacker's strategy* as $\sigma_{\mathbf{a}} : H \to \Delta(T \cup \{\textit{wait}\})$ where $\Delta(T \cup \{\textit{wait}\})$ is a probability distribution over T (or, equivalently, over the corresponding actions $\textit{enter}(t)$) and the action *wait*.

Example 3.3. In Fig. 1, a deterministic patrolling strategy could prescribe the cycle $\langle 04, 05, 06, 11, 18, 17, 16, 10, 04 \rangle$, while a non-deterministic patrolling strategy when the patroller is in vertex 01 after a history h could be:

$$\sigma_{\mathbf{d}}(h) = \begin{cases} 01 & \text{with a probability of 0.25} \\ 02 & \text{with a probability of 0.25} \\ 06 & \text{with a probability of 0.5} \end{cases}$$

An example of attacker's strategy is: play action *wait* for all the histories whose last vertex is not 04 and play $\textit{enter}(18)$ otherwise.

3.2. Solution concept

We initially discuss the appropriate solution concept when the defender cannot commit to a strategy (Section 3.2.1) and then we show that committing to a strategy is never worse for the defender (Section 3.2.2).

3.2.1. Solution concept in absence of any commitment

We consider the defender's strategy in absence of any commitment. The appropriate solution concept for a multi-stage game with imperfect information is the *sequential equilibrium* [44], which refines Nash equilibrium.

The presence of an infinite horizon complicates the study of the game. With an infinite horizon, classic game theory studies a game by introducing symmetries, e.g., a player will repeat a given strategy every $\bar{k}$ turns. Introducing symmetries in our game model amounts to force the players' strategies to be defined on histories with a maximum finite length l. As a result, the strategies are Markovian with a memory of order l.

Example 3.4. When $l = 0$, actions prescribed by the defender's strategy do not depend on any previous action and the probability to visit a vertex is the same for all the vertices where the patroller is. Notice that imposing $l = 0$ is not satisfactory for non-fully connected graph, where the set of actions available to the defender depends on the current vertex. When $l = 1$, the defender chooses her next action on the basis of her last action (equivalently, the next action depends only on the current vertex of the patroller). In this case, the patrolling strategy is first-order Markovian.

Obviously, when increasing the value of l, the defender's expected utility cannot decrease, because the defender considers more information to select her next action. Classical game theory [27] shows that games with infinite horizon admit a maximum length, say $\bar{l}$, of the symmetries such that the expected utility does not increase anymore with $l \geqslant \bar{l}$ (e.g., $\bar{l} = 0$ in [55]). Usually, $\bar{l} = 1$ [27]. In our model, this means that, when the defender's strategy is defined on the last l vertices visited by the patroller, with $l \geqslant \bar{l}$ the defender's expected utility is the same she receives with $l = \bar{l}$. Notice that the number of possible pure strategies $\sigma_{\mathbf{d}}(h)$ and $\sigma_{\mathbf{a}}(h)$ is $O(n^l)$, where n is the number of vertices. Therefore, we expect that, when increasing the value of l, the computational complexity for finding a patrolling strategy exponentially increases. In practical settings, the selection of a value for l is a trade-off between expected utility and computational effort.

3.2.2. Translation to a strategic-form game for a given l

Given a value for l, a PSG can be translated to a strategic-form game with additional constraints over the defender's strategies. In the strategic-form game, the defender's actions are all the feasible probability assignments for $\{\alpha_{h,i}\}$, where $\alpha_{h,i}$ is the probability to execute action *move*(i) given history h. The attacker's actions are *enter-when*(t, h), with $t \in T$, $h \in H$, and *stay-out*. Action *enter-when*(t, h) corresponds to make *wait* until the patroller has followed history h and then make *enter*(t); *stay-out* corresponds to make *wait* forever. The additional constraints, formally defined in Section 3.3.1 below, take into account that the defender's strategies in the original extensive-form game are repeated every l turns. Notice that the game does not depend on the initial vertex of the patroller. This is because the defender's and attacker's strategies do not depend on it.

Example 3.5. Consider Fig. 1. With $l = 1$, the available defender's strategies are all the consistent probability assignments to $\{\alpha_{i,j}\}$ with $i, j \in V$, while the attacker's actions are *enter-when*(t, j) with $t \in T$, $j \in V$ and *stay-out*.

It can be easily observed that this reduced game is strategically equivalent to the original game and therefore every equilibrium of this game is an equilibrium of the original game. Since a Nash equilibrium of a strategic-form game is also a sequential equilibrium, we have that a Nash equilibrium in the reduced game is a sequential equilibrium in the original game.

Since the attacker can wait outside the environment observing the defender's strategy, the defender's commitment to a strategy is credible. Therefore, the leader–follower equilibrium is the appropriate solution concept for PSGs. We state the following result, whose proof is an easy application of the result discussed in [64].

Proposition 3.6. *Given the game described above with a fixed l, the leader never gets worse when committing to a leader–follower equilibrium strategy.*

3.3. Basic algorithm

We apply the algorithm presented in [20] to our setting: we discuss in Section 3.3.1 how to compute the capture probabilities under the constraint that the patrolling strategies are repeated every l turns, and in Sections 3.3.2 and 3.3.3 how a leader–follower equilibrium can be computed when the game is zero-sum and general-sum, respectively. Then, we show in Section 3.3.4 that first-order Markovian strategies might not be optimal and we discuss the main limits of the approach in Section 3.3.5.

3.3.1. Computing the intruder capture probabilities

We denote by $P_c(t, h)$ the intruder capture probability related to action *enter-when*(t, h), defined as the probability that the patroller, starting from the last vertex of h, reaches target t by at most $d(t)$ turns. $P_c(t, h)$ depends on $\{\alpha_{h,i}\}$ in a highly non-linear way with degree $d(t)$. A bilinear (i.e., a special case of quadratic) formulation for the computation of $P_c(t, h)$ can be provided by applying the sequence-form [42] and imposing constraints over the behavioral strategies. From here on we

consider the formulation with $l = 1$ (in this case the history h reduces to a single vertex, i.e., $h \in V$). We define $\gamma_{i,j}^{w,t}$ as the probability with which the patroller reaches vertex j in w turns, starting from vertex i and not sensing target t. The constraints are:

$$\alpha_{i,j} \geqslant 0 \quad \forall i, j \in V \tag{1}$$

$$\sum_{j \in V} \alpha_{i,j} = 1 \quad \forall i \in V \tag{2}$$

$$\alpha_{i,j} \leqslant a(i,j) \quad \forall i, j \in V \tag{3}$$

$$\gamma_{i,j}^{1,t} = \alpha_{i,j} \quad \forall t \in T,\ i, j \in V \setminus \{t\} \tag{4}$$

$$\gamma_{i,j}^{w,t} = \sum_{x \in V \setminus \{t\}} \left(\gamma_{i,x}^{w-1,t} \alpha_{x,j}\right) \quad \forall w \in \left\{2, \ldots, d(t)\right\},\ t \in T,\ i, j \in V \setminus \{t\} \tag{5}$$

$$P_c(t,h) = 1 - \sum_{j \in V \setminus \{t\}} \gamma_{h,j}^{d(t),t} \quad \forall t \in T,\ h \in V \tag{6}$$

Constraints (1), (2) express that $\alpha_{i,j}$ are well defined probabilities; constraints (3) express that the patroller can only move between two adjacent vertices; constraints (4), (5) express the first-order Markov hypothesis over the defender's decision policy; constraints (6) define $P_c(t,h)$. The bilinearity is due to constraints (5). In the worst case (with fully connected graphs), the number of variables $\alpha_{i,j}$ is $O(|V|^2)$ and the number of variables $\gamma_{i,j}^{w,t}$ is $O(|T| \cdot |V|^2 \cdot \max_t\{d(t)\})$, while the number of constraints is $O(|T| \cdot |V|^2 \cdot \max_t\{d(t)\})$. As we show in Appendix A.1, the above formulation can be extended to the case in which l is arbitrary but, in practice, it is intractable, because the number of variables and constraints grows exponentially in l: the number of variables $\alpha_{i,j}$ is $O(|V|^{l+1})$ and the number of variables $\gamma_{h_1,h_2}^{w,t}$ is $O(|T| \cdot |V|^{2l} \cdot \max_t\{d(t)\})$, while the number of constraints is $O(|T| \cdot |V|^{2l} \cdot \max_t\{d(t)\})$. (A more efficient formulation, (about) halving the number of variables and constraints, is reported in Appendix A.2.)

3.3.2. Solving zero-sum patrolling security games

When the defender and the attacker share the same preferences over the targets (i.e., $v_{\mathbf{d}}(t) = v_{\mathbf{a}}(t)$ for all $t \in T$) the resulting game is essentially zero-sum. It is not rigorously zero-sum because two outcomes (i.e., *intruder-capture* and *no-attack*) provide the defender with the same utility and the attacker with two different utilities (i.e., $-\epsilon$ and 0, respectively). However, we can temporarily discard the outcome *no-attack*, assuming that action *stay-out* will not be played by the attacker. We will reconsider such action in the following. Without the outcome *no-attack* the game is zero-sum. In this case, the defender's leader–follower strategy corresponds to its maxmin strategy, i.e., the strategy that maximizes the defender's minimum expected utility. We provide a mathematical programming formulation to find it. We introduce the variable u, as the lower bound over defender's expected utility.

Formulation 3.7. The leader–follower equilibrium of a zero-sum PSG is the solution of:

$$\max u$$

$$\text{constraints } (1), (2), (3), (4), (5), (6)$$

$$u \leqslant u_{\mathbf{d}}(\textit{intruder-capture}) P_c(t,h) + u_{\mathbf{d}}(\textit{penetration-}t)\left(1 - P_c(t,h)\right) \quad \forall t \in T,\ h \in V \tag{7}$$

Constraints (7) define u as a lower bound on the defender's expected utility. By solving this problem we obtain the maximum lower bound u^*, i.e., the maxmin value. The values of variables $\{\alpha_{i,j}\}$ corresponding to u^* represent the optimal patrolling strategy. The number of constraints (7) is $O(|T| \cdot |V|)$. The formulation is bilinear and cannot be reduced to a linear problem because constraints (5) and (6) are not convex [17].

Now, we reconsider action *stay-out* and its corresponding outcome *no-attack*. Easily, from the solution of the above mathematical programming problem, we compute the attacker's expected utility, say v^*, as the utility of the attacker's best response given the capture probabilities corresponding to u^*. If $v^* < 0$, then the attacker will play *stay-out*. Otherwise, she will play the optimal action prescribed by the above mathematical program.

3.3.3. Solving general-sum patrolling security games

The mathematical programming formulation for the general-sum case is an extension of the multi-linear programming approach described in [20] (the approach proposed in [50] cannot be adopted here because we would obtain a mixed-integer quadratic problem and, currently, no solver would be able to solve it). In our case, the programming problem is a multi-bilinear one.

We define two mathematical programming problems. The first one checks whether or not there exists at least one defender's strategy $\sigma_{\mathbf{d}}$ such that *stay-out* is a best response for the attacker. If such a strategy exists, then the defender will follow it, its utility being maximum for *stay-out*.

Formulation 3.8. A leader–follower equilibrium in which the attacker's best strategy is *stay-out* exists when the following mathematical programming problem is feasible:

$$\text{constraints (1), (2), (3), (4), (5), (6)}$$
$$u_{\mathbf{a}}(\textit{intruder-capture})\,P_c(t,h) + u_{\mathbf{a}}(\textit{penetration-t})\big(1 - P_c(t,h)\big) \leqslant 0 \quad \forall t \in T,\ h \in V \tag{8}$$

Constraints (8) express that *stay-out* is better than *enter-when*(t,h) for all t and h. The number of constraints (8) is $O(|T| \cdot |V|)$.

If the above formulation is not feasible, we need to search for the attacker's best response such that the defender's expected utility is the largest. For each action *enter-when*(s,q) we calculate the optimal defender's expected utility under the constraint that such action is the attacker's best response.

Formulation 3.9. The largest defender's expected utility when attacker's best response is *enter-when*(s,q) is the solution of:

$$\max u_{\mathbf{d}}(\textit{penetration-s})\big(1 - P_c(s,q)\big) + u_{\mathbf{d}}(\textit{intruder-capture})\,P_c(s,q)$$
$$\text{constraints (1), (2), (3), (4), (5), (6)}$$
$$u_{\mathbf{a}}(\textit{intruder-capture})\big(P_c(s,q) - P_c(t,h)\big) + u_{\mathbf{a}}(\textit{penetration-s})\big(1 - P_c(s,q)\big)$$
$$- u_{\mathbf{a}}(\textit{penetration-t})\big(1 - P_c(t,h)\big) \geqslant 0 \quad \forall t \in T,\ h \in V \tag{9}$$

The objective function maximizes the defender's expected utility. Constraints (9) express that no action *enter-when*(t,h) gives a larger value to the attacker than action *enter-when*(s,q) (which is assumed to be the best response). The number of constraints (9) is $O(|T| \cdot |V|)$.

We calculate the patrolling strategies $\{\alpha_{i,j}\}$ of all the $|T| \cdot |V|$ above mathematical programming problems (one for each action *enter-when*(s,q) assumed to be the best response). The leader–follower equilibrium is the strategy $\{\alpha_{i,j}\}$ that maximizes the defender's expected utility.

Example 3.10. We report in Fig. 3 the patrolling strategy corresponding to the leader–follower equilibrium for the setting of Fig. 1. We have omitted all the vertices that are never visited by the strategy. The corresponding attacker's best response is *enter-when*$(08, 12)$.

3.3.4. Non-optimality of first-order Markovian strategies

The algorithm for solving PSGs reported in the previous sections has been formulated for $l = 1$. We showed in [12] that, when the graph representing the environment is fully connected, $\dot{l} = 0$ and therefore no strategy with $l > 0$ is better than the optimal strategy with $l = 0$. The problem of determining $\dot{l}$ for an arbitrary graph is very complex and largely beyond the scope of this paper. However, an interesting insight on this problem is given by the following proposition, whose proof is in Appendix B.1:

Proposition 3.11. *There are settings in which first-order Markovian patrolling strategies provide an expected utility strictly smaller than higher-order Markovian patrolling strategies.*

The above result entails that, in general, $\dot{l}$ may be larger than one. We can provide a lower bound over the efficiency of first-order Markovian strategies. Call $\frac{u}{u^*}$ the *efficiency* of a patrolling strategy σ, where u is the patroller's expected utility of playing σ and u^* is the patroller's expected utility of playing the optimal high-order Markovian strategy.

Theorem 3.12. *No topology-independent lower bound over the efficiency of a first-order Markovian leader–follower equilibrium strategy σ tighter than $\frac{u}{\sum_i v_{\mathbf{d}}(i)}$ can be provided, where u is the patroller's expected utility of playing σ.*

The proof is based on the fact that, given the values of a set of targets, we can always build a topology such that the deterministic strategy exists.

The following theorem shows a lower bound independent of the values of the targets (proof is reported in Appendix B.2).

Theorem 3.13. *The lower bound over the efficiency of a first-order Markovian leader–follower equilibrium strategy σ is $\frac{1}{2}$ and can be asymptotically achieved.*

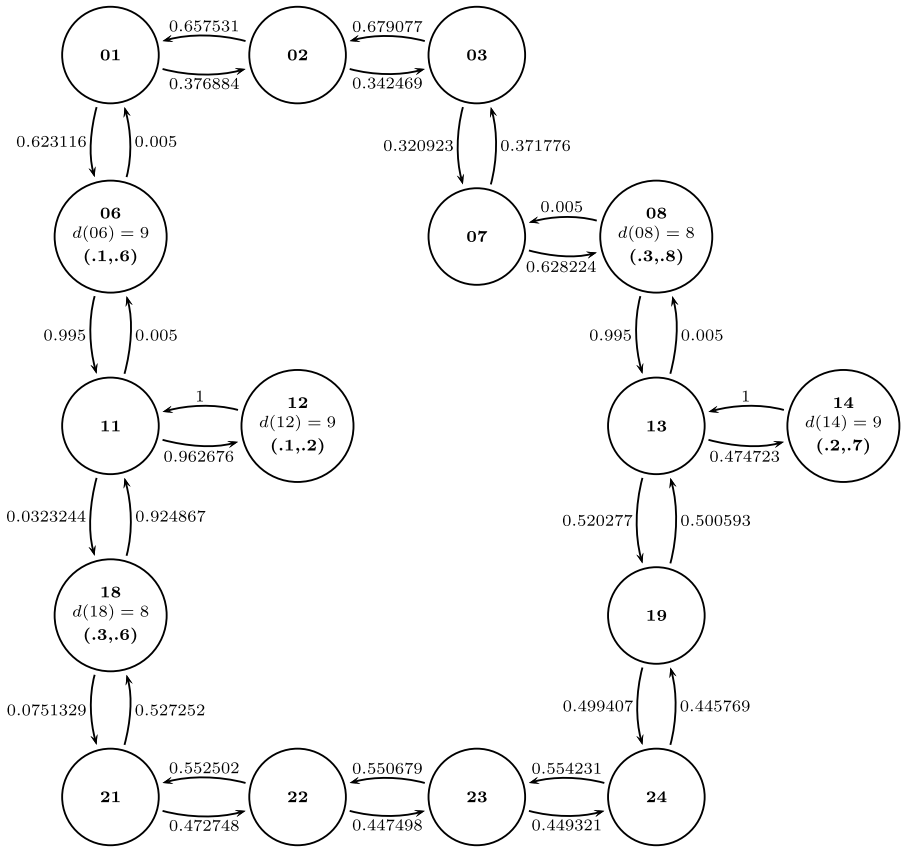


Fig. 3. Optimal patrolling strategy for Fig. 1 (the omitted vertices and arcs are never covered by the strategy).

3.3.5. Limits

The basic algorithm presented in the previous sections and based on the combination of results presented in [20] and [42] has two main limits.

The first limit of the approach is its *computational hardness* for solving realistically large game instances. In general, solving non-linear mathematical programming problems requires remarkable computational efforts. As we will discuss in our experimental evaluations, only small settings (w.r.t. the number of vertices and targets) can be solved with $l = 1$. Solving settings with $l > 1$ is practically intractable because, as we showed in Section 3.3.1, the number of variables and constraints rise exponentially with $2l$. This fact has two consequences. On the one hand, the limited scalability w.r.t. the settings' size prevents the model from being applied to practical scenarios, even with $l = 1$. On the other hand, the practical impossibility of increasing the value of l precludes the opportunity to find more effective patrolling strategies, whose existence is suggested by Proposition 3.11.

To tackle these issues, we propose two approaches. In the first one (Section 4), we limit the generality of the solution by looking only for deterministic (pure) strategies. We show that the limit on the value of l can be overcome in the specific case of deterministic strategies. More specifically, if a PSG admits an equilibrium deterministic strategy $\sigma_{\mathbf{d}}$ for an arbitrary value of l such that the associated intruder's best response is *stay-out*, then $\sigma_{\mathbf{d}}$ can be efficiently found by exploiting the structure of the problem, avoiding mathematical programming and reducing the computational burden. This is because the computation of equilibrium deterministic strategies is treated separately from the computation of more general equilibrium non-deterministic strategies. The second approach (Section 5) is based on the idea of simplifying the patrolling setting by introducing a pre-processing phase that eliminates variables and constraints, while preserving the game theoretical consistency and the solution optimality. As a consequence, a reduced patrolling setting can be solved more efficiently for non-deterministic patrolling strategies.

The second limit is that the resulting patrolling strategies may be *inconsistent*. This happens when an attacker's best response *enter-when*(t, x) has the property that x is never visited by the patroller after an infinite number of turns. In Fig. 4 we report an example of an inconsistent patrolling strategy. The intruder's best response given the patrolling strategy depicted in figure is *enter-when*$(12, 14)$, but the probability for the patroller of visiting 14 after an infinite number of turns is zero.

Essentially, a strategy inconsistency is due to the fact that a single patroller cannot patrol effectively all the targets. For maximizing its expected utility, the defender will patrol only a subset of the (most important) targets, leaving uncovered

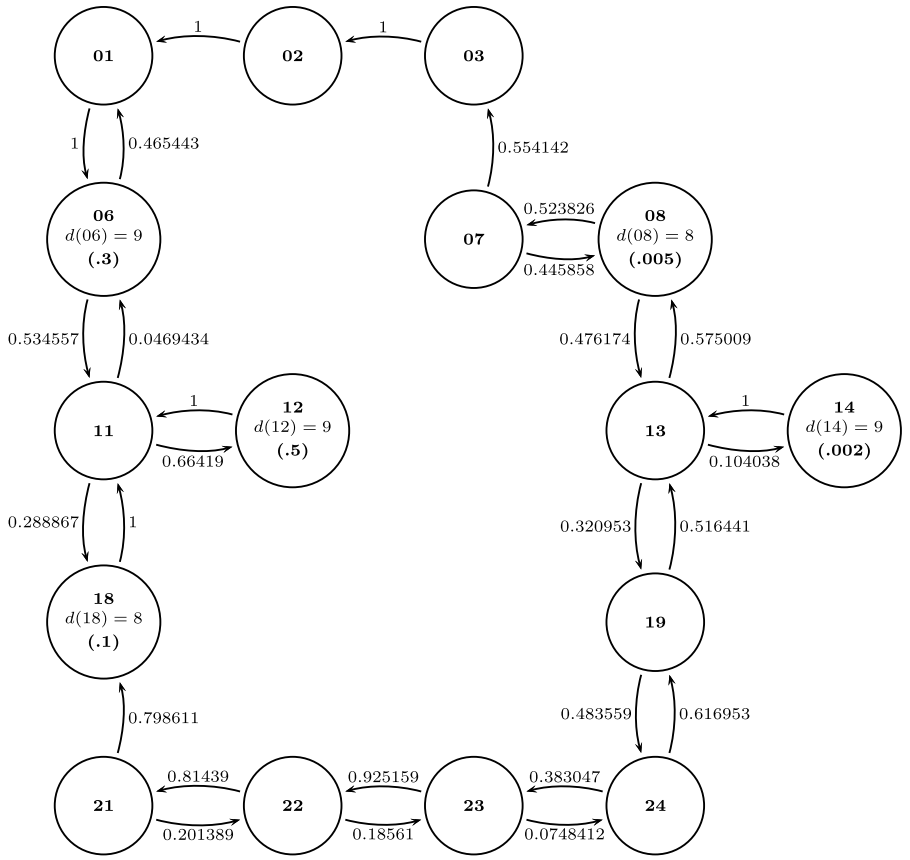


Fig. 4. Inconsistent strategy in a zero-sum PSG: some vertices cannot be covered even after an infinite number of turns. For example, vertex 14 cannot be reached from vertex 06.

other (less important) targets, and the attacker may prefer to attack a patrolled target rather than a non-patrolled one. In Fig. 4, the attacker prefers attacking target 12, even if the capture probability is strictly positive, rather than attacking target 14 with a capture probability of zero. An inconsistent patrolling strategy is not practically effective and must be discarded. In Section 6.2, we provide an algorithm that deals with inconsistencies. The algorithms we present in the following sections are not influenced by the presence of inconsistencies.

4. Finding deterministic patrolling strategies in large games

In this section, we describe a method to compute a deterministic patrolling strategy, a problem we initially addressed in [11]. In Section 4.1, we formally state the problem. We discuss its computational complexity in Section 4.2, we determine an upper bound over the solution's length and we propose a simple but inefficient algorithm in Section 4.3, and we provide a more efficient algorithm in Section 4.4.

