Solving Patrolling Problems in the Internet Environment

Tomáš Brázdil, Antonín Kučera and **Vojtěch Řehák***
Faculty of Informatics, Masaryk University, Brno, Czech Republic
{brazdil,kucera,rehak}@fi.muni.cz

Abstract

We propose an algorithm for constructing efficient patrolling strategies in the Internet environment, where the protected targets are nodes connected to the network and the patrollers are software agents capable of detecting/preventing undesirable activities on the nodes. The algorithm is based on a novel compositional principle