4.1. Problem formulation

A deterministic patrolling strategy $\sigma_{\mathbf{d}}$ can be conveniently represented as a sequence of vertices, allowing l to be arbitrary. Apart from degenerate cases due to strategy inconsistencies (discussed in Section 6.2), a leader–follower equilibrium of a PSG can be in pure strategy (or deterministic) if the attacker's best response is *stay-out*, otherwise the defender can gain more by randomizing. By definition, when a deterministic equilibrium strategy is adopted by the patroller, each target t is left uncovered for a number of turns not larger than its penetration time $d(t)$ and thus every action *enter-when*(t, j) would result in a capture for the intruder. This kind of solution is close to those produced by the frequency-based approaches, but these are not applicable when the visit of each specific vertex is subject to specific constraints, as it happens in our case.

Without loss of generality, a deterministic strategy can be defined only on targets, assuming that the patroller will move between two targets along the shortest path. Accordingly, we reduce graph $G = (V, A, T, v, d)$ to a weighted graph $G' = (T, A', w, d)$, where targets T are the vertices of G'; A' is the set of arcs connecting the targets defined as a function $a' : T \times T \to \{0, 1\}$ and derived from set A as follows: for every pair of targets $i, j \in T$ and $i \neq j$, $a'(i, j) = 1$ if at least one of the shortest paths connecting i to j in G does not visit any other target, $a'(i, j) = 0$ otherwise; w is a weight function

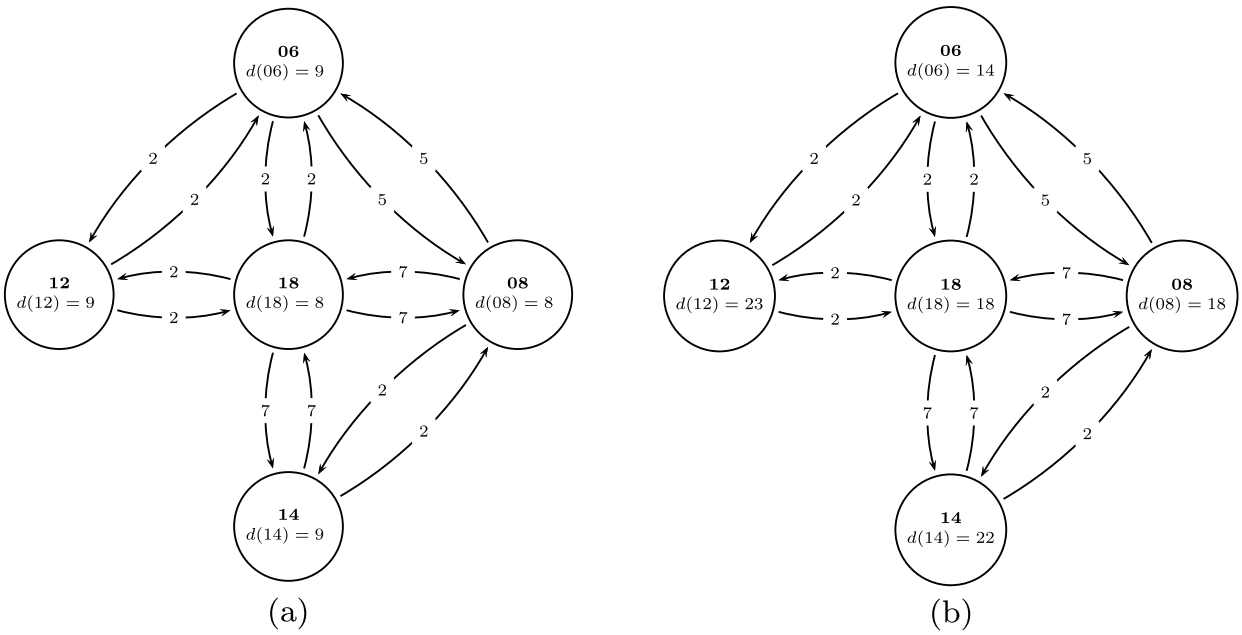


Fig. 5. (a) Reduced graph G' corresponding to that of Fig. 1. (b) The same graph as in (a), but with different (relaxed) penetration times.

defined as $w : T \times T \to \mathbb{N} \setminus \{0\}$ where $w(i, j)$ is the length of the shortest path between i and j in G ($w(i, j)$ is defined only when $a'(i, j) = 1$ and is the number of turns the patroller spends for going from target i to target j); and d is the same function defined as in G. The reduction from G to G' can be obtained by applying Dijkstra's algorithm to every pair of targets in G. For the sake of presentation, in the rest if this section we denote by σ a deterministic patrolling strategy over G' and we refer to vertices of G', instead of targets of G.

Example 4.1. Consider the graph reported in Fig. 1. The corresponding reduced graph G' is reported in Fig. 5(a). G' is composed of only 5 vertices. (The graph in Fig. 5(b) differs from that in (a) in the values of penetration times; we will use it in a later example.)

Pure strategy equilibria are usually found by iterating over players' best responses or by sampling strategy profiles. However, here the problem is different: we know the best response of the intruder, i.e., *stay-out*, and we need to search efficiently for the patroller's best strategy. The application of best response search methods would lead us to enumerate all the possible strategies and to check them one after another. This would be very inefficient, the search space being very large. We can provide a more convenient formulation based on constraint programming.

We define a function $\sigma : \{1, 2, \ldots, s\} \to T$ that represents a sequence of vertices of G', where $\sigma(j)$ is the jth element of the sequence. The length of the sequence is s and is not known *a priori*. The *temporal length* of a sequence of visits is computed by summing up the weights of covered arcs, i.e., $\sum_{j=1}^{s-1} w(\sigma(j), \sigma(j+1))$. The time interval between two visits of a vertex is calculated similarly, summing up the weights of the arcs covered between the two visits. A *solution* is a sequence σ such that: (i) σ is cyclical, i.e., the first vertex coincides with the last one, namely, $\sigma(1) = \sigma(s)$; (ii) every vertex in T is visited at least once, i.e., there are no uncovered vertices; (iii) when indefinitely repeating the cycle, for any $i \in T$, the time interval between two successive visits of i is never larger than $d(i)$.

Let us denote by $O_i(j)$ the position in σ of the jth occurrence of vertex i and by o_i the total number of i's occurrences in a given σ. For instance, consider Fig. 5(a): given $\sigma = \langle 14, 08, 18, 08, 14 \rangle$, $O_{08}(1) = 2$ and $O_{08}(2) = 4$, while $o_{08} = 2$ and $o_{06} = 0$. (Notice that, given a sequence σ, quantities $O_i(j)$ and o_i can be easily calculated.) With such definitions, we can formally re-state the problem in a constraint programming fashion (the presence of highly non-linear constraints makes it hard to resort to integer linear programming formulations, extending the works discussed in Section 2.3).

Formulation 4.2. A deterministic patrolling strategy σ such that the intruder's best response is *stay-out* is a solution of:

$$\sigma(1) = \sigma(s) \tag{10}$$

$$o_i \geqslant 1 \quad \forall i \in T \tag{11}$$

$$a'\big(\sigma(j-1), \sigma(j)\big) = 1 \quad \forall j \in \{2, 3, \ldots, s\} \tag{12}$$

$$\sum_{j=O_i(k)}^{O_i(k+1)-1} w\big(\sigma(j),\sigma(j+1)\big) \leqslant d(i) \quad \forall i \in T,\ \forall k \in \{1,2,\ldots,o_i-1\} \tag{13}$$

$$\sum_{j=1}^{O_i(1)-1} w\big(\sigma(j),\sigma(j+1)\big) + \sum_{j=O_i(o_i)}^{s-1} w\big(\sigma(j),\sigma(j+1)\big) \leqslant d(i) \quad \forall i \in T \tag{14}$$

Constraint (10) states that σ is a cycle, i.e., the first and last vertices of σ coincide; constraints (11) state that every vertex is visited at least once in σ; constraints (12) state that for every pair of consecutively visited vertices, say $\sigma(j-1)$ and $\sigma(j)$, $a'(\sigma(j-1),\sigma(j))=1$, i.e., vertex $\sigma(j)$ can be directly reached from vertex $\sigma(j-1)$ in G'; constraints (13) state that, for every vertex i, the temporal interval between two successive visits of i in σ is not larger than $d(i)$; similarly, constraints (14) state that for every vertex i the temporal interval between the last and first visits of i is not larger than $d(i)$, i.e., the deadline of i must be respected also along the cycle closure.

Example 4.3. Consider the graph of Fig. 5(a), no sequence σ of visits satisfies all the constraints listed above. Indeed, the shortest cycle covering only vertices 06 and 08, i.e., $\langle 06, 08, 06\rangle$, has a temporal length larger than the penetration times of both the involved vertices, so there is no way to cover these vertices (and others) within their penetration times. As we will show below, the graph of Fig. 5(b) admits a deterministic equilibrium strategy.

4.2. NP-completeness

Call DET-STRAT the problem of deciding if a deterministic patrolling strategy such that the intruder's best response is *stay-out*, as defined in the previous section, exists in a given G'.

Theorem 4.4. *The DET-STRAT problem is NP-complete.*

The proof, reported in Appendix B.3, shows that the Direct Hamiltonian Circuit problem can be reduced to the DET-STRAT problem. Although the DET-STRAT is a hard problem, we will show that it is possible to design a constraint programming based algorithm able to efficiently compute a solution for settings composed of a large number of targets.

4.3. An upper bound on the solution length and a simple algorithm

The peculiarity of the problem stated in Formulation 4.2 is that the length of the solution s and the number of occurrences o_i of vertex i are not known *a priori*, but they are part of the solution. The common approach adopted in the constraint programming literature to tackle problems with an arbitrary number of variables involves two phases: initially analytical bounds over the number of the variables are derived and then a set of problems, each one with the number of variables fixed to a value within the bounds, are solved. Although our problem resembles problems of cyclical CSP-based scheduling (e.g., [21]), to the best of our knowledge, the situation where the number of variables is part of the solution itself is still unaddressed. We derive a non-trivial upper bound over the temporal length of the solution.

Theorem 4.5. *If an instance of Formulation* 4.2 *is feasible, then there exists at least a solution σ with temporal length no longer than* $\max_{t\in T}\{d(t)\}$.

We report the proof in Appendix B.4. Exploiting Theorem 4.5, upper and lower bounds for the solution length s can be derived. They are defined respectively as $\bar{s} = \lceil \frac{\max_{t\in T}\{d(t)\}}{\min_{i,j}\{w(i,j)\}} \rceil$ and $\underline{s} = |T|+1$. Once we have fixed a value for s, upper and lower bounds over the number of occurrences of each vertex t are $\bar{o}_t = s-|T|+1$ and $\underline{o}_t = 1$ respectively. By using these bounds we can use Algorithm 1 to solve an instance of Formulation 4.2.

Algorithm 1: SIMPLE_DET-STRAT

```
for all the s in {s, s+1, ..., s̄} do
    for all the o = (o_(1), ..., o_(|T|)) in {1, 2, ..., s − |T| + 1}^|T| do
        assign σ ← CSP(s, o)
        if σ is not empty then
            return σ
return FAILURE
```

The call to $CSP(s,o)$ solves a standard constraint programming problem where the value of s and the number of targets' occurrences are fixed. This task can be easily accomplished with commercial CP solvers [37]. Despite its simplicity

and the possibility to use off-the-shelf solvers, Algorithm 1 is not efficient and requires a long time even for simple patrolling settings because it requires the resolution of many constraint programming problems and, for most of them, $CSP(s, o)$ explores the whole search space, which is exponential in the worst case. This pushes us to design an *ad hoc* algorithm.

4.4. Solving algorithm

We present our basic algorithm in Section 4.4.1, we report an execution example in Section 4.4.2, and we show how to improve it in Section 4.4.3.

4.4.1. Basic algorithm

We consider each $\sigma(j)$ as a variable with domain $F_j \subseteq T$. The constraints over the values of the variables are (10)–(14). We search for an assignment of values to all the variables such that all the constraints are satisfied. Our algorithm basically searches the state space with backtracking exploiting forward checking [57] in the attempt to reduce the branching of the search tree. Despite its simplicity, it is very efficient in practice. We report our method in Algorithms 2, 3, and 4.

Algorithm 2 assigns $\sigma(1)$ a vertex $i \in T$. Since the solution σ is a cycle that visits all vertices, every vertex can be assigned to $\sigma(1)$ without affecting the possibility to find a feasible solution.

Algorithm 2: FIND_SOLUTION(T, A', w, d)

```
1 select a vertex i in T
2 assign σ(1) ← i
3 call RECURSIVE_CALL(T, A', w, d, σ, 2)
```

Algorithm 3 assigns $\sigma(j)$ a vertex from domain $F_j \subseteq T$, which contains available values for $\sigma(j)$ that are returned by the forward checking algorithm (Algorithm 4). If F_j is empty or no vertex in F_j can be successfully assigned to $\sigma(j)$, then Algorithm 3 returns failure and a backtracking is performed.

Algorithm 3: RECURSIVE_CALL(T, A', w, d, σ, j)

```
1  if σ(1) = σ(j − 1) and constraints (11) hold then
2      if constraints (14) hold then
3          return σ
4      else
5          return FAILURE
6  else
7      assign F_j ← FORWARD_CHECKING(T, A', w, d, σ, j)
8      for all the i in F_j do
9          assign σ(j) ← i
10         assign σ' ← RECURSIVE_CALL(T, A', w, d, σ, j + 1)
11         if σ' is not FAILURE then
12             return σ'
13     return FAILURE
```

Algorithm 4 restricts F_j to the vertices that are directly reachable from the last assigned vertex $\sigma(j-1)$ and such that their visits do not violate constraints (13)–(14). Notice that checking constraints (13)–(14) requires knowing the weights (temporal costs) related to the arcs between vertices that could be assigned subsequently, i.e., between the variables $\sigma(k)$ with $k > j$. For example, consider the graph of Fig. 5(b) and suppose that the partial solution currently constructed by the algorithm is $\sigma = \langle 14 \rangle$. In this situation, we cannot check the validity of constraints (13)–(14) since we have no information about times to cover the arcs between the vertices that will complete the solution. Therefore, we estimate the unknown temporal costs by employing an admissible heuristic (i.e., a non-strict underestimate) based on the minimum cost between two vertices. The heuristic being admissible, no feasible solution is discarded. We denote the heuristic value by $\overline{w}$, e.g., $\overline{w}(i, \sigma(1))$ denotes the weight of the shortest path between i and $\sigma(1)$. We assume $\overline{w}(i, i) = 0$ for any vertex i.

Given a partial solution σ from 1 to $j-1$, the forward checking algorithm considers all the vertices directly reachable from $\sigma(j-1)$ and keeps those that do not violate the relaxed constraints (13)–(14) computed with heuristic values. More precisely, it considers a vertex i directly reachable from $\sigma(j-1)$ and assumes that $\sigma(j) = i$. Step 5 of Algorithm 4 checks relaxed constraints (14) with respect to i, assuming that the weight along the cycle closure from $\sigma(j) = i$ to $\sigma(1)$ is minimum. In the above example, with $\sigma(1) = 14$, the vertices directly reachable from $\sigma(1)$ are 08 and 18. The algorithm considers $\sigma(2) = 08$. By Step 5, we have $w(\sigma(1), 08) + \overline{w}(08, \sigma(1)) = 4 \leqslant d(08) = 18$ and then Step 5

is satisfied. It can be easily observed that such condition holds also when $\sigma(2) = 18$. Step 8 of Algorithm 4 checks relaxed constraints (14) with respect to all the vertices $k \neq i$, assuming that both the weight to reach k from $\sigma(j) = i$ and the weight along the cycle closure from k to $\sigma(1)$ are minimum. Consider again the above example. It can be easily observed that when $\sigma(2) = 08$ such conditions hold for all k. Instead when $\sigma(2) = 18$ and $k = 06$, we have $w(\sigma(1), 18) + \overline{w}(18, 06) + \overline{w}(06, \sigma(1)) = 16 > d(06) = 14$. The relaxed constraint is violated and vertex 18 will not be inserted in F_j. Similarly, Step 6 checks relaxed constraints (13) with respect to i and Step 9 checks relaxed constraints (13) with respect to any k assuming that the weight to reach k from $\sigma(j) = i$ is minimum. In the above example, starting from $\sigma = \langle 14 \rangle$, the relaxed constraints are satisfied only when $i = 08$ and therefore $F_j = \{08\}$. Finally, we notice that Steps 5 and 8 are checked only when $o_i = 0$ and $o_k = 0$, respectively, since it can be easily proved that when $o_i > 0$ and $o_k > 0$ these conditions always hold.

Algorithm 4: FORWARD_CHECKING(T, A', w, d, σ, j)

```
1  assign F_j ← ∅
2  assign s ← j − 1
3  for all members i in T such that a'(σ(s), i) = 1 do
4      if conditions
5      (o_i = 0 ∧ Σ_{l=1}^{s−1} w(σ(l), σ(l+1)) + w(σ(s), i) + w̄(i, σ(1)) ⩽ d(i) or
6      o_i > 0 ∧ Σ_{l=O_i(o_i)}^{s−1} w(σ(l), σ(l+1)) + w(σ(s), i) ⩽ d(i)) and,
7      for all k ≠ i,
8      (o_k = 0 ∧ Σ_{l=1}^{s−1} w(σ(l), σ(l+1)) + w(σ(s), i) + w̄(i, k) + w̄(k, σ(1)) ⩽ d(k) or
9      o_k > 0 ∧ Σ_{l=O_k(o_k)}^{s−1} w(σ(l), σ(l+1)) + w(σ(s), i) + w̄(i, k) ⩽ d(k))
10     hold then
11         add i to F_j
12 return F_j
```

We state the following theorem, whose proof is in Appendix B.5.

Theorem 4.6. *The above algorithm is sound and complete.*

4.4.2. Example

We apply our algorithm to the example of Fig. 5(b). We perform a random selection in Step 1 of Algorithm 2 (to choose the first visited vertex of the sequence) and in Step 7 of Algorithm 3 (to choose the elements of F_j as part of the current candidate solution). We report part of the execution trace (Fig. 6 depicts the complete search tree). (a) The algorithm assigns $\sigma(1) = 14$. (b) The domain F_2 (depicted in the figure between curly brackets beside vertex $\sigma(1) = 14$) is produced (according to the discussion of the previous sections) as follows: vertex 08 is added to F_2, since all the conditions in Algorithm 4 with $i = 08$ are satisfied; vertex 18 is not added to F_2, since the condition in Step 8 of Algorithm 4 with $k = 06$ is not satisfied, formally, $w(14, 18) + \overline{w}(18, 06) + \overline{w}(06, 14) > d(06)$; no other vertex is added to F_2, since no other vertex is directly reachable from 14. (c) The algorithm assigns $\sigma(2) = 08$. (d) The domain F_3 is produced similarly as above, yielding to $F_3 = \{06\}$. (e) The algorithm assigns $\sigma(3) = 06$ and continues.

Some issues are worth noting. In the 10th node of the search tree, a sequence σ with $\sigma(1) = \sigma(s)$ and including all the vertices was found. However, this sequence does not satisfy constraints (14). If the search is not stopped and backtracked at the 10th node (in Step 5 of Algorithm 3), the algorithm would never terminate. Indeed, the subtrees that would follow this vertex would be the infinite repetition of part of the tree already built. Finally, in the 6th node, no possible successor is allowed by the forward checking, and therefore the algorithm backtracks.

4.4.3. Improving efficiency and heuristics

Our algorithm can be improved as follows. Consider the conditions in Steps 5 and 8 of Algorithm 4. Except for the first execution of Algorithm 4 (i.e., when $j = 2$), the satisfaction of the condition at Step 5 for a given j is guaranteed if the condition in Step 8 for $j - 1$ is satisfied. Therefore, we can safely limit the algorithm to check the conditions at Step 5 exclusively when $j = 2$. The same considerations hold also for the conditions in Steps 6 and 9. Therefore, we can safely limit the algorithm to check the conditions at Steps 6 and 9 exclusively when $j = 2$.

We introduce a more sophisticated stopping criterion called LSC (Length Stopping Criterion) based on Theorem 4.5 and such that if $\sum_{l=1}^{s-1} w(\sigma(l), \sigma(l+1)) + \overline{w}(\sigma(s), \sigma(1)) > \max_{t \in T}\{d(t)\}$, then the search is stopped and backtracked. We also introduce an *a priori* check (IFC, Initial Forward Checking): before starting the search, we consider each vertex as the root node of the search tree and we apply the forward checking. If at least one domain is empty, the algorithm returns failure. Otherwise, the tree search is started.

Finally, we propose some heuristic criteria for choosing from set F_j the next vertex to expand in Step 8 of Algorithm 3: lexicographic (h_l), random with uniform probability distribution (h_r), maximum and minimum number of incident arcs ($h_{\max a}$ and $h_{\min a}$), less visited ($h_{\min v}$), and maximum and minimum penetration time ($h_{\max d}$ and $h_{\min d}$). For all the ordering

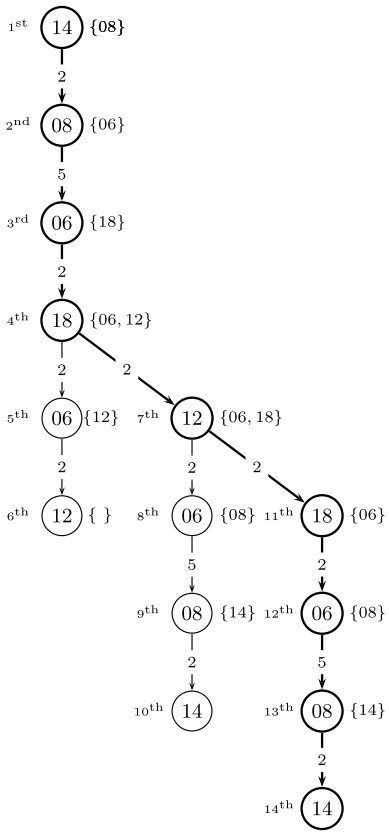


Fig. 6. Search tree for the example of Fig. 5(b); bold nodes and arrows denote the obtained solution; F_js are reported besides nodes $\sigma(j-1)$; xth denotes the order in which the tree's nodes are analyzed.

criteria except h_r, we use a criterion for breaking ties that randomly selects a vertex with a uniform probability (RTB, Random Tie Break). We can use the same heuristics also for selecting the initial node of the search tree in Step 1 of Algorithm 2. In Section 7, we will experimentally evaluate these heuristics.

5. Finding non-deterministic patrolling strategies in large games

In this section, we describe techniques to reduce game instances to make the computation of non-deterministic patrolling strategies affordable for large games. In Section 5.1 we present some algorithms to remove agents' dominated strategies. In Section 5.2 we discuss how to compute utility lossless abstractions and in Section 5.3 how to compute abstractions with utility loss.

5.1. Removing dominated strategies

Action *a dominates* action *b* when the expected utility of playing *a* is larger than that of playing *b* independently of the actions played by other agents and, therefore, no rational agent will play *a*. Removing dominated actions allows one to obtain an equivalent (with the same equilibria) but smaller game with a consequent reduction of the computational time needed for its resolution. We present two techniques to remove the defender's and attacker's dominated actions in Section 5.1.1 and Section 5.1.2, respectively, and then we discuss the possibility to iterate the removal in Section 5.1.3.

5.1.1. Removing defender's dominated actions

A defender's action $move(j)$ is dominated when, if such action is removed from the set of defender's available actions and thus the defender cannot visit vertex j, its expected utility does not decrease. This happens when, after removing $move(j)$, no capture probability $P_c(t,i)$ $\forall t \in T$, $i \in V \setminus \{j\}$ (i.e., for each attacker's strategy) decreases. Practically, removing $move(j)$ means removing vertex j and all its incident arcs from G.

Defender's dominated actions are identified in two steps. The first one focuses on vertices and corresponding incident arcs and is based on the following theorem, whose proof is reported in Appendix B.6.

Theorem 5.1. *Visiting a vertex that is not on any shortest path between any pair of targets is a dominated action.*

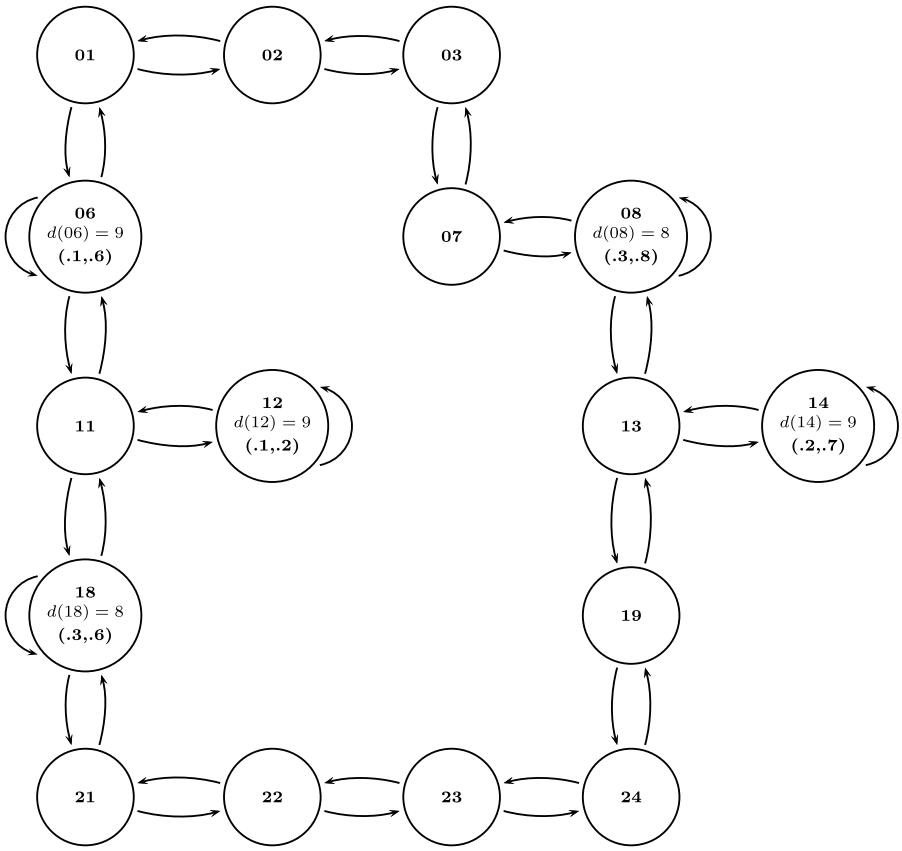


Fig. 7. Graph G_r for the patrolling setting of Fig. 1, obtained by removing the defender's dominated strategies.

When there are multiple shortest paths connecting the same pair of targets (t_1, t_2), visiting each vertex of some of them can be a dominated action. We state the following theorem, whose proof is reported in Appendix B.7.

Theorem 5.2. *Given targets t_1 and t_2 and two shortest paths $P = \langle t_1, \ldots, p_i, \ldots, t_2\rangle$ and $Q = \langle t_1, \ldots, q_i \ldots, t_2\rangle$ of length L between them, if for all $k \in \{2, \ldots, L-1\}$ and $t \in T \setminus \{t_1, t_2\}$ we have $dist(p_k, t) \geqslant dist(q_k, t)$, then visiting each internal vertex of P (i.e., all p_i excluding t_1 and t_2) is dominated.*

The first step identifies actions that are dominated independently of the current vertex of the patroller. If *move*(j) is dominated, then the patroller should not visit j from any adjacent vertex.

In the second step we account for the current vertex occupied by the patroller by considering all the defender's actions *move*(j), when the current vertex is i. We state the following theorem, whose proof is in Appendix B.8:

Theorem 5.3. *If the patroller is in vertex $i \in V \setminus T$, remaining in the same vertex i for a further turn is a dominated action.*

The application of Theorem 5.3 allows us to remove all the self-loops of $V \setminus T$. No more defender's strategies can be removed independently of the attacker's strategy, otherwise the capture probabilities and the defender's expected utility could decrease. Therefore, the above theorems allow one to remove all the defender's dominated strategies.

We call $G_r = (V_r, A_r, T, v, d)$ the reduced graph produced by removing from G all the vertices and arcs according to Theorems 5.1, 5.2, and 5.3. From here on, we work on G_r, instead of G.

Example 5.4. We report in Fig. 7 the graph G_r for our running example of Fig. 1 after having removed the vertices and arcs corresponding to the defender's dominated strategies.

5.1.2. Removing attacker's dominated actions

Attacker's action *enter-when*$(\bar{t}, \bar{i})$ is dominated by action *enter-when*$(\bar{s}, \bar{j})$ if $EU_{\mathbf{a}}(\textit{enter-when}(\bar{t}, \bar{i})) \leqslant EU_{\mathbf{a}}(\textit{enter-when}(\bar{s}, \bar{j}))$ for every (mixed) strategy $\sigma_{\mathbf{d}}$, where $EU_{\mathbf{a}}(\cdot)$ is the attacker's expected utility. Checking whether an action is dominated can be formulated as an optimization problem.

Formulation 5.5. Action *enter-when*$(\bar{t}, \bar{i})$ is dominated by *enter-when*$(\bar{s}, \bar{j})$ if the result of the following mathematical program is not strictly positive:

$$\max \mu \tag{15}$$

$$\begin{aligned} &\text{constraints (1), (2), (3), (4), (5), (6)} \\ &u_{\mathbf{a}}(\textit{penetration-}\bar{t})\big(1 - P_c(\bar{t}, \bar{i})\big) - u_{\mathbf{a}}(\textit{penetration-}\bar{s})\big(1 - P_c(\bar{s}, \bar{j})\big) \\ &\quad + u_{\mathbf{a}}(\textit{intruder-capture})\big(P_c(\bar{t}, \bar{i}) - P_c(\bar{s}, \bar{j})\big) = \mu \end{aligned} \tag{16}$$

Constraints (16) define μ as a lower bound on the difference between the expected utilities of actions *enter-when*$(\bar{t}, \bar{i})$ and *enter-when*$(\bar{s}, \bar{j})$. μ corresponds to the maximum achievable difference and therefore, if not positive, *enter-when*$(\bar{t}, \bar{i})$ is dominated by *enter-when*$(\bar{s}, \bar{j})$. The above problem has (asymptotically) the same number of constraints of Formulation 3.7.

The non-linearity, the size of each problem, and the large number of problems to be solved (one for each pair of actions), make Formulation 5.5 computationally expensive for the removal of the attacker's dominated actions. However, by exploiting the problem structure, we can design a more efficient algorithm. Initially, we state the following theorem that provides two necessary and sufficient conditions for dominance (we exploit fully mixed strategies in which every action is played with strictly positive probability to remove even weakly dominated strategies); the proof is in Appendix B.9.

Theorem 5.6. *Action enter-when*$(\bar{t}, \bar{i})$ *is dominated by enter-when*$(\bar{s}, \bar{j})$ *if and only if for all fully mixed strategies* $\sigma_{\mathbf{d}}$ *it holds that*:

(i) $u_{\mathbf{a}}(\textit{penetration-}\bar{t}) \leqslant u_{\mathbf{a}}(\textit{penetration-}\bar{s})$ *and*
(ii) $P_c(\bar{t}, \bar{i}) > P_c(\bar{s}, \bar{j})$.

Now we provide an efficient algorithm that removes dominated actions by using conditions (i) and (ii) of Theorem 5.6. We report it as Algorithms 5 and 6. The algorithm builds trees where each node q contains a vertex $\eta(q)$. For each target t, we build a tree of depth $d(t)$ where the root is t and the successors of a node q are all the nodes q' such that: $\eta(q')$ is adjacent to $\eta(q)$ (i.e., $a(\eta(q), \eta(q')) = 1$) and $\eta(q')$ is different both from $\eta(q)$ and from the vertex contained by the father of q (i.e., $\eta(q') \neq \eta(q)$, $\eta(q') \neq \eta(\textit{father}(q))$). We introduce the set *domination*(t, v) containing all vertices i such that *enter-when*(t, i) is dominated by *enter-when*(t, v). We build this set iteratively by initially setting *domination*$(t, v) = V$ for all $t \in T$, $v \in V$ and, every time a node q is explored, updating it as follows:

$$\textit{domination}\big(t, \eta(q)\big) = \textit{domination}\big(t|\eta(q)\big) \cap \big\{\eta(p),\, p \in \textit{predecessors}(q)\big\}$$

where *predecessors*(q) is the set of predecessors of q. After the construction of the tree with root t, *domination*(t, v) contains all (and only) the vertices v' such that $P_c(t, v) < P_c(t, v')$. This is because, to reach t from v by $d(t)$ turns the patroller must always pass though $v' \in \textit{domination}(t, v)$ and therefore, by Markov chains with perturbation, $P_c(t, v) = P_c(t, v') \cdot \phi < P_c(t, v')$ with $\phi < 1$. Thus, conditions (i) and (ii) of Theorem 5.6 being satisfied, *enter-when*(t, v') is (weakly) dominated by *enter-when*(t, v).

Using the trees of paths we identify dominations within the scope of individual targets. However, dominations can exist also between actions involving different targets. To find them, we set:

$$\textit{tabu}(t) = \big\{v \in V \text{ s.t. } \exists t',\, t \in \textit{domination}\big(t', v\big),\ u_{\mathbf{a}}(\textit{penetration-}t) \leqslant u_{\mathbf{a}}\big(\textit{penetration-}t'\big)\big\}$$

for all $t \in T$. *tabu*(t) contains all the vertices v such that there exists a pair $t' \in T$, $t' \neq t$, $v' \in V$ with $u_{\mathbf{a}}(\textit{penetration-}t) \leqslant u_{\mathbf{a}}(\textit{penetration-}t')$ and $P_c(t, v) > P_c(t', v')$ and then, conditions (i) and (ii) of Theorem 5.6 being satisfied, *enter-when*(t, v) is dominated by *enter-when*(t', v'). We set

$$\textit{nondominated}(t) = V \setminus \left\{\bigcup_{v \in V} \textit{domination}(t, v) \cup \textit{tabu}(t)\right\}$$

for all $t \in T$. All (and only) the actions *enter-when*(t, i) such that $i \in \textit{nondominated}(t)$ are not dominated. We state the following theorem, whose proof is trivial due to the construction of the algorithm.

Algorithm 5: INTRUDER_DOMINATION

```
for each t ∈ T do
    tabu(t) = {}
    for each v ∈ V do
        domination(t, v) = V
    EXPAND(t, t, {t}, 0)
for each t ∈ T do
    tabu(t) = {v ∈ V | ∀t ∃t', t ∈ domination(t', v), u_a(penetration-t) ⩽ u_a(penetration-t')}
    nondominated(t) = V \ {⋃_{v∈V\{t}} domination(t, v) ∪ tabu(t)}
```

Algorithm 6: EXPAND($v, t, B, depth$)

```
1 N = {f | father(v) ≠ η(f) ≠ v,  a(η(f), v) = 1}
2 for each f ∈ N do
3     domination(t, η(f)) = domination(t, η(f)) ∩ η(B)
4 if depth < d(t) then
5     for each f ∈ N do
6         EXPAND(f, t, {B ∪ f}, depth + 1)
```

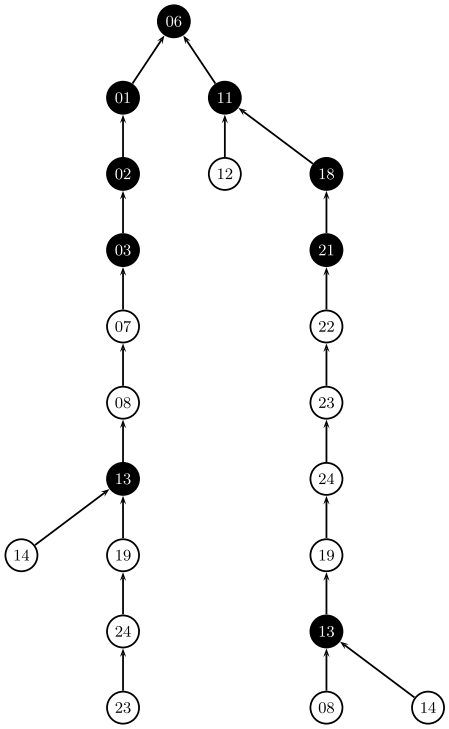


Fig. 8. Search tree for finding dominated actions for target 06 of Fig. 7, white nodes constitute the *nondominated*(06) set.

Theorem 5.7. *Algorithms* 5 *and* 6 *are sound and complete.*

The worst-case computational complexity is $O(|T| \cdot b^{\max_t\{d(t)\}})$, where b is largest outdegree of the vertices. Although the complexity is exponential in $\max_t\{d(t)\}$, in practice the computational time is negligible even for large patrolling settings, as we will show in Section 7.

Example 5.8. In Fig. 8, black nodes denote vertices i such that actions *enter-when*$(06, i)$ are dominated; e.g., *enter-when*$(06, 13)$ is dominated since every occurrence of 13 in the search tree has a node with 14 as child.

Finally, on the basis of the result of Algorithms 5 and 6, we can discard some targets if they appear only in dominated actions.

Corollary 5.9. *Target $t \in T$ such that the actions enter-when(t, i) for all i are dominated will never be entered by the intruder and then can be discarded.*

5.1.3. Iterated dominance

After the removal of defender's and attacker's dominated strategies (in this order), we can only remove some other defender's dominated strategies. We state the following theorem, whose proof is reported in Appendix B.10.

Theorem 5.10. *Assigning a positive probability to $\alpha_{t,t}$ with $t \in T$ is a dominated action if the attacker's action enter-when(t, t) is dominated.*

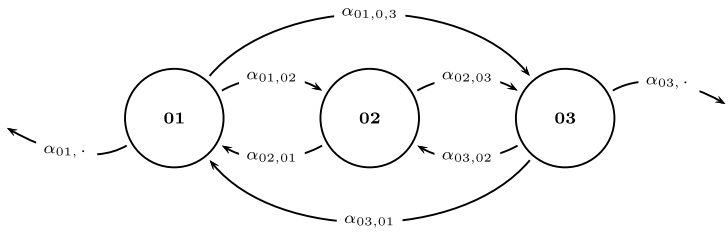


Fig. 9. Abstraction over vertices 01, 03.

No more steps of iterated dominance are possible because, after the removal of the arcs prescribed by Theorem 5.10, the attacker's dominated strategies do not change.[1] We remark that, by resorting to the concept of *never best response* [60], additional attacker's actions can be removed. However, differently from what happens for removal of dominated actions, to remove never best responses we cannot avoid using non-linear mathematical programming. As a result, removing a never best response has the same computational complexity of solving an instance of Formulation 3.9.

5.2. Utility lossless abstractions

Although the removal of dominated strategies greatly reduces the computational time, the resolution of realistic-size games is still hard (as we show in Section 7). An effective technique that has received a lot of attention in the literature to deal with large games is *strategy abstraction* [30,31]. In this section we apply utility lossless abstractions to PSGs. We introduce the definition of abstractions in Section 5.2.1, we define a general class of utility lossless abstractions in Section 5.2.2, and we discuss how they can be applied to a PSG in Section 5.2.3.

5.2.1. Abstraction definition

The basic idea behind abstractions is to group together multiple actions into a single macro action. This allows one to reduce the size of the game. The most interesting kind of abstractions are those without loss of utility (also called *information lossless*), allowing one to find the optimal solution of a game by solving the abstracted one. A number of works on abstractions have been devoted to extensive-form games with imperfect information and, in particular, to poker games [30,31]. However, the application of the seminal result in [30] to a PSG produces a game that is exactly the same size as the original one, because PSGs are general-sum and without chance moves. This pushes us to define *ad hoc* abstractions.

Definition 5.11. An abstraction over a pair of non-adjacent vertices i, j is a pair of defender's macro actions *move-along*(i, j) and *move-along*(j, i) with the following properties[2]:

(a) when the defender makes macro action *move-along*(i, j), the patroller moves from the current vertex i to vertex j along the shortest path visiting turn by turn the vertices composing the path,
(b) the completion of *move-along*(i, j) requires a number of turns equal to the length of the shortest path between i and j,
(c) during the execution of a macro action the defender cannot take other actions,
(d) the attacker can intrude a target during the defender's execution of a macro action.

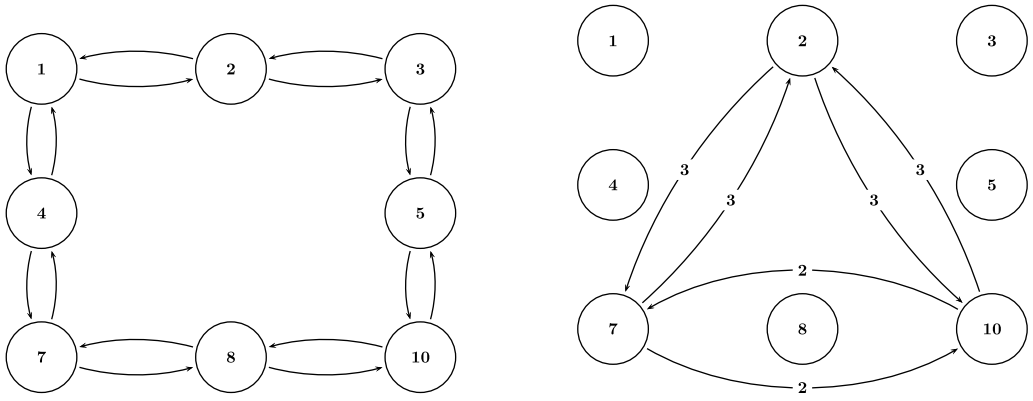
Example 5.12. Consider Fig. 9. By applying an abstraction over vertices 01, 03 we remove the arcs labelled with $\alpha_{01,02}$, $\alpha_{02,01}$, $\alpha_{02,03}$, $\alpha_{03,02}$ (where $\alpha_{i,j}$ corresponds to action *move*(j) from i) and we introduce the arcs labelled with $\alpha_{01,03}$, $\alpha_{03,01}$. When the patroller is in 01 and goes to 03, it will spend two turns, during which it moves from 01 to 02 (first turn) and from 02 to 03 (second turn). When the patroller is in 02, it cannot stop the execution of the current macro action and make another action.

Definition 5.13. An abstraction over G is the result of the application of some abstractions over pairs of vertices. We obtain it by removing from G some disjoint connected subgraphs $G' \subset G$ and introducing in G for each G':

(a) a set of arcs $\{(i, j)\}$, where i, j are vertices in $G \setminus G'$ and both i and j are adjacent to vertices in G' (each arc (i, j) corresponds to a macro action *move-along*(i, j)),
(b) a function $e : V \times V \to \mathbb{N}$ assigning each arc the time needed by the patroller for traversing it.

[1] We notice that, after the removal of attacker's dominated actions, we can discover that some targets will never be entered by the attacker. However, in our case, these targets are on some shortest paths connecting other targets and therefore they cannot be removed.

[2] For the sake of presentation, we consider a situation in which the two vertices are connected by a single shortest path. If this is not the case, we can define a pair of macro actions for each shortest path between two vertices.



(a) Original graph (b) Abstracted graph

Fig. 10. An example of abstraction.

Thus, an abstraction over G involves a number of abstractions over pairs of (non-adjacent) vertices.

Example 5.14. We report an example of abstracted G in Fig. 10.

The main problem to address is the selection of the vertices to be removed such that the obtained abstracted setting preserves the equilibrium strategies.

5.2.2. Defining utility lossless abstractions

When the patroller moves along an abstracted arc (i, j), the attacker can take advantage, because it knows some of the next defender's moves.

Example 5.15. Consider Fig. 9. If the defender decides to move from 01 to 03 and the attacker observes the patroller in 02 after having observed it in 01, then the intruder knows that the patroller will reach 03 at the next turn.

We produce utility lossless abstractions such that the set of attacker's dominated strategies (computed as discussed in Section 5.1.2) is an invariant, namely they are left unchanged by the application of abstractions. This is motivated by the following theorem, whose proof is trivial and then omitted.

Theorem 5.16. *A necessary condition for an* (ex ante) *abstraction to be without utility loss is that the set of attacker's dominated strategies is invariant.*

We provide some necessary conditions for a vertex to be removed without changing the set of attacker's dominated strategies.

Corollary 5.17. *The removal of a vertex i during the application of an abstraction can be without utility loss if*:

(a) *when $i \notin T$, for all $t \in T$, actions enter-when(t, i) are dominated,*
(b) *when $i \in T$, for all $t \in T$, actions enter-when(t, i) are dominated and, for all $j \in V$, actions enter-when(i, j) are dominated.*

It is not enough to assure that the set of attacker's dominated strategies does not change, because we need to assure that solving the abstracted game we can find a strategy no worse than the optimal strategy in the non-abstracted game. We denote by $dom(i, t)$ the set of vertices i' such that *enter-when*(t, i') is not dominated and dominates *enter-when*(t, i) (as calculated in Section 5.1.2). We state the following theorem, whose proof is reported in Appendix B.11.

Theorem 5.18. *Given an abstraction over G, if, for all the abstractions over pairs of vertices i, j and for all vertices k on the shortest path connecting i and j*:

(a) $dist(i, dom(k, t)) \geqslant dist(i, k)$ *and*
(b) $dist(j, dom(k, t)) \geqslant dist(j, k)$

for all targets $t \in T$, then the set of attacker's dominated strategies is invariant and solving the abstracted game gives a strategy as good as the optimal strategy for the original game.

Notice that, after having abstracted a PSG by using our utility lossless abstractions, we can directly solve it without removing other attacker's dominated strategies, these being invariant w.r.t. the non-abstracted game. General abstractions stronger (in terms of the number of removed vertices) than those described in the above theorem can be provided in specific cases, but their computation is not efficient. For instance, abstractions that take the set of never best responses as an invariant can be stronger, but they require the use of non-linear mathematical programming.

5.2.3. Computing utility lossless abstractions

Call $C \subset V$ the set of vertices that satisfy Corollary 5.17. We introduce the binary variables $x_i \in \{0,1\}$ with $i \in C$, where $x_i = 1$ means that vertex i is removed by a utility lossless abstraction and $x_i = 0$ means that it is not. We introduce the integer variables $s_{i,t} \in \{0,\dots,n\}$ with $i \in V$ and $t \in T$ (n is the number of vertices in V), where $s_{i,t}$ gives the distance between vertex i and target t once abstractions have been applied. We call $succ(i,j)$ the set of vertices adjacent to i in the shortest paths connecting i and j.

Formulation 5.19. An abstraction is without utility loss if the following integer linear mathematical programming formulation associated with the abstraction is feasible:

$$s_{i,t} = dist(i,t) \quad \forall i \notin C,\ t \in T \tag{17}$$

$$s_{i,t} \geqslant dist(i,t) \quad \forall i \in C,\ t \in T \tag{18}$$

$$s_{i,t} \leqslant dist(i,t) + nx_i \quad \forall i \in C,\ t \in T \tag{19}$$

$$s_{i,t} \leqslant s_{j,t} + 1 - n(1-x_i) \quad \forall i \in C,\ t \in T,\ j \in succ(i,k),\ k \in dom(i,t) \tag{20}$$

$$s_{i,t} \geqslant s_{j,t} + 1 + n(1-x_i) \quad \forall i \in C,\ t \in T,\ j \in succ(i,k),\ k \in dom(i,t) \tag{21}$$

$$s_{i,t} \leqslant dist(j,t) \quad \forall i \in C,\ t \in T,\ j \in dom(i,t) \tag{22}$$

Constraints (17) force $s_{i,t}$ to be equal to the distance between i and t (for all the non-removable vertices i); constraints (18) force $s_{i,t}$ to be equal to or larger than the distance between i and t (for all the removable vertices i); constraints (19) force $s_{i,t}$ to be equal to the distance between i and t if $x_i = 0$ (for all the removable vertices i); constraints (20) and constraints (21) force $s_{i,t}$ to be equal to $s_{j,t} + 1$ with $j \in succ(i,k)$ where $k \in dom(i,t)$ if $x_i = 1$ (for all the removable vertices i); constraints (22) force $s_{i,t}$ to be not larger than $dist(j,t)$ with $j \in dom(i,t)$.

The above formulation allows us to check whether or not an abstraction is without loss of utility. We are now interested in finding the strongest abstraction that produces the game that requires the minimum computational effort to be solved, namely the game with the minimum number of α variables (arcs). Call ϕ_i the outdegree of vertex i. By removing vertex i from the graph we remove $2\phi_i$ arcs (corresponding to $2\phi_i$ variables α) and we introduce $\phi_i(\phi_i - 1)$ new arcs (corresponding to $\phi_i(\phi_i - 1)$ new variables α).[3] In practice, we can reduce the number of variables only if $\phi_i \leqslant 3$.

Formulation 5.20. The strongest utility lossless abstraction is obtained as the solution of the following integer linear optimization mathematical programming problem:

$$\begin{aligned} &\max \sum_{i \in C} x_i \\ &\quad x_i = 0 \quad \forall i \in C, \quad \phi_i > 3 \\ &\quad \text{constraints (17), (18), (19), (20), (21), (22)} \end{aligned} \tag{23}$$

We call A' the set of arcs of the abstracted game and we represent A' with function $a' : V \times V \to \{0,1\}$. An abstracted game, presenting arbitrary (larger than one) weights, cannot be solved by using Formulation 3.7. An extension of such formulation working with arbitrary weights is presented in Appendix A.4. Notice that the above result implicitly shows that we cannot discard all the non-targets vertices.

5.3. Utility loss abstractions

The application of utility lossless abstractions has the potential to drastically reduce the size of PSGs making their computation more affordable. However, for very large games (especially those containing cycles), utility lossless abstractions produce abstracted games that are still hard to solve. For these games, we can relax the utility lossless constraints to produce reduced games whose solutions are not guaranteed to be optimal for the non-abstracted game.

[3] Rigorously speaking by removing vertex i we remove also a number of variables γ; however, we experimentally observed that the computational effort depends more strongly on the number of variables α than on the number of variables γ.

While utility lossless abstractions produce a game in which the set of attacker's dominated strategies is invariant, with *utility loss* abstractions we produce a game in which a weaker condition holds: each target is not *exposed*. More precisely, we say that a target t is exposed when there is an action *enter-when*(t, x) such that the related capture probability is zero. This happens when the vertex x can be visited by the patroller and $dist(t, x) > d(t)$. Essentially, finding utility lossless abstractions and utility loss abstractions is conceptually similar, the main difference being in the definition of the upper bound on $s_{i,t}$: when abstractions are without utility loss, we need that $s_{i,t}$ be not larger than the maximum distance between $dom(i, t)$ and t, instead, when abstractions are with utility loss, we need that $s_{i,t}$ be not larger than $d(t)$. This kind of utility loss abstractions is the strongest one. Indeed, removing further vertices would introduce a delay that makes a target exposed and therefore would make the associated capture probability equal to zero. All the candidates C that can be removed are all the vertices except the targets.

Formulation 5.21. An abstraction is with utility loss if the following integer linear mathematical program associated with the abstraction is feasible:

$$\text{constraints (17), (18), (19)}$$

$$s_{i,t} \leqslant s_{j,t} + 1 - n(1 - x_i) \quad \forall i \in C,\ t \in T,\ j \in succ(i, t) \tag{24}$$

$$s_{i,t} \geqslant s_{j,t} + 1 + n(1 - x_i) \quad \forall i \in C,\ t \in T,\ j \in succ(i, t) \tag{25}$$

$$s_{i,t} \leqslant dist(i, t) + 1 - n(1 - x_i) \quad \forall i \in C,\ t \in T,\ succ(i, t) = \emptyset,\ \exists k,\ a(i, k) = 1,\ dist(k, t) = dist(i, t) \tag{26}$$

$$s_{i,t} \geqslant dist(i, t) + 1 + n(1 - x_i) \quad \forall i \in C,\ t \in T,\ succ(i, t) = \emptyset,\ \exists k,\ a(i, k) = 1,\ dist(k, t) = dist(i, t) \tag{27}$$

$$s_{i,t} = dist(i, t) \quad \forall i \in C,\ t \in T,\ succ(i, t) = \emptyset,\ \forall k,\ a(i, k) = 1,\ dist(k, t) < dist(i, t) \tag{28}$$

$$s_{i,t} \leqslant d(t) \quad \forall i \in C,\ t \in T \tag{29}$$

Constraints (24), (25), and (29) relax the corresponding (utility lossless) constraints (17), (18), and (22) considering directly target t instead of the set $dom(i, t)$. Constraints (26), (27), and (28) are analogous to constraints (24) and (25), but they are applied when, for a given vertex i and a target t, there is not any successor (i.e., $succ(i, t) = \emptyset$). This happens in the presence of cycles and precisely when i is the farthest vertex from t. Constraints (26) and (27) are applied when there exists a vertex k that is as far as i from t, while constraints (28) are applied when x is the farthest vertex.

As we did for utility lossless abstractions, we search for the strongest abstractions that produce the smallest game.

Formulation 5.22. The strongest utility loss abstraction for a PSG is obtained as the solution of the following integer linear optimization mathematical programming problem:

$$\max \sum_{i \in C,\, \phi_i \leqslant 3} x_i$$

$$\text{constraints (17), (18), (19), (24), (25), (26), (27), (28), (29)}$$

As in the previous section, we call A' the set of arcs of the abstracted games and we represent A' with function $a' : V \times V \to \{0, 1\}$.

Example 5.23. In Fig. 11 we report the setting obtained after the application of the strongest utility loss abstraction on the setting of Fig. 1.

The application of utility loss abstractions produces a reduced game whose attacker's dominated strategies are potentially different from those in the original game. Furthermore, Algorithms 5 and 6 cannot be applied to the abstracted game because they do not consider the possibilities that the distance between vertices is larger than one and that the intruder can enter some target when the patroller is moving from a vertex to another. We present in Appendix A.5 an extension of Algorithms 5 and 6 applicable to graphs generated with utility loss abstractions. Once we have removed attacker's dominated strategies, the computation of the leader–follower equilibrium is based on the same mathematical programming formulation used for utility lossless abstractions (Section 5.2.3) except that, for each non-dominated action *enter-when*(t, v), we use $d(t) - delay(t, v)$ instead of $delay(t, v)$.

6. Summarizing the solving algorithm and addressing inconsistencies

We collect the algorithmic results of previous sections in Section 6.1 and we discuss how inconsistencies can be addressed in Section 6.2.

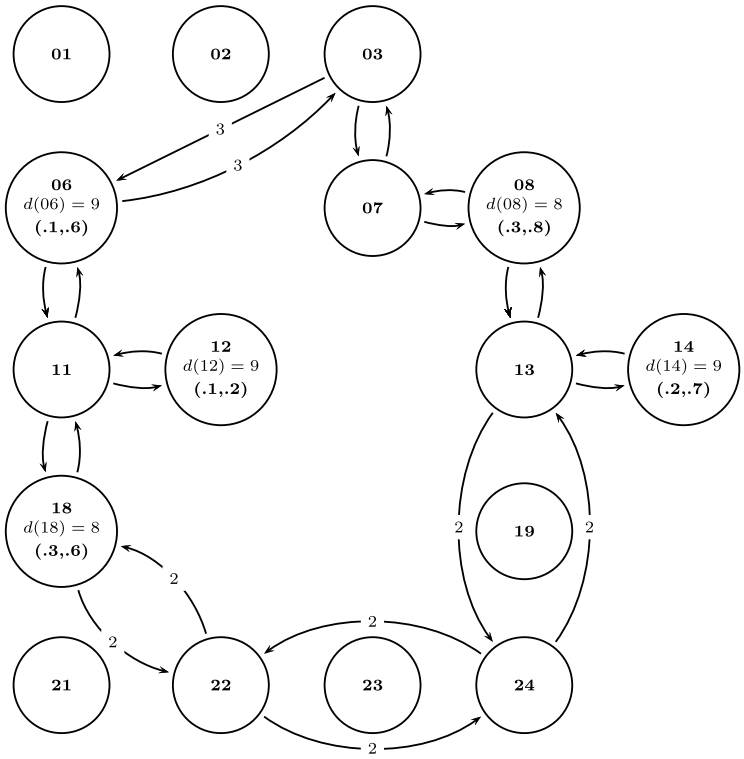


Fig. 11. Utility loss abstraction for the setting of Fig. 1.

6.1. Global solving algorithm

Algorithm 7 summarizes the proposed approach for solving large instances of PSGs. The deterministic strategy is computed (Step 1) by reducing the game as described in Section 4.1 and (Step 2) by solving a CSP as described in Section 4.4. If no deterministic solution exists (Step 5), then a non-deterministic strategy is searched for. The game is reduced (Step 6) as: if lossless abstractions are used, agents' dominated actions are removed as described in Sections 5.1.1, 5.1.2, and 5.1.3 and then abstractions are found as described in Section 5.2; if, instead, lossy abstractions are used, defender's dominated actions are removed as described in Section 5.1.1, then abstractions are found as described in Section 5.3, and finally the intruder's actions are removed as described in Appendix A.5. After the reduction of the game, the leader–follower equilibrium is found (Step 7) by solving the mathematical programs as described in Sections 3.3.2, 3.3.3, and 5.2.3.

Algorithm 7: SOLVING(G)

```
1 reduce G to G'
2 search for a deterministic strategy of G'
3 if G' admits a deterministic strategy then
4 |   return it
5 else
6 |   remove agents' dominated actions and find abstractions
7 |   solve the associated mathematical programming problems
8 |   return it
```

6.2. Addressing inconsistencies

We recall that an inconsistent strategy is characterized by an attacker's best response *enter-when*(t, x) such that x is never visited by the patroller after an infinite number of turns. The problem of checking whether or not a strategy is consistent can be formulated as the problem of checking, for each best response *enter-when*(t, x) and once arcs not covered by the strategy are removed, whether or not there is a connected portion of G including both t and x. If this connected portion exists, then the strategy is consistent. If a strategy is inconsistent, two cases are possible: (i) when the uncovered portion of G does not contain targets, replacing G with its covered portion is safe: it is trivial to see that by solving the obtained reduced game we obtain a non-worse equilibrium strategy for the defender; (ii) when the uncovered portion of G

contains targets, discarding the uncovered portion is not safe: intuitively, the presence of non-patrolled targets can worsen the defender's utility since the intruder can attack them with a guarantee of not being captured. The second case is more complex. An inconsistent strategy must be discarded and we need to recompute it on a sub-portion of the graph. However, simply discarding the uncovered portion of the graph (with the targets it contains) does not provide any guarantee on the solution's optimality. This situation raises the problem of finding the optimal subset of targets to cover. We provide an algorithm to address this problem.

Call $T_c \subseteq T$ the set of targets to cover and $T_{-c} = T \setminus T_c$ the set of targets that are not covered. We call $G(T_c)$ the reduced patrolling setting that comprises only the targets T_c. We obtain it by removing T_{-c} from G's sets of vertices V and targets T and then taking the intersection of all the connected components of the graph in which all the remaining targets T_c are present. Note that for some T_c the setting $G(T_c)$ can be empty, for example when the removal of T_{-c} splits the graph in two connected components each one containing at least one target. Our algorithm enumerates all the possible T_c and computes the leader–follower equilibrium in $G(T_c)$ explicitly considering that the attacker can attack a target in T_{-c} with a capture probability of zero. This is done by introducing in Formulations 3.7 and 3.9 additional actions available to the attacker: attacking targets $t \in T_{-c}$ with a utility equal to $v_{\mathbf{a}}(t)$ independently of the defender's strategy. Finally, we select the consistent strategy that maximizes the defender's expected utility.

Guarantees on solution's optimality provided by this simple algorithm stem from the following lemma, whose proof is reported in Appendix B.12.

Lemma 6.1. *If solving $G(T_c)$ for some T_c returns an inconsistent strategy $\sigma_{\mathbf{d}}$, then the defender's expected utility from $\sigma_{\mathbf{d}}$ is not larger than the defender's expected utility when restricting to the targets covered by $\sigma_{\mathbf{d}}$.*

Therefore, there is at least a consistent patrolling strategy $\sigma'_{\mathbf{d}}$ with $T'_c \subseteq T_c$ such that $\sigma'_{\mathbf{d}}$ is not worse than $\sigma_{\mathbf{d}}$ for the defender. Then, we provide some insight to limit the number of possible sets T_c to be enumerated. We state the following lemma, whose proof is reported in Appendix B.13.

Lemma 6.2. *For any T_c and T'_c with $T'_c \subseteq T_c$, if the optimal strategy on $G(T_c)$ is consistent, then the defender's expected utility of the optimal strategy in $G(T'_c)$ is not better.*

The above lemma shows that, when the resolution of the game with all the targets returns a consistent strategy, solving patrolling settings in which the patroller is limited to patrol a subset of targets does not return a better strategy. We state the following theorem, whose proof is in Appendix B.14.

Theorem 6.3. *Algorithm 8 with $T_c = T$ returns the optimal solution.*

Algorithm 8: REMOVING_INCONSISTENCY(G, T_c)

```
1 solve G(T_c)
2 if the solution is consistent then
3     return the defender's utility
4 else
5     assign u = 0
6     for all t ∈ T_c do
7         u = max{u, GENERAL_SUM _INCONSISTENCY(G, T_c \ {t})}
8     return u
```

7. Experimental evaluation

In this section, we evaluate *scalability* and solution *quality* of our algorithm in patrolling settings comparable with those encountered in real-world scenarios [43]. Experiments have been conducted on a Linux (2.6.24 kernel) machine equipped with an Intel XEON 2.33 GHz CPU, 4 GB RAM, and 4 MB cache. We evaluate our algorithm to find deterministic and non-deterministic patrolling strategies in Section 7.1 and Section 7.2, respectively. In Section 7.3 we evaluate the quality of the non-deterministic solutions.

7.1. Finding a deterministic equilibrium strategy

Without loss of generality, we abstract away from the specific topology of the original patrolling setting and concentrate only on G'. We developed a random generator of graph instances G' with two parameters as input: the number of vertices n (corresponding to targets in the original graph G) and the number of arcs m (corresponding to what we would obtain applying the reduction procedure of Section 4.1 to G). Given n and m, a random connected graph with n vertices is produced, $m - n$ arcs are added and their weights are set to 1. Values $d(k)$ are uniformly drawn from the interval

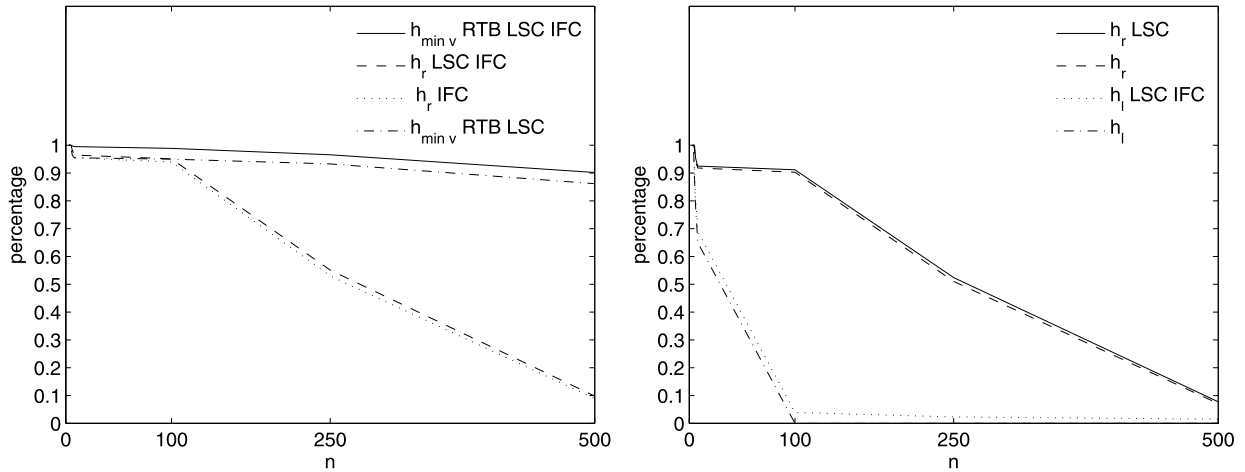


Fig. 12. Percentage of termination for the most significant algorithm configurations for finding deterministic patrolling strategies (for the sake of presentation, we split the configurations on two different subfigures; n is the number of targets).

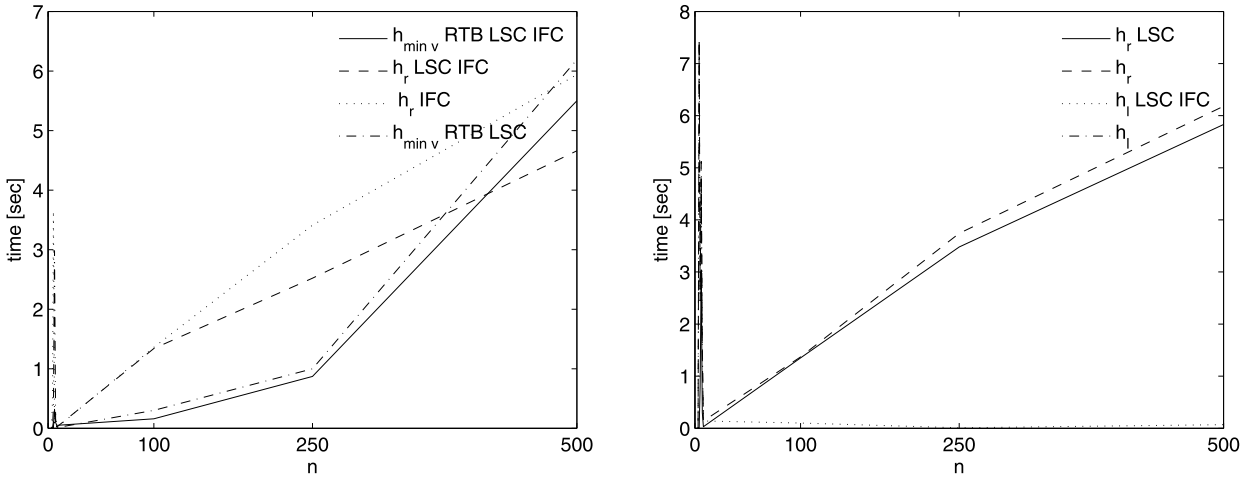


Fig. 13. Computational times for the most significant algorithm configurations for finding deterministic patrolling strategies (for the sake of presentation, we split the configurations on two different subfigures; n is the number of targets).

$[\min_{i,j}\{\overline{w}(i,j)+\overline{w}(j,i)\}, 2n^2 \max_{i,j} \overline{w}(i,j)]$, where $\overline{w}(i,j)$ is the length of the shortest path between vertices i and j. The lower bound of the interval comes from the consideration that settings with $d(k) < \min_{i,j}\{\overline{w}(i,j)+\overline{w}(j,i)\}$ are unfeasible and our algorithm immediately detects it (by IFC). The upper bound is justified by considering that if a problem is feasible, then it will always admit a solution shorter than $2n^2 \max_{i,j}\{\overline{w}(i,j)\}$. The program for generating graphs and those implementing our algorithms have been coded in C.

Since our objective is to find a solution that satisfies all the constraints and not the optimal solution according to a given metric (e.g., minimizing the cycle length), we evaluate: the *termination percentage* (either with a solution or with a failure) of the algorithm within 10 minutes, the *computational time* in the case of termination, the *success percentage* in the case of termination (i.e., the percentage with which a solution has been found).

For each ordering criterion introduced in Section 4.4.3 (i.e., h_l, h_r, $h_{\max a}$, $h_{\min a}$, $h_{\min v}$, $h_{\max d}$, $h_{\min d}$), with and without LSC and IFC, we consider $n \in \{3, 4, 5, 6, 7, 8, 100, 250, 500\}$ and, for each n, we produce 1000 instances of G' with m uniformly drawn from the interval $[n, (n-1)n]$ (if $m < n$ the graph is not connected, if $m > (n-1)n$ at least a pair of vertices is connected by more than one arc).

The most significant experimental results are summarized in Fig. 12 (termination percentages) and Fig. 13 (average computational times), while an exhaustive view is reported as Table 2 in Appendix C. We do not report all the combinations of heuristics for improving efficiency because some of them are not enough significant. All the values are averaged over the 1000 instances. The results on the success percentages appear less significant, being strictly correlated to the termination percentages. Essentially, the lower the termination percentage the higher the success percentage. This is because the algorithm almost always terminates when a solution is found. Proving that an instance does not admit any solution usually requires the algorithm to explore the entire tree and this can rarely happen within 10 minutes.

For all the algorithm configurations, the average computational time keeps reasonably short (few seconds) even with large values of n. On the other hand, the termination percentage varies significantly in different configurations. This behavior resembles that of many constraint programming algorithms, whose termination time is usually either very short (when a solution is found) or the deadline is exceeded. Moreover, the obtained results present outliers that can be detected by observing the maximum computational times in Table 2. Some cases were harder than the average to solve and required an amount of time significantly larger (in practice, these cases both reduce the percentage of termination and increase the computational time). These hard cases represent outliers within the population of instances obtained with the random graph generator. They are typically characterized by tangled topologies or oddly-distributed relative deadlines.

We now comment the performance of the techniques of Section 4.4.3 for improving the efficiency of our algorithm.

The best ordering criterion seems to be $h_{\min v}$ with RTB. The experimental results with $h_{\max a}$, $h_{\min a}$, $h_{\max d}$, $h_{\min d}$ are very similar to those obtained with h_l and then omitted. The criterion $h_{\min v}$ with RTB leads the algorithm to terminate with a percentage close to h_r for small values of n and about 80% larger for large values of n. Instead, h_l provides very bad performance, especially for large values of n, when the algorithm terminates with percentages close to 0%.

The LSC improved stopping increases the termination percentage by a value between 0% and 2%, without distinguishable effects on the computational time. This improvement depends on the configuration of the algorithm since it affects the construction of the search tree. Its adoption increases the percentage of termination without finding any solution.

The IFC criterion increases the termination percentage by a value between 1% and 4%, reducing the computational time (since many non-feasible settings can be recognized before starting the construction of the tree). This improvement does not depend on the configuration of the algorithm since it does not affect the search, working before it. Its adoption increases the percentage of termination without finding any solution.

Hence, the best algorithm configuration appears to be $h_{\min v}$ with RTB, LSC, and IFC. The results are satisfactory: the termination percentage is high also for large settings, even with 500 targets, and the corresponding average computational time, about 5.5 s, is reasonably short. The above results justify our decision to set the threshold at 10 minutes (600 s).

7.2. Simplifying large patrolling security games and finding a randomized equilibrium strategy

Given the different and heterogeneous formulations of the patrolling problem proposed in the literature (and summarized in Table 1), the definition of a data set for experimentation and comparison has not yet been addressed (at least, to the best of our knowledge). Some partial attempts to define experimental settings for non-adversarial patrolling can be found in [59] (used also in [47]) where the authors propose two arbitrary topology maps with about 50/60 vertices. The lack of a suitable data set for our experimental activity pushes us to develop an *ad hoc* data set.

Our data set for adversarial patrolling is partitioned in two parts. The first one is composed of settings with *perimeter* topologies (in which we further distinguish between *open* and *closed* topologies), while the second part is composed of settings with *arbitrary* topologies. The settings with arbitrary topologies have been obtained both by introducing targets in the setting presented in [59] and by producing new patrolling settings inspired by environments in RADISH (a repository containing data sets of environmental acquisitions performed with real robots [36]). We characterize the patrolling settings w.r.t. the number n of vertices and the *density* δ of targets, representing the percentage of targets over vertices (i.e., $\delta = |T|/n$).

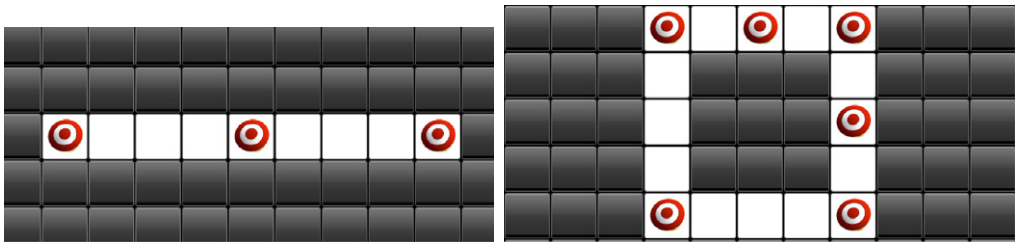
We evaluated and compared multiple configurations of our algorithm that differ in the efforts devoted to the pre-processing phase. More precisely, given a PSG instance, we consider: basic algorithm (*basic*): we plainly compute the optimal strategy as described in Section 3.3 on the original setting without exploiting any kind of reduction; removal of dominated strategies (*dom*): we apply the algorithms described in Section 5.1 to reduce the game, then we solve it with the basic algorithm; utility lossless abstraction (*lossless*): we remove the players' dominated strategies, we apply the strongest utility lossless abstraction, as described in Section 5.2, and finally we solve it with the basic algorithm; utility loss abstraction (*lossy*): we apply the strongest utility loss abstraction as described in Section 5.3, we remove the intruder's dominated strategies from the obtained game, and finally we solve it with the basic algorithm.

We imposed a time deadline: one hour for the zero-sum settings and 24 hours for the general-sum ones. We coded our algorithm in MATLAB and we formulated all the mathematical programming problems with AMPL [25]. We used CPLEX [37] and SNOPT [61] for solving the linear and non-linear mathematical programs, respectively.

7.2.1. Open perimetral settings

We generated our settings with the following features (see Fig. 14(a) for an example): $n \in \{10, \ldots, 200\}$ and $\delta \in \{10\%, \ldots, 50\%\}$; targets are randomly selected among the vertices with the constraint that the two extreme vertices must be targets; for each target t a random value $v_{\mathbf{d}}(t)$ is chosen under the global constraint $\sum_{t \in T} v_{\mathbf{d}}(t) = 1$ and penetration times $d(t)$ are independently drawn from the interval $\{\overline{D}_t, \overline{D}_t + 1, \ldots, 2\overline{D}_t - 1\}$ where $\overline{D}_t$ is the maximum distance of t from an extreme vertex (a penetration time shorter than $\overline{D}_t$ could make a target to be exposed, while with penetration times longer than $2\overline{D}_t - 1$ deterministic equilibrium strategies exist). For general-sum settings, for each target t a random value in $[0, 1]$ is chosen for $v_{\mathbf{a}}(t)$. For each pair of values (n, δ) we generated 5 patrolling settings and we analyzed the average values.

The removal of dominated actions allows one to discard about 95% of the attacker's actions on average and up to 99% (details can be found in Table 3, see Appendix C). The number of non-dominated attacker's actions happens to be very small



(a) Open (b) Closed

Fig. 14. Example of open and closed perimetral settings.

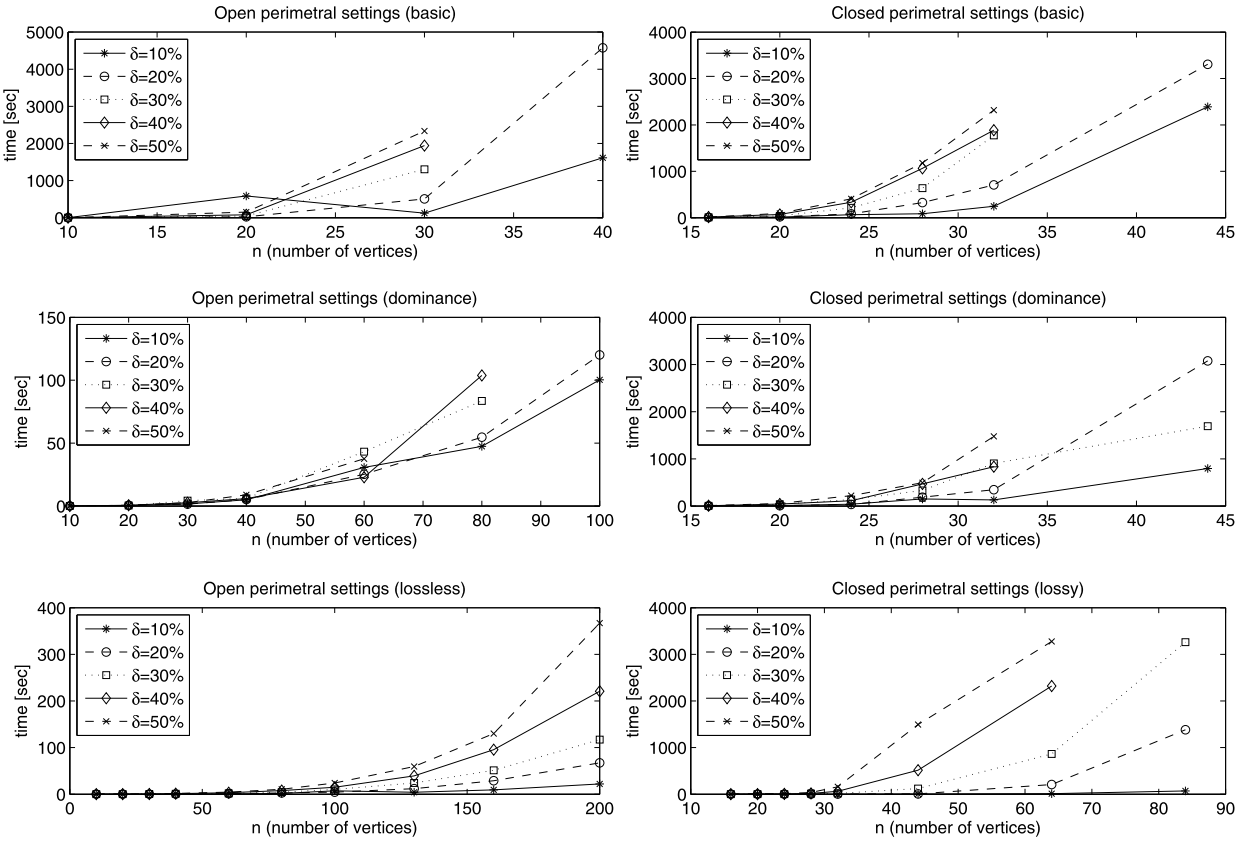


Fig. 15. Computational times for different algorithm configurations in open and closed perimetral settings.

(no more than 12) even with large settings (with more than 10^4 vertices). This is essentially due to two reasons. First, each attacker's action *enter-when*(t, h) where h is not an extreme vertex is dominated by *enter-when*(t, h') where h' is extreme. Second, a large number of targets in non-extreme vertices will never be attacked because there is some other target with a larger value and presenting a strictly smaller capture probability.

The application of lossless abstractions allows one to generate abstracted graphs discarding, after the removal of dominated actions, about 70% of the vertices on average and up to 93% (see Table 4 in Appendix C). This is essentially due to the very low number of non-dominated actions. Since there are few non-dominated actions and each of them imposes a set of constraints over the computation of the abstractions, the associated optimization problem is weakly constrained and many vertices can be discarded.

Graphs in Fig. 15 (left-hand side) show how the computational time (of the terminated executions) varies w.r.t. n and δ for the zero-sum case. With general-sum instances the results are similar. To give a general idea, the average computational time for a general-sum instance is roughly the number of (non-dominated) attacker's actions multiplied by $\pm 10\%$ of the computational time of a zero-sum instance. The detailed data for zero-sum and general-sum instances can be found in Table 5, see Appendix C.

For all the configurations, the computational times rise exponentially as δ^n. The *basic* configuration (top left graph of Fig. 15) requires a large amount of computational time even with small settings and it is applicable only up to $n = 40$ vertices and $\delta = 10\%$ of targets. Going beyond this limit is impractical. The *dom* configuration (middle left graph of Fig. 15) solves much larger settings thanks to a significant reduction of the defender's actions. In this case, the computational limit is for $n > 100$ and it is due to the excessively large number of vertices and therefore of defender's actions. The percentage of the pre-processing time over the total computational time with *dom* is 4% on average (the max is 18% and the min is 1%). The *lossless* configuration (bottom left graph of Fig. 15) solves larger instances composed up to 200 vertices and 50% of targets thanks to a reduction of the actions of the attacker and the defender. The percentage of the pre-processing time over the total computational time with *lossless* is 76% on average (the max is 99% and the min is 9%). Given that realistically large open perimetral settings (more than 200 vertices and 100 targets) are solvable with the *lossless* configuration, we did not apply the *lossy* configuration.

7.2.2. Closed perimetral settings

Closed perimetral settings are generated as squares with edges whose length is f vertices such that (see Fig. 14(b) for an example): $n \in \{10, \ldots, 200\}$ and $\delta \in \{10\%, \ldots, 50\%\}$. Targets are randomly selected among the vertices with the constraints that the four corners are targets and that the graph is not reducible, by removal of patroller's dominated actions, to an open setting; penetration times $d(t)$ are independently drawn from the interval $\{2f, 2f + 1, \ldots, n - 1\}$ (a penetration time shorter than $2f$ could make a target to be exposed, while with penetration times longer than $n-1$ deterministic equilibrium strategies exist). The agents' values over targets are generated as for the open perimetral settings. For each pair of values (n, δ) we generated 5 patrolling settings and we analyzed the average values.

The removal of dominated actions applied to the original settings discards about 41% of the attacker's actions on average and up to 50% (see Table 6 in Appendix C). The number of removed actions is much lower than with open perimeters. This is essentially because there are not extreme vertices.

The lossless abstractions do not remove any vertex. The set of removable vertices is empty because every vertex appears (as t or h) in at least one non-dominated action *enter-when*(t, h). The lossy abstraction allows one to generate abstracted graphs discarding about 50% of the vertices on average and up to 70% (see Table 7 in Appendix C). The application of removal of dominated actions applied to lossy abstraction graphs allows one to discard up to 50% of the actions (see again Table 6).

Graphs in Fig. 15 (right-hand side) show how the computational time (of the terminated executions) varies w.r.t. n and δ for the zero-sum case. The computational time of a general-sum instance is roughly ($\pm 9\%$) the number of (non-dominated) attacker's actions multiplied by the computational time of a zero-sum instance. The detailed data are in Table 8, Appendix C.

Closed perimetral settings are more difficult w.r.t. open ones. The *basic* configuration (top right graph of Fig. 15) ran out of memory with $n = 44$ vertices and $\delta = 20\%$ of targets and only slight improvements are obtained when enabling the removal of dominated strategies. With the *dom* configuration (middle right graph of Fig. 15), the limit over the setting size is only improved to 44 vertices and 30% targets. The percentage of pre-processing time w.r.t. the total computational time with *dom* is $< 1\%$ on average (the max is 2%, and the min is $< 1\%$). The *lossless* configuration provides the same results of *dom*. The *lossy* configuration (bottom right graph of Fig. 15) allows one to solve large instances composed up to 84 vertices and 50% of targets. The percentage of the pre-processing time over the total computational time with *lossy* is 53% on average (max is 99% and min is 12%).

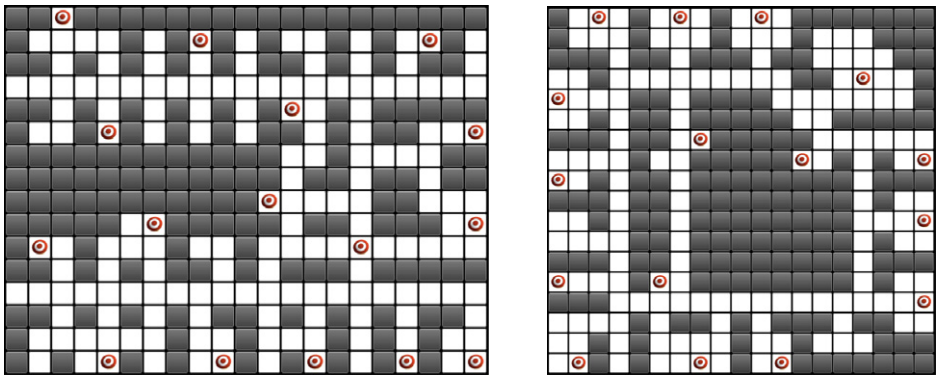
Notice that in the specific case in which all the vertices are targets and all the targets present the same value (for both agents) and the same penetration time, settings can be solved by applying the algorithm presented in [3]. In this case, such algorithm outperforms ours, requiring polynomial time.

7.2.3. Arbitrary settings

We developed a software tool [14] to compose patrolling settings and we generated arbitrary settings. In addition to the settings proposed in [59], we considered a subset of the indoor environments from RADISH repository: albert-b-laser, austin_ aces3, cmu_nsh_level_a, DLR-Spatial_Cognition, fr079, kwing_wld, intel_oregon, mit-csail-3rd-floor, sdr_site_b, stanford-gates1, and ubremen_cartesium. Fig. 16 shows two examples.

For every topology, we manually reproduced several bidimensional grids representing the map for different n and δ where $n \in \{50, \ldots, 166\}$ and $\delta \in \{5\%, \ldots, 30\%\}$. Broadly speaking, with large values of n, each cell is associated to a small area of the environment, while, with smaller n, each cell represents a larger part of the environment. Target cells are randomly selected for different values of δ and penetration times $d(t)$ are randomly chosen in the interval $\{\overline{D}_t, \overline{D}_t + 1, \ldots, 2\overline{D}_t - 1\}$ where $\overline{D}_t$ is the maximum distance of t from any vertex. The target values are generated as in the open and closed perimetral settings. For each topology and each pair of values (n, δ) we generated 5 patrolling settings and we analyzed the average values. In this case, we did not consider any time threshold in experiments, therefore memory was the only limited computational resource.

The performance of our reduction techniques is between that of the open and that of the closed perimetral settings. The removal of dominated actions applied to the original setting eliminates about 58% of the attacker's actions on average and up to 83% (see Table 9 in Appendix C). The application of lossless abstractions generates abstracted graphs discarding, after the removal of dominated actions, about 20% of the vertices on average and up to 35% (see Table 10 in Appendix C). The



(a) DLR–Spatial_Cognition (b) intel_oregon

Fig. 16. Two examples of arbitrary patrolling settings (white cells are associated to vertices, targets are denoted with circles) with 166 vertices and 10% targets.

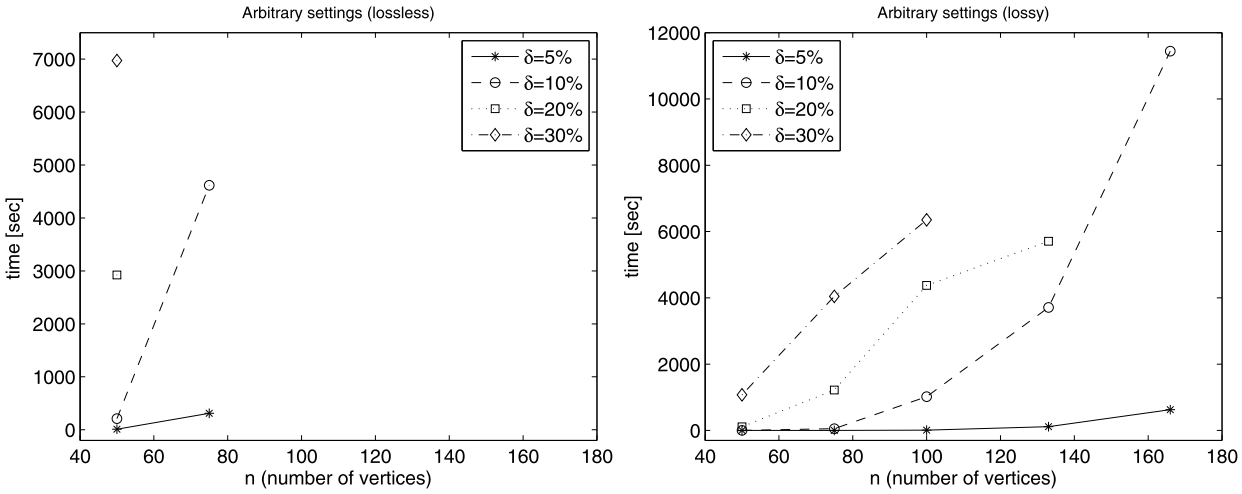


Fig. 17. Computational times for different algorithm configurations in arbitrary settings.

lossy abstraction generates abstracted graphs discarding about 55% of the vertices on average and up to 76% (Table 10). The application of removal of dominated actions applied to lossy abstraction graphs discards up to 75% of the actions (Table 9).

Graphs in Fig. 17 show how the computational time varies w.r.t. n and δ for the zero-sum case. Results are reported for terminated executions (those which did not exceed the memory limit) and for all the obtained strategies no inconsistencies were observed. With general-sum instances the results are similar: the computational time of a general-sum instance is roughly ($\pm 10\%$) the number of (non-dominated) attacker's actions multiplied by the computational time of a zero-sum instance. The detailed data for zero-sum and general-sum instances can be found in Table 11 in Appendix C.

We solved the game instances only with the *lossless* and *lossy* configurations (results with *dom* are worse than those with *lossless*, while with *basic* all the instances required more than 4 GB RAM). As it can be seen from Table 11, arbitrary settings turned out to be less hard than closed perimetral ones. By employing the *lossless* configuration we encountered a limit with $n = 75$ vertices (cells) with $\delta = 10\%$ of targets. The *lossy* configuration allowed us to solve instances up to 166 cells with 10% of targets. The computational time spent for pre-processing for both *lossless* and *lossy* configurations is similar to that found for closed perimetral settings.

As a last experiment, we relaxed the penetration times in the above instances such that they admit a pure strategy equilibrium for which the intruder acts *stay-out*. We solved these instances with the *lossless* and *lossy* configurations. The average computational times are close to those reported in Table 11. They are about 10^3 larger than those obtained with the algorithm specific for deterministic strategies we described in Section 4. In addition, the solutions returned by the non-deterministic algorithm (being first-order Markovian) do not assure the patroller to capture the intruder with a probability of one. These findings justify our approach of looking for deterministic strategies with a specialized algorithm and not with the general algorithm for non-deterministic strategies.

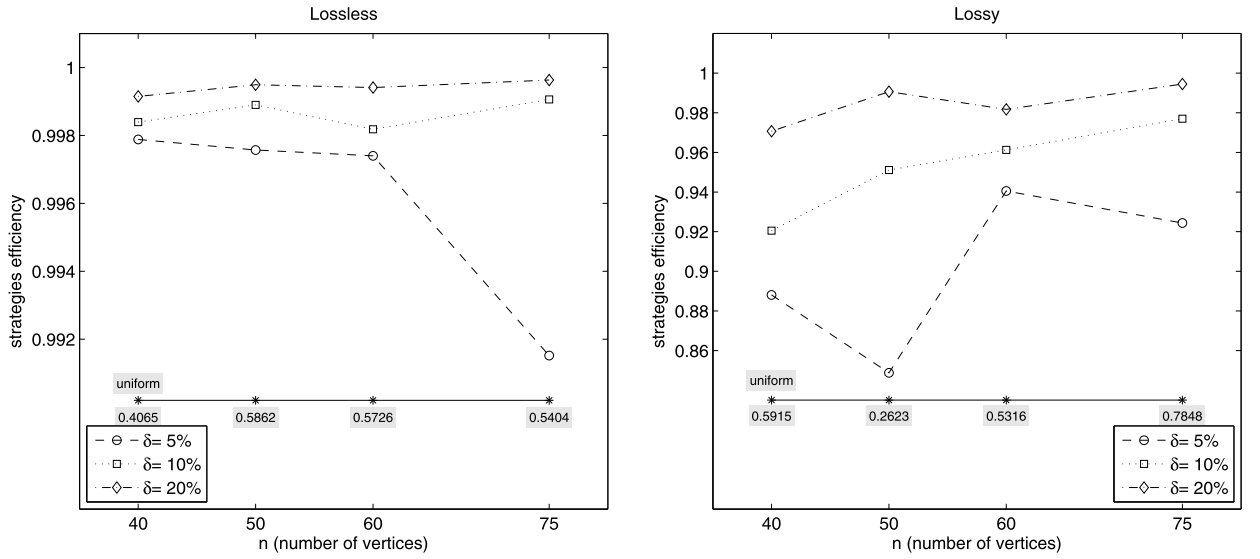


Fig. 18. Efficiency of first-order Markovian strategies (with lossless and lossy abstractions) for different settings when there is a non-first-order Markovian deterministic strategy. Grey boxes report the efficiency for the baseline uniform strategy.

7.3. Solution quality evaluation

We evaluate the quality of the solutions produced by our algorithms in terms of efficiency as discussed in Section 3.3.4: we evaluate the efficiency of first-order Markovian strategies w.r.t. higher-order Markovian strategies and we evaluate the efficiency of solutions produced with lossy abstractions w.r.t. solutions produced with lossless abstractions.

For all the previous settings (open, closed, and arbitrary topologies), the value u^* of the corresponding optimal (high-order Markovian) strategy is not known. In this case, as discussed in Section 3.3.4, we can calculate a lower bound on efficiency considering $\sum_i v_{\mathbf{d}}(i)$ instead of u^*. With respect to the results presented in the previous sections, the average value of this lower bound on efficiency is 0.97 for the lossless configurations (max is 0.99, min is 0.95) while 0.92 for the lossy ones (max is 0.99, min is 0.81).

In order to obtain a more accurate evaluation of solutions' quality, we considered settings which admit a deterministic equilibrium strategy. Hence, for these settings, u^* is known in advance and is equal to the sum of all the targets' values. In these settings, efficiencies can be computed exactly without resorting to a lower bound. The arbitrary graphs we generated in this phase have the following features: $n \in \{40, \ldots, 75\}$ and $\delta \in \{5\%, \ldots, 20\%\}$; targets are randomly selected and their value is chosen as in the open and closed perimetral settings. The penetration times are the smallest values such as there exists a non-first-order Markovian deterministic strategy, but there is not any first-order Markovian deterministic strategy. This represents the worst case for the non-deterministic first-order Markovian strategies. (Obviously, relaxing the penetration times, first-order Markovian non-deterministic strategies perform better.) For each pair of values (n, δ) we generated 10 patrolling settings and we analyzed the average values.

Graphs in Fig. 18 show the solutions' efficiency for first-order Markovian strategies with lossless and lossy abstractions for zero-sum instances. Strategies are compared against a baseline case whose efficiency is reported in grey boxes. The baseline strategy is a Markovian strategy that assigns a uniform probability of movement to adjacent vertices on the abstracted (lossless and lossy, respectively) graph and in settings with a small target density ($\delta = 5\%$). A remarkable improvement in efficiency is generally obtained when passing from the baseline case to a Markovian equilibrium strategy. Results obtained for general-sum instances are similar. The detailed data are in Table 12 in Appendix C.

Lossless first-order Markovian strategies are very effective, providing at least 99% of the utility provided by the deterministic strategy. The expected utility strictly increases in δ, the defender preserving more targets.

The employment of lossy first-order Markovian strategies is very satisfactory, providing at least 85% of the utility provided by the deterministic strategy. As in the lossless case, the expected utility strictly increases in δ.

We additionally compared the quality of the solutions with lossy abstractions w.r.t. lossless abstractions in all the settings used in Sections 7.2.2 and 7.2.3. In these cases the loss in efficiency is not larger than 5%. This, being an average case, is consistent with the worst case discussed above.

Finally, a remark about inconsistencies is worth. The absolute utility loss of a consistent strategy w.r.t. the optimal (high-order Markovian) strategy is upper bounded by $\min_{t \in T} v(t) P_c(t, h)$ where *enter-when*(t, h) is a valid best response for the intruder. This happens because at the equilibrium (excluding some very particular topologies) the intruder is likely induced to attack the covered target with the lowest value for the defender. This guarantees some relatively limited (w.r.t. targets' values) absolute utility loss. When inconsistencies are present, the set of covered targets can change, resulting in a larger

minimum value. Thus, the upper bound on absolute utility loss can be large with a consequent arbitrary degradation of the solution's quality.

8. Conclusions and future works

Security games constitute a class of games that are receiving an increasing attention in artificial intelligence and are used in several security applications. They are characterized by two players, a defender and an attacker, and by a set of targets with some values. The attacker wants to intrude a target, while the defender wants to protect the targets. Both attacker and defender can deploy some resources to accomplish their goals. Security games are usually modeled as leader–follower games where the defender can commit to a strategy and where the underlying games are in normal form.

In this paper, we studied security games by taking into account the situation in which the attacker can observe the realization of the strategy of the defender and decide, on the basis of this observation, when to attack without being subject to any temporal deadline. This option, if available, will be always exploited by the attacker, which could gain a larger utility. We extended the currently available models introducing an underlying game in extensive form and with infinite horizon. Since the "natural" application of this model is to mobile robot patrolling, we called *patrolling security games* (PSGs) this new game class. Our model allows one to study adversarial patrolling problems where the topology of the area to be patrolled is represented with an arbitrary graph with targets. We limited our study to the case in which each agent has a single resource: an intruder and a patroller for the attacker and the defender, respectively. The main contribution of our work has been the development of algorithmic techniques for the computation of the leader–follower equilibrium in large instances of PSGs.

Differently from the state of the art in security games, in our setting the leader's commitment is directly in behavior strategies and the problem to compute a leader–follower equilibrium can be formulated as a bilinear mathematical programming problem. The non-linearity is due to the presence of Markovian constraints introduced for dealing with infinite horizon. This makes the computation of the equilibrium a hard problem to solve even for small settings. To get around this limitation we developed specific techniques to find an equilibrium in pure strategies, when possible, and to reduce the size of the game instances, when we need to resort to mixed strategies.

The computation of an equilibrium in pure strategies for PSGs can be formulated as a variant of the TSP. Although it is NP-complete, we showed that an efficient algorithm can be designed by resorting to constraint programming. The computation of a mixed strategy equilibrium is much harder and is tractable in practice only in the case of first-order Markovian strategies. We designed reduction techniques based on the removal of dominated strategies and on utility lossless and utility loss abstractions (these techniques constitute an original contribution to security games). In this way, we drastically reduce the size of the game and can solve large instances. Furthermore, we showed that, in some cases, first-order Markovian strategies produce solutions whose quality is comparable with that of optimal (high-order) solutions.

Several issues are worth further investigation. The first one is the improvement of our model and techniques for the specific patrolling problem we studied. The model could be extended along the following directions: the exploitation of multiple resources for the attacker and the defender (preliminary results are reported in [15]), the refinement of the movement models of the intruder and of the patroller, and the refinement of the patroller's sensing capabilities. Results along this direction have been presented in [13], where the intruder is refined by introducing a more realistic movement model and restricting its observation capabilities. Moreover, it would be interesting to study new algorithms to generate abstractions that keep into account also the defender's expected utility [10], thus searching for the best abstraction for a given trade-off between computational time and expected utility. The second issue for future work is the application of the PSG model to novel applications. We are currently exploring two applications: the patrolling by active mobile cameras (preliminary results are reported in [16]) and computer security. A third issue is the employment of some of the techniques proposed in this paper to problems different from PSGs. For instance, our abstractions could be used for general security games on graphs to speed up the generation of optimal schedules preserving the expected utility.

Appendix A. Extensions and refinements

A.1. Formulation with arbitrary l

We provide a generalization of the formulation presented in Section 3.3.1 to compute intruder capture probabilities when the history length l is arbitrary. Let us introduce some notation. We denote by $H(l)$ the space of all the possible histories h with length l whose first and last vertices are h^1 and h^l, respectively. With a slight overload of notation, we denote with h^{-1} and h^{-l} the histories obtained from h by removing its first and last vertices respectively. We introduce the notion of adjacent histories. Given a history h the set of all its adjacent histories is $AH(h) = \{h_x \in H(l),\ h^{-1} = h_x^{-l}\}$. For example, given the environment of Fig. 1 and the history of length 5, $h = \langle 06, 01, 02, 03, 07\rangle$, a history adjacent to h is $h_x = \langle 01, 02, 03, 07, 08\rangle$. The general formulation is the following:

$$\alpha_{h,j} \geqslant 0 \quad \forall h \in H(l),\ j \in V \tag{30}$$

$$\sum_{j \in V} \alpha_{h,j} = 1 \quad \forall h \in H(l) \tag{31}$$

$$\alpha_{h,j} \leqslant a\big(h^l, j\big) \quad \forall h \in H(l),\ j \in V \tag{32}$$

$$\gamma^{1,t}_{h_1,h_2} = \alpha_{h_1,h_2^l} \quad \forall t \in T,\ h_2 \in AH(h_1), \tag{33}$$

$$\gamma^{w,t}_{h_1,h_2} = \sum_{\substack{h_x \in AH(h_2) \\ t \notin h_x}} \big(\gamma^{w-1,t}_{h_1,h_x} \alpha_{h_x,h_2^l}\big) \quad \forall w \in \{2,\ldots,d(t)\},\ t \in T,\ h_1,h_2 \in H(l),\ h_2^l \neq t, \tag{34}$$

$$P_c(t,h_1) = 1 - \sum_{\substack{h_x \in H(l) \\ t \notin h_x}} \gamma^{d(t),t}_{h_1,h_x} \quad \forall t \in T,\ h_1 \in H(l) \tag{35}$$

The meaning of the constraints is similar to that of the corresponding constraints of Section 3.3.1.

A.2. Reducing the number of variables and constraints

Constraints (1)–(6) can be rewritten in a more efficient way for reducing the number of variables and constraints. We start by giving the intuition with an example.

Example A.1. Consider *enter-when*(14, 12) in Fig. 1. Focus on the event in which the patroller starting from 12 reaches 12 after 2 turns without passing through 14 and call $\gamma^{2,14}_{12,12}$ the probability associated with it. If this event happens, then the probability with which the patroller reaches 14 by $d(14) = 9$ turns is zero, because the distance between 12 and 14 is 9 and only 7 turns are left, and therefore the intruder cannot be captured. As a result, in the computation of the capture probability related to *enter-when*(14, 12), we can safely discard all the events following the one associated with $\gamma^{2,14}_{12,12}$.

Consider action *enter-when*(t,h) with $l = 1$: when the attack to t is started, the patroller is in vertex h and, in order to capture the intruder, it has to visit target t at least once in the following $d(t)$ turns. In such scenario, a sufficient condition for a successful intrusion is the following: if after some turns, say ρ, the patroller has not yet visited t and occupies a vertex i such that $\rho + dist(i,t) > d(t)$, then the intrusion has success, where $dist(i,j)$ is the shortest distance between two vertices i and j. In other words, if the realization of the patroller's path drove it in a vertex from which t cannot be reached by its penetration time, the intruder cannot be captured at all. We can easily determine the minimum value of ρ for a target t when the patroller occupies a generic vertex i as $\rho(i,t) = d(t) - dist(i,t) + 1$. We can now reduce the number of variables of the form $\gamma^{w,t}_{h,j}$ that are included in the computation of $P_c(t,h)$ by setting $\rho(j,t)$ as the upper bound of index w. To do this, constraints (5) and (6) are replaced with the following ones:

$$\gamma^{w,t}_{i,j} = \sum_{\substack{x \in V \setminus \{t\} \\ w \leqslant \rho(x,t)}} \big(\gamma^{w-1,t}_{i,x} \alpha_{x,j}\big), \quad t \in T,\ i,j \in V,\ j \neq t,\ \forall w \in \{2,\ldots,\rho(j,t)\} \tag{36}$$

$$P_c(t,h) = 1 - \Bigg(\sum_{j \in V \setminus \{t\}} \gamma^{\rho(j,t),t}_{h,j} + \sum_{j \in V \setminus \{t\}} \sum_{w \leqslant \rho(j,t)-1} \gamma^{w,t}_{h,j} \sum_{\substack{x \in V \setminus \{t\} \\ w \geqslant \rho(x,t)}} \alpha_{j,x} \Bigg) \quad \forall t \in T,\ h \in V \tag{37}$$

Constraints (36) are the same as constraints (5) with the addition of the upper bound $\rho(j,t)$ on the w index of each $\gamma^{w,t}_{i,j}$ variable. The term enclosed between parentheses in constraints (37) is the success probability of an action *enter-when*(t,i). Its first addendum accounts for all the path realizations that start from i and end in a vertex j *exactly* after $\rho(j,t)$ turns. The second addendum accounts for all the path realizations that end in a vertex x at a turn $w > \rho(x,t)$ given that at turn $w-1$ they visited a vertex j without having reached the corresponding upper bound $\rho(j,t)$. The exact number of variables and constraints eliminated by this refinement strongly depends on the specific instance of the PSG. From our experimental evaluations, we observed that, on average, both the number of variables $\gamma^{w,t}_{i,j}$ and the number of constraints approximately halve.

A.3. Capture probability formulation with intruder movements

In this section we provide an extension that considers a more realistic movement model for the intruder. We assume that, when performing an attack, the intruder follows some path starting from an access vertex and ending in a target. For the sake of simplicity, we denote with ℓ the length of a path and we assume that the intruder can disappear from a target after a number of turns equal to the penetration time of that target. (Modeling situations in which the intruder, once completed an intrusion, escapes from the environment following another path is an easy further extension of the results presented in this section.) We call P the set of all possible paths and we denote with $p \in P$ a single path whose ith vertex is p^i. The actions available to the intruder now are defined as *enter-when*(p,i), namely attack following path p as soon as the patroller is in vertex i. Performing such action, the intruder will spend ℓ turns to cover the path and $d(p^\ell)$ turns to

complete the intrusion in the target and, therefore, it will be detectable for $\ell + d(p^\ell)$ turns. Assuming that $p^w = p^\ell$ if $w \geqslant \ell$, the capture probabilities can be computed by replacing constraints (4)–(6) of Section 3.3.1 with the following ones:

$$\gamma_{i,j}^{1,p} = \alpha_{i,j} \quad \forall p \in P,\ i \in V,\ j \in V - \{p^1\} \tag{38}$$

$$\gamma_{i,j}^{w,p} = \sum_{x \in V - \{p^{w-1}\}} \left(\gamma_{i,x}^{w-1,p} \alpha_{x,j}\right) \quad \forall w \in \{2, \ldots, \ell + d(p^\ell)\},\ p \in P,\ i, j \in V,\ j \neq p^w \tag{39}$$

$$P_c(p, i) = \sum_{j \in V - \{p^\ell\}} \gamma_{i,j}^{\ell + d(p^\ell), p} \quad \forall i \in V,\ p \in P \tag{40}$$

A.4. Capture probability formulation for graphs with arbitrary weights

In order to compute the intruder capture probabilities when graphs have edges with arbitrary weights, we need the following constraints that capture the possibility that traversing arcs can require more than one turn:

$$\alpha_{i,j} \leqslant a'(i, j) \quad \forall i, j \in V \tag{41}$$

$$\gamma_{i,j}^{e(i,j),t} = \alpha_{i,j} \quad \forall t \in T,\ i, j \in V,\ j \neq t \tag{42}$$

$$\gamma_{i,j}^{w,t} = \sum_{\substack{x \in V \setminus \{t\} \\ w \leqslant \rho(x,t) \\ w \geqslant e(i,x) + e(x,j)}} \left(\gamma_{i,x}^{w - e(x,j),t} \alpha_{x,j}\right) \quad \forall w \in \{2, \ldots, \rho(j,t)\},\ t \in T,\ i, j \in V,\ j \neq t \tag{43}$$

$$P_c(t, i) = 1 - \sum_{j \in V \setminus \{t\}} \gamma_{i,j}^{\rho(j,t),t} - \sum_{j \in V \setminus \{t\}} \sum_{w \leqslant \rho(j,t) - 1} \gamma_{i,j}^{w,t} \sum_{\substack{x \in V \setminus \{t\} \\ w \geqslant e(i,j) + \rho(x,t)}} \alpha_{j,x} \quad \forall t \in T,\ i \in V \tag{44}$$

Substituting the above constraints to constraints (3), (4), (36), and (37), respectively, in Formulations 3.7, 3.8, and 3.9 we can calculate the equilibrium patrolling strategies.

Using graphs with arbitrary weights makes the patroller movement model more expressive, allowing one to capture the time spent in movements. In order to make this model even more expressive, we could combine arbitrary weight graphs with the model used in [3], where the patrolling robot has a heading and changing the heading requires one turn during which the robot stays in the same vertex. The resulting model would lead to define the states of the patroller as a pair: a vertex and an orientation. This more realistic model has the drawback to increase the number of patroller's states and, therefore, it would make finding a leader–follower equilibrium harder.

A.5. Removing attacker's dominated strategies with abstracted games

The algorithm to remove the attacker's dominated strategies in the abstracted game is a simple variation of the algorithm presented in Section 5.1.2. We report it as Algorithms 9 and 10.

Algorithm 9: INTRUDER_DOMINATION

```
1  for each t ∈ T do
2      tabu(t) = {}
3      for each v ∈ V do
4          domination(t, v) = V
5          delay(t, v) = 0
6      EXPAND(t, t, {t}, 0)
7      for each v ∈ V do
8          for each w ∈ domination(t, v) do
9              if dist(v, w) < delay(t, w) then
10                 domination(t, v) = domination(t, v) \ {w}
11 for each t ∈ T do
12     tabu(t) = {v ∈ V | ∀t' ∃t ∈ domination(t', v), u_a(penetration-t) ⩽ u_a(penetration-t')}
13     nondominated(t) = V \ {⋃_{v∈V\{t}} domination(t, v) ∪ tabu(t)}
```

Algorithm 9 works exactly as Algorithm 5 except for the following points. In Step 2 it defines a variable $delay(t, v) = 0$, in Step 3 the length of the paths is measured in terms of temporal cost, in Step 5 for each vertex $\eta(q)$ we consider the largest number of turns (i.e., the delay) the patroller must spend along the abstracted arcs to reach $\eta(q)$, and in Step 6 the set domination is reduced considering also the delay $delay(t, v)$. In particular, focusing on this last step, given a target t

Algorithm 10: EXPAND($v, t, B, depth$)

```
1 N = {f | father(v) ≠ η(f) ≠ η(v), a(η(f), η(v)) = 1}
2 for each f ∈ N do
3     domination(t, η(f)) = domination(t, η(f)) ∩ η(B)
4     if dist(η(v), η(f)) > 1 then
5         delay(t, η(v)) = max{delay(t, η(v)), dist(η(v), η(f)) − 1}
6 if depth < d(t) then
7     for each f ∈ N do
8         EXPAND(f, t, {B ∪ f}, depth + 1)
```

and *domination*(t, v) computed as described in Steps 3–5, a vertex v dominates v' only if $delay(t, v) \leqslant dist(v, v')$. Indeed, if $delay(t, v) > dist(v, v')$, we have not any guarantee that the capture probability of *enter-when*(t, v) is smaller than the capture probability of *enter-when*(t, v') with a delay of $delay(t, v')$.

Appendix B. Proofs

B.1. Proof of Proposition 3.11

Consider the setting of Fig. 1 with the following penetration times:

$$d(06) = 14, \qquad d(08) = 18, \qquad d(12) = 23, \qquad d(14) = 22, \qquad d(18) = 18$$

This PSG admits a leader–follower deterministic patrolling strategy where the patroller follows a cycle over the targets moving along the shortest paths (i.e., 14, 08, 06, 18, 12, 18, 06, 08, 14). The best intruder's action is *stay-out* independently of the value of ϵ, otherwise it would be captured with a probability of one. It can be easily observed that this patrolling strategy implies $l = 2$. Suppose to apply Formulation 3.8 to such a game. We can show that we can always find a value of ϵ such that there is no patrolling strategy with $l = 1$ such that *stay-out* is the intruder's best response. Consider action *enter-when*$(06, 23)$, the associated capture probability is always smaller than one when $l = 1$. Indeed, the values $\alpha_{11,i}$ with $i \in \{06, 12, 18\}$ are strictly positive to assure that the patroller can cover all the targets. Then, by Markov chains, it follows that the probability that the patroller reaches vertex 06 starting from vertex 23 within 9 turns is strictly smaller than one. We can always find a strictly positive value of ϵ such that, when the patroller follows the strategy with $l = 1$, the intruder strictly prefers to attack a target rather than not to attack. Since the intruder will attack and the probability of being captured is strictly lower smaller one, the utility expected by the patroller from following the strategy with $l = 1$ will be strictly smaller than that expected utility from following the deterministic equilibrium strategy with $l = 2$.

B.2. Proof of Theorem 3.13

No first-order Markovian leader–follower equilibrium strategy can provide less than $0.5 \cdot \sum_i v_{\mathbf{d}}(i)$. Assume by contradiction that a leader–follower equilibrium strategy σ^* gives less than $0.5 \sum_i v_{\mathbf{d}}(i)$. Then there is a target t such that $v_{\mathbf{d}}(t) \geqslant 0.5 \cdot \sum_i v_{\mathbf{d}}(i)$. If the defender covers only such target with a probability of one, it would gain at least $0.5 \cdot \sum_i v_{\mathbf{d}}(i)$. Therefore, σ^* cannot be a leader–follower strategy and we have a contradiction.

We now show that efficiency $\frac{1}{2}$ can be arbitrarily achieved. Consider a setting with two targets t_1 and t_2, both with penetration time equal to d and value, for the defender, equal to M. Assume that the topology is linear with the two targets as extreme vertices. Assume that the number of vertices (including targets) is $2d - 2$. This setting admits a deterministic equilibrium strategy and therefore the optimal high-order strategy gives the defender $2M$. The first-order Markovian leader–follower equilibrium strategy is such that, as $d \to +\infty$, the capture probability of *enter-when*(t_1, t_2) and *enter-when*(t_2, t_1) (i.e., the two best responses) go to zero. Therefore, the defender's expected utility approaches to M and the efficiency is arbitrarily close to $\frac{1}{2}$.

B.3. Proof of Theorem 4.4

We prove the NP-completeness by reducing the Directed Hamiltonian Circuit problem (DHC) [49] to the DET-STRAT problem. DHC is the problem of determining if a Hamiltonian path, i.e., a path that visits each vertex exactly once, exists in a given directed graph. This is a well-known NP-complete problem. Let us consider a generic instance of the DHC problem given by a directed graph $G_h = (V_h, A_h)$ where V_h is the set of vertices and A_h is the set of arcs. In order to prove that DHC can be reduced to the DET-STRAT problem, we show that for every instance G_h of the DHC problem an instance G'_s of the DET-STRAT problem can be built in polynomial time and that, by solving the DET-STRAT problem on G'_s, we obtain also a solution for the DHC problem on G_h. An instance $G'_s = (T_s, A_s, w_s, d_s)$ can be easily constructed from G_h in the following way: $T_s = V_h$, $A_s = A_h$, for every $v \in T_s$ we impose $d(v) = |V_h|$ and $w_s(v, v') = 1$ for all $v, v' \in T_s$. It is straightforward to see that a solution of G'_s, if it exists, is a Hamiltonian cycle. Indeed, since the relative deadline of every target is equal to the

number of targets, a deterministic equilibrium strategy should visit each target exactly once, otherwise at least one relative deadline would be violated (being $w_s(v, v') = 1$ for all $v, v' \in T_s$). Therefore, computing the solution for G'_s provides by construction a solution for G_h or, in other words, the DHC problem can be reduced to the DET-STRAT problem, proving its NP-completeness (the proof is completed by noting that it is trivially polynomial to verify that a given sequence of vertices is a solution of the DET-STRAT problem).

B.4. Proof of Theorem 4.5

In order to prove the theorem it is sufficient to prove that, if a problem is solvable, then there exists a solution σ in which there is at least a vertex that only appears once, excluding $\sigma(s)$. Indeed, if this statement holds then the maximum temporal length of σ is bounded by $d(i)$ where i is the vertex that appears only one time in σ. It easily follows that, in the worst case, the maximum temporal length of σ is $\max_{t \in T}\{d(t)\}$.

We now prove that, if the problem is solvable, then there is a solution in which at least a vertex appears only once. To prove this, we consider a solution σ wherein $\sigma(1)$ is the vertex with the minimum relative deadline, i.e., $\sigma(1) = \arg\min_{t \in T}\{d(t)\}$. (Notice that this assignment does not preclude finding a solution.) We call k the minimum integer such that all the vertices appear in the subsequence $\sigma(1) - \sigma(k)$. We show that, if the problem is solvable, then it is not necessary that vertex $v = \sigma(k)$ appears again after k. A visit to v after k would be observed if either it is necessary to pass through v to reach $\sigma(1)$ or it is necessary to re-visit v, due to its relative deadline, before $\sigma(1)$. However, since all the vertices but $v = \sigma(k)$ are visited before k, all the vertices but v can be visited without necessarily visiting v. Furthermore, the deadline of $\sigma(1)$ is by hypothesis harder than $\sigma(k)$'s one and then the occurrence of $v = \sigma(k)$ after k is not necessary. Therefore, vertex $\sigma(k)$ occurs only one time.

B.5. Proof of Theorem 4.6

We initially prove the soundness of the algorithm. We need to prove that all the solutions it produces satisfy constraints (10)–(14). Constraints (10), (11), and (14) are satisfied by Algorithm 3. If at least one of them does not hold, no solution is produced. The satisfaction of constraints (12) is assured by Algorithm 4 in Step 3, while the satisfaction of constraints (13) is assured by Algorithm 4 in Steps 6 and 9.

In order to prove completeness we need to show that the algorithm produces a solution whenever at least one exists. In the algorithm there are only two points in which a candidate solution is discarded. The first one is the forward checking in Algorithm 4. Indeed, it iteratively applies constraints (13)–(14) to a partial sequence σ exploiting a heuristic over the future weights (i.e., the time spent to visit the successive vertices). Since the employed heuristic is admissible, no feasible candidate solution can be discarded. The second point is the stopping criterion in Algorithm 3: when all the vertices occur in σ (at least once) and the first and the last vertex in σ are equal, no further successor is considered and the search is stopped. If σ satisfies all the constraints, then σ is a solution, otherwise backtracking is performed. We show that, if a solution can be found without stopping the search at this point, then a solution can be found also by stopping the search and backtracking (the *vice versa* does not hold). This issue is of paramount importance since it assures that the algorithm terminates (in Section 4.4.2 we provide an example in which, without this stopping criterion, the search could not terminate). Consider a σ such that $\sigma(1) = \sigma(s)$ and including all the vertices in T. The search subtree following $\sigma(s)$ and produced by the proposed algorithm is (non-strictly) contained in the search tree following from $\sigma(1)$. This is because the constraints considered by the forward checking from $\sigma(s)$ on are (non-strictly) harder than those considered from $\sigma(1)$ to $\sigma(s)$. The increased hardness is due to the activation of constraints (13) that are needed given that at least one occurrence of each vertex is in σ. Thus, if a solution can be found by searching from $\sigma(s)$, then a shorter solution can be found by stopping the search at $\sigma(s)$ and backtracking. This concludes the proof of completeness.

B.6. Proof of Theorem 5.1

Call z a vertex that is not on any shortest path between any pair of targets. If a strategy $\sigma_{\mathbf{d}}$ prescribes that the patroller can make action $move(z)$ with a strictly positive probability, then it can be easily observed that, if the patroller does not make such action, it cannot decrease its expected utility. Indeed, the intruder capture probability $P_c(t, x)$ for any $t \in T$ and $x \in V$ cannot decrease since visiting z would only introduce an unnecessary temporal cost.

B.7. Proof of Theorem 5.2

The proof is trivial. When the patroller moves along Q for going from t_1 to t_2, the intruder capture probabilities are not smaller than in the case in which the patroller moves along P.

B.8. Proof of Theorem 5.3

The idea is that by setting $\alpha(i, i) = 0$ for every $i \in V \setminus T$, the intruder capture probabilities do not decrease. We consider a simple situation with two vertices adjacent to j, but the same argument can be applied to situations in which j has any

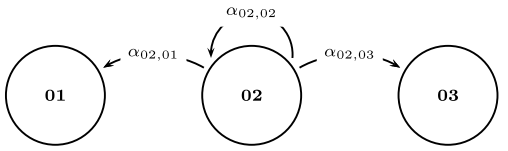


Fig. 19. Example used in the proof of Theorem 5.3.

number of adjacent vertices. Consider Fig. 19 where all vertices are not targets. Given $\alpha_{02,01}, \alpha_{02,02}, \alpha_{02,03}$, the probability to reach 01 from 02 after an infinite number of turns is $\frac{\alpha_{02,01}}{1-\alpha_{02,02}}$, while the probability to reach 03 from 02 after an infinite number of turns is $\frac{\alpha_{02,03}}{1-\alpha_{02,02}}$. By setting $\alpha'_{02,01} = \frac{\alpha_{02,01}}{1-\alpha_{02,02}}$, $\alpha'_{02,02} = 0$, and $\alpha'_{02,03} = \frac{\alpha_{02,03}}{1-\alpha_{02,02}}$, the probabilities to reach 01 and 03 from 02 do not decrease for any possible number of turns. Therefore, we obtain the thesis. Of course, it is easy to see that the same does not hold when we set $\alpha_{i,i} = 0$ with $i \in T$.

B.9. Proof of Theorem 5.6

We prove the 'if' part. (i) and (ii) imply $(1 - P_c(\bar{t}, \bar{i}))u_{\mathbf{a}}(\textit{penetration-}\bar{t}) < (1 - P_c(\bar{s}, \bar{j}))u_{\mathbf{a}}(\textit{penetration-}\bar{s})$ for every fully mixed strategy $\sigma_{\mathbf{d}}$. By continuity, with non-fully mixed strategies we have $(1 - P_c(\bar{t}, \bar{i}))u_{\mathbf{a}}(\textit{penetration-}\bar{t}) \leqslant (1 - P_c(\bar{s}, \bar{j}))u_{\mathbf{a}}(\textit{penetration-}\bar{s})$, that, since $u_{\mathbf{a}}(\textit{intruder-capture})$ is non-positive, implies the definition of dominance.

We prove the 'only if' part of (i). For all the possible patrolling settings, if $u_{\mathbf{a}}(\textit{penetration-}\bar{t}) > u_{\mathbf{a}}(\textit{penetration-}\bar{s})$, it is possible to find a fully mixed strategy $\sigma_{\mathbf{d}}$ such that $EU_{\mathbf{a}}(\textit{enter-when}(\bar{t}, \bar{i})) > EU_{\mathbf{a}}(\textit{enter-when}(\bar{s}, \bar{j}))$ in the following way. We set all the probabilities leading to $\bar{s}$ from $\bar{j}$ equal to $1 - \epsilon$ with $\epsilon > 0$ arbitrarily small. If the path connecting $\bar{t}$ to $\bar{i}$ is not strictly contained in the path connecting $\bar{s}$ to $\bar{j}$, then we can set some probability in the path connecting $\bar{t}$ to $\bar{i}$ equal to ϵ and thus $(1 - P_c(\bar{t}, \bar{i})) \simeq 1$ and $(1 - P_c(\bar{s}, \bar{j})) \simeq 0$, satisfying the previous inequality. If the path connecting $\bar{t}$ to $\bar{i}$ is strictly contained in the path connecting $\bar{s}$ to $\bar{j}$, we have $(1 - P_c(\bar{t}, \bar{i})) < (1 - P_c(\bar{s}, \bar{j}))$. However, we can set the probabilities leading to $\bar{s}$ from $\bar{t}$ equal to $1 - \epsilon'$ such that $P_c(\bar{s}, \bar{j}) \geqslant (1 - \epsilon')^k P_c(\bar{t}, \bar{i})$ where k is the distance between $\bar{t}$ and $\bar{s}$. It is always possible to find an ϵ' such that $P_c(\bar{s}, \bar{j}) - P_c(\bar{t}, \bar{i})$ is arbitrarily small and, since the difference between $u_{\mathbf{a}}(\textit{penetration-}\bar{t})$ and $u_{\mathbf{a}}(\textit{penetration-}\bar{s})$ is finite, $EU_{\mathbf{a}}(\textit{enter-when}(\bar{t}, \bar{i})) > EU_{\mathbf{a}}(\textit{enter-when}(\bar{s}, \bar{j}))$.

We prove the 'only if' part of (ii). If there exists a strategy $\sigma_{\mathbf{d}}$ such that $P_c(\bar{t}, \bar{i}) < P_c(\bar{s}, \bar{j})$, then the path connecting $\bar{t}$ to $\bar{i}$ is not strictly contained in the path connecting $\bar{s}$ to $\bar{j}$. In this case, we can find a $\sigma_{\mathbf{d}}$ (as we discussed above) such that $(1 - P_c(\bar{t}, \bar{i})) \simeq 1$ and $(1 - P_c(\bar{s}, \bar{j})) \simeq 0$, and therefore action $\textit{enter-when}(\bar{t}, \bar{i})$ is not dominated.

B.10. Proof of Theorem 5.10

The proof is trivial. The intruder's action $\textit{enter-when}(t, t)$ being dominated, the intruder will never enter t when the patroller is in t. Therefore, setting $\alpha_{t,t} = 0$, the probability that the intruder will be captured when it enters t will never decrease.

B.11. Proof of Theorem 5.18

The proof has two steps. In the first one, we show that, after the application of the lossless abstractions, the set of intruder's dominated strategies is left unchanged, and therefore we can focus only on the dominant strategies. In the second one, we show that, for any strategy σ in the non-abstracted game, we can find, by solving the abstracted game, a strategy σ' that gives the patroller a utility not smaller than that given by σ. With abuse of notation, we denote by $P_c(x, y, z)$ the probability that the intruder is captured after z turns once it entered vertex x when the patroller was at vertex y.

We prove the first step by showing that in the abstracted game the intruder's probabilities to be captured when it takes a dominated action (in the non-abstracted original game) are larger than when it takes a dominant action (in the original game). Exactly, given an abstraction over a pair of vertices i, j and called k a vertex belonging to the shortest path between i and j, we need to prove that, for every target t and $dom(k, t)$:

$$P_c\big(k, t, d(t)\big) \geqslant P_c\big(dom(k, t), t, d(t)\big)$$

By applying our abstractions, we have:

$$P_c\big(k, t, d(t)\big) = \max\big\{P_c\big(i, t, d(t) - dist(k, i)\big), P_c\big(j, t, d(t) - dist(k, j)\big)\big\}$$

and

$$P_c\big(dom(k,t),t,d(t)\big) = \max\Bigg\{ \sum_{h=dist(dom(k,t),i)}^{d(t)-dist(i,t)} P_r\big(dom(k,t),i,h\big)\cdot P_c\big(i,t,d(t)-h\big), \sum_{h=dist(dom(k,t),j)}^{d(t)-dist(j,t)} P_r\big(dom(k,t),j,h\big)\cdot P_c\big(j,t,d(t)-h\big)\Bigg\}$$

Since $dist(dom(k,t),j) \geqslant dist(k,j)$ and $dist(dom(k,t),i) \geqslant dist(k,i)$:

$$P_c\big(k,t,d(t)\big) \geqslant \max\Bigg\{ \sum_{h=dist(k,i)}^{d(t)-dist(i,t)} P_r(k,i,h)\cdot P_c\big(i,t,d(t)-h\big), \sum_{h=dist(k,j)}^{d(t)-dist(j,t)} P_r(k,j,h)\cdot P_c\big(j,t,d(t)-h\big)\Bigg\} \geqslant P_c\big(dom(k,t),t,d(t)\big)$$

We prove the second step. Consider the basic situation of Fig. 9. Suppose that probabilities $\alpha_{01,02}, \alpha_{02,01}, \alpha_{02,03}, \alpha_{03,02}$ constitute a part of a leader–follower equilibrium. We can show that we can always find values of $\alpha_{01,03}, \alpha_{03,01}$ such that the capture probabilities in the abstracted game are not smaller than those in the non-abstracted game. Assign $\alpha_{01,03} = \alpha_{01,02}$ and $\alpha_{03,01} = \alpha_{03,02}$. Assume for simplicity that, once the arc $(01,\cdot)$ is traversed, the probability to come back to 01 is equal to zero. The probability to reach 03 from 01 within 2 turns in the abstracted game is $\alpha_{01,03}$. The probability to reach 03 from 01 within an infinite number of turns in the original game is:

$$\alpha_{01,02}\cdot(1-\alpha_{02,01})\sum_{l=0}^{+\infty}(\alpha_{01,02}\cdot\alpha_{02,01})^l = \frac{\alpha_{01,02}\cdot(1-\alpha_{02,01})}{1-\alpha_{01,02}\cdot\alpha_{02,01}}$$

Being $\frac{1-\alpha_{02,01}}{1-\alpha_{01,02}\cdot\alpha_{02,01}} < 1$ we have that $\frac{\alpha_{01,02}\cdot(1-\alpha_{02,01})}{1-\alpha_{01,02}\cdot\alpha_{02,01}} < \alpha_{01,03}$ and therefore the abstraction preserves the optimality of the solution. Given an arbitrary information lossless abstraction, we can apply iteratively the above procedure showing that computing equilibrium strategies in the abstracted game allows one to find strategies as good as those in the original game.

B.12. Proof of Lemma 6.1

If a patrolling strategy is inconsistent, then all the best response constraints related to the attack of the targets that are not patrolled are not active. This is because the expected utility of the attacker is larger than the value of the non-patrolled targets. As described in Section 6.2, solving the game in which the non-patrolled targets are removed from G is equivalent to substitute the best response constraints related to the non-patrolled targets with new relaxed constraints. These constraints being relaxed, the optimal solution is at least good as in the initial problem.

B.13. Proof of Lemma 6.2

The proof is by contradiction. Assume that $G(T_c)$ admits a consistent solution. Assume, by contradiction, that a sub-partition $T'_c \subset T_c$ leads to better consistent solutions. Consider targets $T_c \setminus T'_c$. These targets are not attacked in the solution of $G(T'_c)$, otherwise the solution of $G(T'_c)$ would be worse than that of $G(T_c)$, and the associated capture probabilities are zero. However, if targets $T_c \setminus T'_c$ were not attacked in the solution of $G(T'_c)$, they would not be attacked neither in $G(T_c)$ and therefore the defender could improve its utility by not patrolling such targets, but this leads to a contradiction.

B.14. Proof of Theorem 6.3

The algorithm is optimal because it enumerates all the possible partitions of targets in T_c and T_{-c} except considering sub-partitions of T_c when solving $G(T_c)$ returns a consistent strategy. By Lemma 6.2, these sub-partitions cannot contain better solutions than that associated to $G(T_c)$.

Appendix C. Experimental results tables

Table 2
Computing a deterministic patrolling strategy: experimental results for finding a deterministic patrolling strategy with the most significant configurations of the algorithm (we set a time deadline of 10 minutes). Best configurations are in bold.

	n	3	4	5	6	7	8	100	250	500
$h_{\min v}$	term (%)	**100**	**100**	**100**	**99.8**	**99.6**	**99.5**	**98.9**	**96.6**	**90.2**
RTB	time (s)	< **0.01**	< **0.01**	< **0.01**	0.32	**0.10**	0.05	0.16	0.87	5.50
LSC	dev (s)	< 0.01	< 0.01	< 0.01	5.17	1.78	0.96	3.52	14.47	30.28

Table 2 (*continued*)

	n	3	4	5	6	7	8	100	250	500
IFC	max (s)	< 0.01	< 0.01	< 0.01	98.00	35.00	19.00	78.26	316.9	413.94
	min (s)	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	0.01	0.07
	suc (%)	60	58	62	59	61	60	61	63	64
h_r	term (%)	**100**	**100**	**100**	98.5	97.5	96.5	95.1	55.1	9.8
LSC	time (s)	< **0.01**	< **0.01**	0.11	**0.09**	0.16	0.02	1.34	2.52	4.66
IFC	dev (s)	< 0.01	< 0.01	1.64	1.70	1.73	0.18	6.19	16.75	51.62
	max (s)	< 0.01	< 0.01	32.00	33.00	24.00	2.00	93.36	513.66	590.87
	min (s)	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	0.01	0.07
	suc (%)	60	58	62	59	62	62	62	78	92
h_r	term (%)	**100**	**100**	99.0	97.2	96.7	95.5	94.0	53.0	8.9
IFC	time (s)	< **0.01**	0.44	3.65	0.14	0.26	**0.01**	1.35	3.41	5.94
	dev (s)	< 0.01	8.68	38.89	2.24	2.36	0.16	39.32	18.02	55.14
	max (s)	< 0.01	173.55	594.10	43.03	31.86	2.09	561.95	501.72	582.77
	min (s)	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	0.01	0.07
	suc (%)	60	58	62	59	63	63	63	80	93
$h_{\min v}$	term (%)	**100**	**100**	**100**	96.7	96.0	95.5	95.0	93.3	86.2
RTB	time (s)	< **0.01**	< **0.01**	0.34	2.98	0.16	**0.01**	0.30	1.00	6.19
LSC	dev (s)	< 0.01	< 0.01	6.29	33.77	2.24	0.11	6.50	15.32	35.77
	max (s)	< 0.01	< 0.01	125.03	519.75	42.22	2.41	145.22	366.42	498.04
	min (s)	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	0.01	0.07
	suc (%)	60	58	62	60	63	63	63	63	67
h_r	term (%)	**100**	**100**	**100**	95.4	93.9	92.5	91.2	52.4	7.7
LSC	time (s)	< **0.01**	< **0.01**	0.79	3.04	0.30	0.03	1.36	3.48	5.83
	dev (s)	< 0.01	< 0.01	13.58	24.32	2.77	0.21	39.53	18.46	55.65
	max (s)	< 0.01	< 0.01	270.03	303.72	41.53	2.83	566.04	531.64	596.42
	min (s)	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	0.02	0.01	0.07
	suc (%)	60	58	62	61	63	64	64	82	94
h_r	term (%)	**100**	**100**	98.7	94.2	93.0	91.8	90.3	51.0	7.1
	time (s)	< **0.01**	7.45	2.45	4.78	1.38	0.14	1.37	3.74	6.18
	dev (s)	< 0.01	55.45	28.61	42.13	9.96	1.03	6.28	18.45	56.80
	max (s)	< 0.01	556.92	506.72	496.84	140.31	12.86	93.26	516.72	576.52
	min (s)	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	0.02	0.01	0.07
	suc (%)	60	58	62	61	63	64	65	83	95
h_l	term (%)	**100**	99.2	91.0	81.1	75.3	69.0	3.9	2.3	1.5
LSC	time (s)	< **0.01**	7.45	2.45	4.78	1.38	0.14	**0.10**	**0.01**	**0.07**
IFC	dev (s)	< 0.01	55.45	28.61	42.13	9.96	1.03	< 0.01	< 0.01	< 0.01
	max (s)	< 0.01	548.41	505.74	497.46	140.11	12.01	< 0.01	0.01	0.07
	min (s)	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	0.01	0.07
	suc (%)	60	58	65	< 0.01	< 0.01	< 0.01	< 0.01	0.01	0.07
h_l	term (%)	**100**	99.2	88.0	78.0	71.7	65.0	0.0	0.0	0.0
	time (s)	< **0.01**	7.42	2.61	5.12	1.61	0.20	–	–	–
	dev (s)	< 0.01	55.23	28.66	42.65	10.57	1.29	–	–	–
	max (s)	< 0.01	548.41	505.74	497.46	140.11	12.01	–	–	–
	min (s)	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	–	–	–
	suc (%)	60	58	67	71	74	86	–	–	–

Table 3
Computing a non-deterministic patrolling strategy in open perimetral settings: number of candidate best responses for the *basic* case (without any reduction) and the *dom* case (once dominated strategies have been removed).

Number of vertices (n)	Percentage of targets/vertices (δ)									
	10%		20%		30%		40%		50%	
	basic	dom	basic	dom	basic	dom	basic	dom	basic	dom
10	10	2	20	2	30	3.8	40	4	50	4.6
20	40	2	80	3.2	120	4	160	4.8	200	6
30	90	3.4	180	4.4	270	4	360	5.28	450	5.8
40	160	2.8	320	5.4	480	5.2	640	7	800	9.2
60	360	5	720	7	1080	7	1440	7.8	1800	7.6
80	640	6	1280	6.2	1920	8.6	2560	8.2	3200	8
100	1000	5.6	2000	5.8	3000	9.4	4000	8.6	5000	9.2
130	1690	11.8	3380	7.2	5070	7.8	6760	9	8450	9.2
160	2560	6.6	5120	8.4	7680	7.4	10,240	11.2	12,800	9
200	4000	6.2	8000	8.8	12,000	7.4	16,000	10	20,000	11.8

Table 4
Computing a non-deterministic patrolling strategy in open perimetral settings: number of remaining vertices after the application of lossless abstractions.

Number of vertices (n)	Percentage of targets/vertices (δ)				
	10%	20%	30%	40%	50%
10	5	5	5.2	5.2	5.6
20	6	6.4	6.6	6.4	7.2
30	6.6	7.2	7.8	8	8.4
40	7.2	8.2	8.8	9.6	10.4
60	8.6	9.8	10.2	10.6	10.2
80	9.8	9.2	11	11.8	11
100	10.2	9.4	12.4	11.8	12.2
130	10.4	10.8	11.4	12.2	12.6
160	11.2	12.8	11.8	14.2	13
200	11	12.75	11.5	14.4	15.4

Table 5
Computing a non-deterministic patrolling strategy in open perimetral settings: computational times and standard deviations (in parentheses) with zero-sum (top line) and general-sum (bottom line) settings.

Number of vertices (n)		Percentage of targets/vertices (δ)				
		10%	20%	30%	40%	50%
10	basic	0.36 (0.1)	0.39 (0.2)	0.48 (0.2)	0.57 (0.2)	0.77 (0.2)
		3.44 (1.5)	7.51 (1.2)	15.01 (2.4)	22.87 (3.0)	39.01 (4.6)
	dom	0.05 (0.0)	0.05 (0.0)	0.07 (0.0)	0.08 (0.0)	0.08 (0.0)
		0.12 (0.0)	0.16 (0.0)	0.29 (0.0)	0.30 (0.1)	0.37 (0.1)
	lossless	0.08 (0.0)	0.08 (0.0)	0.09 (0.0)	0.10 (0.0)	0.11 (0.0)
		0.11 (0.0)	0.15 (0.0)	0.36 (0.1)	0.43 (0.1)	0.57 (0.2)
20	basic	586.34 (1.2)	23.55 (5.9)	46.45 (20)	77.38 (24)	150.99 (84)
		23,018 (1502)	1899 (393)	5782 (963)	12,531 (1283)	33,912 (4142)
	dom	0.29 (0.1)	0.59 (0.2)	0.39 (0.3)	0.63 (0.3)	0.51 (0.4)
		0.61 (0.2)	1.98 (0.6)	1.65 (0.7)	2.89 (0.9)	3.13 (0.9)
	lossless	0.12 (0.0)	0.15 (0.0)	0.15 (0.0)	0.18 (0.0)	0.21 (0.1)
		0.22 (0.0)	0.51 (0.0)	0.59 (0.1)	0.82 (0.1)	1.43 (0.3)
30	basic	122.24 (24)	506.68 (259)	1304 (516)	1941 (638)	2336 (679)
		11,731 (1331)	–	–	–	–
	dom	1.69 (1.5)	1.47 (1.7)	4.31 (3.6)	2.59 (4.4)	2.69 (5.7)
		5.82 (0.7)	7.01 (1.3)	16.64 (4.2)	14.61 (5.3)	15.91 (5.5)
	lossless	0.19 (0.1)	0.23 (0.1)	0.32 (0.1)	0.36 (0.1)	0.42 (0.1)
		0.71 (0.2)	1.04 (0.3)	1.52 (0.7)	1.84 (0.8)	2.32 (0.9)
40	basic	1614.14 (315)	–	–	–	–
	dom	4.94 (1.1)	5.13 (1.9)	6.10 (2.1)	5.85 (2.5)	8.88 (2.8)
		13.92 (2.4)	28.18 (3.9)	31.98 (4.1)	40.51 (4.7)	79.14 (11.5)
	lossless	0.36 (0.1)	0.44 (0.1)	0.80 (0.2)	1.02 (0.2)	1.61 (0.3)
		1.01 (0.2)	2.37 (0.5)	4.19 (0.9)	7.21 (1.0)	14.90 (2.7)
60	basic	–	–	–	–	–
	dom	30.66 (18)	25.23 (18)	43.24 (20)	22.83 (25)	37.57 (29)
		153.30 (26.1)	176.61 (30.2)	302.68 (45.2)	178.07 (54.6)	285.53 (60.4)
	lossless	0.67 (0.1)	1.22 (0.3)	1.77 (0.3)	3.10 (0.9)	4.48 (1.3)
		3.35 (0.5)	8.61 (1.0)	13.00 (2.4)	24.21 (4.2)	33.84 (6.3)
80	basic	–	–	–	–	–
	dom	47.43 (21)	54.68 (24)	83.51 (29)	103.74 (70)	–
	dom	250.82 (30.1)	341.63 (53.1)	731.42 (85.2)	867.42 (93.1)	
	lossless	1.63 (0.3)	2.64 (0.7)	4.57 (0.8)	7.66 (1.9)	10.55 (2.5)
		9.72 (1.4)	16.87 (2.6)	39.52 (5.7)	66.42 (9.2)	89.09 (13.6)
100	basic	–	–	–	–	–
	dom	100.25 (36)	120.08 (41)	–	–	–
		568.12 (69.4)	701.42 (100.7)			
	lossless	7.32 (0.6)	4.34 (0.7)	10.23 (1.9)	14.89 (2.5)	24.03 (2.9)
		40.99 (8.7)	25.17 (6.6)	96.16 (11.5)	128.62 (16.3)	230.63 (42.7)
130	basic	–	–	–	–	–
	dom	–	–	–	–	–
	lossless	4.32 (0.5)	11.79 (3.0)	24.09 (2.5)	39.25 (3.7)	59.25 (3.6)
		50.88 (8.2)	90.47 (14.5)	190.31 (26.6)	366.12 (40.1)	569.72 (237.6)
160	basic	–	–	–	–	–

Table 5 (*continued*)

Number of vertices (n)		Percentage of targets/vertices (δ)				
		10%	20%	30%	40%	50%
	dom	–	–	–	–	–
	lossless	9.15 (2.1)	28.78 (7.3)	51.25 (7.6)	95.47 (11)	129.95 (15)
		60.52 (8.5)	213.50 (32.5)	380.09 (56.2)	1067 (156)	1201 (137)
200	basic	–	–	–	–	–
	dom	–	–	–	–	–
	lossless	22.11 (5)	67.24 (9)	117.20 (10)	220.73 (21)	367.06 (51)
		142.12 (21.2)	561.25 (70.2)	856.29 (99.1)	22,018 (583)	4118 (973)

Table 6
Computing a non-deterministic patrolling strategy in closed perimetral settings: number of candidate best responses for the *basic* case (without any reduction) and the *dom* case (once dominated strategies have been removed) when no abstractions or lossy abstractions are used.

Number of vertices (n)		Percentage of targets/vertices (δ)									
		10%		20%		30%		40%		50%	
		basic	dom	basic	dom	basic	dom	basic	dom	basic	dom
16	no	32	22	64	40	77	46	103	55	178	78
	lossy	14	10	21	17	44	26	63	37	80	49
20	no	40	25	80	38	120	83	160	102	200	121
	lossy	16	18	36	20	49	34	80	52	120	76
24	no	96	32	116	62	173	114	231	121	178	153
	lossy	15	17	44	27	70	48	117	69	160	92
28	no	112	55	157	99	236	154	314	195	392	206
	lossy	27	22	56	40	117	70	156	94	224	118
32	no	128	78	205	118	308	190	410	204	512	296
	lossy	18	18	72	45	126	84	204	113	288	165
44	no	194	117	388	212	580	300	–		–	
	lossy	44	38	96	66	221	123	357	209	528	273
64	no	–		–		–		–		–	
	lossy	66	49	208	126	418	238	800	309	1088	556
84	no	–		–		–		–		–	
	lossy	112	76	391	192	725	432	–		–	

Table 7
Computing a non-deterministic patrolling strategy in closed perimetral settings: number of remaining vertices after the application of lossy abstractions.

Vertices (n)	Percentage of targets/vertices (δ)				
	10%	20%	30%	40%	50%
16	7	7.2	8.8	10.6	10
20	7.8	9	8.2	10.4	12.4
24	6	8.8	10.4	13.2	14.4
28	9	10.8	12.8	13.2	16.2
32	6	11.8	14.2	17.6	18.2
44	11.6	12	17.4	21.2	24
64	11.4	16	22.4	30.8	34.2
84	14	23	29.8	–	–

Table 8
Computing a non-deterministic patrolling strategy in closed perimetral settings: computational times and standard deviations (between parentheses) with zero-sum (top line) and general-sum (bottom line) settings.

Number of vertices (n)		Percentage of targets/vertices (δ)				
		10%	20%	30%	40%	50%
16	basic	5.87 (1.5)	5.83 (2.5)	7.92 (3.6)	12.85 (4.6)	15.04 (5.1)
		198.53 (25.37)	386.75 (42.73)	609.47 (120.34)	1324 (300.08)	1929 (546.99)
	dom	3.32 (1.3)	3.79 (2.2)	3.82 (1.4)	6.40 (2.7)	7.37 (3.0)
		73.04 (12.29)	155.24 (20.64)	178.04 (35.72)	358.00 (80.11)	574.86 (130.73)

(*continued on next page*)

Table 8 (*continued*)

Number of vertices (n)		Percentage of targets/vertices (δ)				
		10%	20%	30%	40%	50%
	lossy	1.58 (0.4) 16.52 (3.75)	0.25 (0.1) 4.25 (2.17)	0.37 (0.2) 9.62 (5.63)	1.95 (0.7) 72.15 (10.80)	1.44 (0.8) 70.56 (11.67)
20	basic	14.89 (4) 1190 (167.63)	20.07 (9) 1725 (238.41)	39.81 (10) 4521 (550.98)	70.07 (25) 11,315 (2452)	93.61 (26) 18,362 (4873)
	dom	8.45 (3.0) 380.12 (98.81)	5.96 (3.0) 207.54 (50.24)	37.41 (17) 3007 (784.21)	40.55 (20) 3952 (996.55)	54.87 (21) 6426 (1652)
	lossy	0.52 (0.25) 9.42 (2.43)	0.25 (0.1) 4.77 (1.00)	0.64 (0.3) 21.53 (4.09)	2.07 (1.6) 100.57 (33.72)	5.51 (2.9) 419.00 (167.52)
24	basic	65.43 (35) 6280 (1263)	82.88 (43) 9421 (3172)	214.00 (51) 37,009 (9730)	332.17 (111) 73,962 (23,662)	406.66 (101) –
	dom	39.71 (21) 2496 (762)	33.03 (16) 1875 (540)	137.96 (30) 14,422 (5927)	111.86 (35) 13,526 (4227)	219.53 (53) 30,624 (10,826)
	lossy	0.56 (0.2) 9.57 (2.09)	0.38 (0.2) 11.58 (3.72)	1.94 (1.1) 93.11 (25.82)	6.28 (4.1) 433.73 (102.37)	11.17 (3.5) 1042 (296)
28	basic	85.46 (0.1) 9562 (3712)	325.44 (2.0) 51,938 (27,261)	637.50 (12) –	1064.62 (14) –	1178 (22) –
	dom	150.67 (41) 8421 (2536)	188.01 (101) 18,225 (5116)	330.47 (176) 48,722 (17,428)	468.69 (152) 78,625 (22,945)	494.90 (282) –
	lossy	0.37 (0.1) 8.66 (2.33)	2.27 (1.3) 91.72 (19.85)	11.69 (3.6) 788.72 (204.57)	16.99 (21) 1567 (308.63)	37.39 (41) 4412 (1037)
32	basic	246.38 (92) 31,488 (10,535)	706.17 (278) –	1774 (392) –	1885 (413) –	2319 (415) –
	dom	130.46 (84) 10,140 (3261)	346.72 (130) 47,420 (17,822)	555.53 (270) –	673.56 (339) –	1474 (460) –
	lossy	0.25 (0.1) 4.52 (1.31)	2.57 (1.7) 115.31 (33.71)	15.52 (5.9) 1352 (273.67)	61.39 (56) 6725 (2.887)	158.57 (65) 26,735 (8.113)
44	basic	2390 (68) –	3306 (964) –	–	–	–
	dom	794.68 (536) –	3076 (1029) –	2691 (1122) –	–	–
	lossy	4.74 (2.1) 180.23 (65.11)	6.35 (4.1) 387.62 (128.51)	120.51 (65) –	515.58 (143) –	1494 (981) –
64	basic	–	–	–	–	–
	dom	–	–	–	–	–
	lossy	12.09 (3.9) 572.62 (200.87)	208.26 (100) 2654 (686.99)	862.59 (166) –	2319 (411) –	3276 (534) –
84	basic	–	–	–	–	–
	dom	–	–	–	–	–
	lossy	69.22 (22) 4862 (1.555)	1381 (546) –	3261 (853) –	–	–

Table 9
Computing a non-deterministic patrolling strategy in arbitrary settings: number of candidate best responses for the *basic* case (without any reduction) and the *dom* case (once dominated strategies have been removed) when no abstractions or lossy abstractions are used.

Number of vertices (n)		Percentage of targets/vertices (δ)							
		5%		10%		20%		30%	
		basic	dom	basic	dom	basic	dom	basic	dom
50	no	150	22	250	81	500	237	750	321
	lossy	75	8	110	26	202	88	345	163
75	no	300	43	600	91	–		–	
	lossy	136	18	272	53	420	158	638	262
100	no	–		–		–		–	
	lossy	143	49	332	102	560	247	870	271
133	no	–		–		–		–	
	lossy	232	65	468	160	702	303	–	
166	no	–		–		–		–	
	lossy	413	79	663	195	–		–	

Table 10
Computing a non-deterministic patrolling strategy in arbitrary settings: number of remaining vertices after the application of lossless and lossy abstractions.

Number of vertices (n)		Percentage of targets/vertices (δ)			
		5%	10%	20%	30%
50	lossless	36.25	40.25	40	42
	lossy	25.25	21.75	20.25	23.25
75	lossless	51.75	53.5	–	–
	lossy	34.33	34	28.25	29
100	lossless	–	–	–	–
	lossy	28.67	33.25	28	29
133	lossless	–	–	–	–
	lossy	34.33	35.5	27	–
166	lossless	–	–	–	–
	lossy	57.66	39	–	–

Table 11
Computing a non-deterministic patrolling strategy in arbitrary settings: computational times and standard deviations (between parentheses) with zero-sum (top line) and general-sum (bottom line) settings.

Number of vertices (n)		Percentage of targets/vertices (δ)			
		5%	10%	20%	30%
50	lossless	10.34 (6.8)	210.97 (184.57)	2923 (1479.8)	6972 (4068)
		269.13 (35.12)	15,084 (2367)	–	–
	lossy	0.32 (0.14)	1.39 (1.32)	110.05 (137.67)	1073 (1310)
		2.56 (0.53)	33.61 (4.59)	9415 (1089)	–
75	lossless	311.14 (285.54)	4615 (6421)	–	–
		13,624 (4852)	–		
	lossy	0.53 (0.3)	53.77 (47.04)	1219 (695.3)	6351 (2898)
		9.58 (1.28)	2849 (771)	–	–
100	lossless	–	–	–	–
	lossy	6.79 (3.47)	1017 (882.57)	5704 (2741)	4045 (1644)
		332.71 (49.31)	–	–	–
133	lossless	–	–	–	–
	lossy	115.62 (29.72)	3710 (938)	4375 (1399)	–
		7435 (994)	–	–	–
166	lossless	–	–	–	–
	lossy	625.82 (296.56)	11,442 (1318)	–	–
		49,055 (11,462)	–		

Table 12
Defender's expected utility by using first-order Markovian strategies (with lossless and lossy abstractions) for different zero-sum and (between parentheses) general-sum settings when there is a non-first-order Markovian deterministic strategy with a value of 1.

Vertices (n)		Percentage of targets/vertices (δ)		
		5%	10%	20%
40	lossless	0.997 (0.998)	0.998 (0.998)	0.999 (0.999)
	lossy	0.888 (0.902)	0.920 (0.937)	0.970 (0.965)
50	lossless	0.997 (0.994)	0.998 (0.999)	0.999 (0.999)
	lossy	0.848 (0.899)	0.951 (0.976)	0.990 (0.992)
60	lossless	0.997 (0.998)	0.998 (0.994)	0.999 (0.999)
	lossy	0.940 (0.952)	0.961 (0.983)	0.981 (0.996)
75	lossless	0.991 (0.989)	0.999 (0.999)	0.999 (0.998)
	lossy	0.924 (0.953)	0.977 (0.982)	0.994 (0.997)

References

[1] M. Adler, H. Räcke, N. Sivadasan, C. Sohler, B. Vöcking, Randomized pursuit–evasion in graphs, Combinatorics, Probability and Computing 12 (2003) 225–244.
[2] N. Agmon, On events in multi-robot patrol in adversarial environments, in: Proceedings of International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS), 2010, pp. 591–598.

[3] N. Agmon, S. Kraus, G. Kaminka, Multi-robot perimeter patrol in adversarial settings, in: Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), 2008, pp. 2339–2345.
[4] N. Agmon, S. Kraus, G. Kaminka, Uncertainties in adversarial patrol, in: Proceedings of the International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS), 2009, pp. 1267–1268.
[5] N. Agmon, S. Kraus, G. Kaminka, V. Sadov, Adversarial uncertainty in multi-robot patrol, in: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), 2009, pp. 1811–1817.
[6] N. Agmon, V. Sadov, G. Kaminka, S. Kraus, The impact of adversarial knowledge on adversarial planning in perimeter patrol, in: Proceedings of the International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS), 2008, pp. 55–62.
[7] A. Almeida, G. Ramalho, H. Santana, P. Tedesco, T. Menezes, V. Corruble, Y. Chevaleyre, Recent advances on multi-agent patrolling, in: Proceedings of the Brazilian Symposium on Artificial Intelligence (SBIA), 2004, pp. 126–138.
[8] S. Alpen, Infiltration games on arbitrary graphs, Journal of Mathematical Analysis and Applications 163 (1) (1992) 286–288.
[9] F. Amigoni, N. Gatti, A. Ippedico, A game-theoretic approach to determining efficient patrolling strategies for mobile robots, in: Proceedings of the IEEE/WIC/ACM International Conference on Agent Intelligent Technology (IAT), Sydney, Australia, 2008, pp. 500–503.
[10] N. Basilico, N. Gatti, Automated abstractions for patrolling security games, in: Proceedings of the National Conference on Artificial Intelligence (AAAI), 2011, pp. 1096–1099.
[11] N. Basilico, N. Gatti, F. Amigoni, Developing a deterministic patrolling strategy for security agents, in: Proceedings of the IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT), 2009, pp. 565–572.
[12] N. Basilico, N. Gatti, F. Amigoni, Leader–follower strategies for robotic patrolling in environments with arbitrary topologies, in: Proceedings of the International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS), 2009, pp. 57–64.
[13] N. Basilico, N. Gatti, T. Rossi, S. Ceppi, F. Amigoni, Extending algorithms for mobile robot patrolling in the presence of adversaries to more realistic settings, in: Proceedings of the IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT), 2009, pp. 557–564.
[14] N. Basilico, N. Gatti, P. Testa, Talos: A tool for designing security applications with mobile patrolling robots, in: Proceedings of the International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS), 2011, pp. 1317–1318.
[15] N. Basilico, N. Gatti, F. Villa, Asynchronous multi-robot patrolling against intrusion in arbitrary topologies, in: Proceedings of the National Conference on Artificial Intelligence (AAAI), 2010, pp. 1224–1229.
[16] N. Basilico, D. Rossignoli, N. Gatti, F. Amigoni, A game-theoretical model applied to an active patrolling camera, in: Proceedings of the International Conference on Emerging Security Technologies (EST), 2010, pp. 130–135.
[17] M. Bazaraa, H. Sherali, C. Shetty, Nonlinear Programming: Theory and Algorithms, Wiley, 2006.
[18] Y. Chevaleyre, Theoretical analysis of the multi-agent patrolling problem, in: Proceedings of the IEEE/WIC/ACM International Conference on Agent Intelligent Technology (IAT), Beijing, China, 2004, pp. 302–308.
[19] N. Christofides, J. Beasley, The period routing problem, Networks 14 (2) (1984) 237–256.
[20] V. Conitzer, T. Sandholm, Computing the optimal strategy to commit to, in: Proceedings of the ACM Conference on Electronic Commerce (EC), 2006, pp. 82–90.
[21] D. Draper, A.K. Jónsson, D.P. Clements, D. Joslin, Cyclic scheduling, in: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), 1999, pp. 1016–1021.
[22] Y. Elmaliach, N. Agmon, G. Kaminka, Multi-robot area patrol under frequency constraints, in: Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), 2007, pp. 385–390.
[23] Y. Elmaliach, A. Shiloni, G. Kaminka, A realistic model of frequency-based multi-robot polyline patrolling, in: Proceedings of the International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS), 2008, pp. 63–70.
[24] M.M. Flood, The hide and seek game of von Neumann, Management Science 18 (5) (1972) 107–109.
[25] R. Fourer, D.M. Gay, B.W. Kernighan, A modeling language for mathematical programming, Management Science 36 (5) (1990) 519–554.
[26] P. Francis, K. Smilowitz, M. Tzur, The period vehicle routing problem with service choice, Transportation Science 40 (4) (2006) 439–454.
[27] D. Fudenberg, J. Tirole, Game Theory, The MIT Press, 1991.
[28] S. Gal, Search Games, Academic Press, 1980.
[29] N. Gatti, Game theoretical insights in strategic patrolling: Model and algorithm in normal form, in: Proceedings of the European Conference on Artificial Intelligence (ECAI), 2008, pp. 403–407.
[30] A. Gilpin, T. Sandholm, Lossless abstraction of imperfect information games, Journal of the ACM 54 (5) (2007).
[31] A. Gilpin, T. Sandholm, T.B. Sørensen, A heads-up no-limit Texas hold'em poker player: Discretized betting models and automatically generated equilibrium-finding programs, in: Proceedings of the International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS), 2008, pp. 911–918.
[32] A. Girard, A. Howell, J.K. Hedrick, Border patrol and surveillance missions using multiple unmanned air vehicles, in: Proceedings of the IEEE Conference on Decision and Control (CDC), 2004, pp. 620–625.
[33] A. Glad, O. Simonin, O. Buffet, F. Charpillet, Theoretical study of ant-based algorithms for multi-agent patrolling, in: Proceedings of European Conference on Artificial Intelligence (ECAI), 2008, pp. 626–630.
[34] D. Gulczynski, B. Golden, E. Wasil, The period vehicle routing problem: New heuristics and real-world variants, Transportation Research Part E: Logistics and Transportation Review 47 (5) (2011) 648–668.
[35] Y. Guo, L. Parker, R. Madhavan, Collaborative robots for infrastructure security applications, in: Mobile Robots: The Evolutionary Approach, in: Book Series on Intelligent Systems Engineering, vol. 50, 2007, pp. 185–200.
[36] A. Howard, N. Roy, The robotics data set repository (radish), 2003.
[37] ILOG CP, http://www.ilog.com/products/cp/.
[38] V. Isler, S. Kannan, S. Khanna, Randomized pursuit–evasion in a polygonal environment, IEEE Transactions on Robotics 5 (21) (2005) 864–875.
[39] M. Jain, J. Pita, M. Tambe, F. Ordóñez, P. Paruchuri, S. Kraus, Bayesian Stackelberg games and their application for security at Los Angeles International Airport, SIGecom Exchanges 7 (2) (2008).
[40] C. Kiekintveld, M. Jain, J. Tsai, J. Pita, F. Ordóñez, M. Tambe, Computing optimal randomized resource allocations for massive security games, in: Proceedings of the International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS), 2009, pp. 689–696.
[41] A. Kolen, A. Kan, H. Trienekens, Vehicle routing with time windows, Operations Research 35 (2) (1987) 266–273.
[42] D. Koller, N. Megiddo, B. von Stengel, Efficient computation of equilibria for extensive two-person games, Games and Economic Behavior 14 (2) (1996) 220–246.
[43] A. Kolling, S. Carpin, Extracting surveillance graphs from robot maps, in: Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS), 2008, pp. 2323–2328.
[44] D. Kreps, R. Wilson, Sequential equilibria, Econometrica 50 (4) (1982) 863–894.
[45] J. Letchford, V. Conitzer, Computing optimal strategies to commit to in extensive-form games, in: Proceedings of the ACM Conference on Electronic Commerce (EC), 2010, pp. 83–92.
[46] A. Machado, G. Ramalho, J.-D. Zucker, A. Drogoul, Multi-agent patrolling: An empirical analysis of alternative architectures, in: Proceedings of the Third International Workshop on Multi-Agent-Based Simulation (MABS2002), vol. 2581, 2003, pp. 155–170.

[47] J.S. Marier, C. Besse, B. Chaib-draa, Solving the continuous time multiagent patrol problem, in: Proceedings of IEEE International Conference on Robotics and Automation (ICRA), 2010, pp. 941–946.
[48] L. Martins-Filho, E. Macau, Patrol mobile robots and chaotic trajectories, in: Mathematical Problems in Engineering, Hindawi, 2007.
[49] C. Papadimitriou, Computational Complexity, Addison–Wesley, 1993.
[50] P. Paruchuri, J. Pearce, J. Marecki, M. Tambe, F. Ordonez, S. Kraus, Playing games for security: An efficient exact algorithm for solving Bayesian Stackelberg games, in: Proceedings of the International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS), 2008, pp. 895–902.
[51] P. Paruchuri, J. Pearce, M. Tambe, F. Ordonez, S. Kraus, An efficient heuristic approach for security against multiple adversaries, in: Proceedings of the International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS), 2007, pp. 311–318.
[52] J. Pita, M. Jain, J. Marecki, F. Ordonez, C. Portway, M. Tambe, C. Western, P. Paruchuri, S. Kraus, Deployed ARMOR protection: The application of a game theoretic model for security at the Los Angeles International Airport, in: Proceedings of the International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS), 2010, pp. 125–132.
[53] B. Raa, E. Aghezzaf, A practical solution approach for the cyclic inventory routing problem, European Journal of Operational Research 192 (2) (2009) 429–441.
[54] S. Ruan, C. Meirina, F. Yu, K. Pattipati, R. Popp, Patrolling in a stochastic environment, in: Proceeding of the International Command and Control Research and Technology Symposium (CCRTS), 2005.
[55] A. Rubinstein, Perfect equilibrium in a bargaining model, Econometrica 50 (1) (1982) 97–109.
[56] W. Ruckle, R. Fennel, P.T. Holmes, C. Fennemore, Ambushing random walk. I: Finite models, Operations Research 24 (1976) 314–324.
[57] S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, third ed., Prentice Hall, 2010.
[58] T. Sak, J. Wainer, S.K. Goldenstein, Probabilistic multiagent patrolling, in: Proceedings of the Brazilian Symposium on Artificial Intelligence (SBIA), 2008, pp. 124–133.
[59] H. Santana, G. Ramalho, V. Corruble, B. Ratitch, Multi-agent patrolling with reinforcement learning, in: Proceedings of the Third International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS), 2004, pp. 1120–1127.
[60] Y. Shoham, K. Leyton-Brown, Multiagent Systems: Algorithmic, Game Theoretic and Logical Foundations, Cambridge University Press, 2008.
[61] Stanford Business Software Inc., http://www.sbsi-sol-optimize.com/.
[62] J. Tsai, Z. Yin, J.-Y. Kwak, D. Kempe, C. Kiekintveld, M. Tambe, How to protect a city: Strategic security placement in graph-based domains, in: Proceedings of the International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS), 2010, pp. 1453–1454.
[63] J. Tsitsiklis, Special cases of traveling salesman and repairman problems with time windows, Networks 22 (3) (1992) 263–282.
[64] B. von Stengel, S. Zamir, Leadership with commitment to mixed strategies, CDAM Research Report LSE-CDAM-2004-01, London School of Economics, 2004.
[65] A. Wahsburn, K. Wood, Two-person zero-sum games for network interdiction, Operations Research 43 (2) (1995) 243–251.
[66] V. Yanovski, I. Wagner, A. Bruckstein, A distributed ant algorithm for efficiently patrolling a network, Algorithmica 37 (2003) 165–186.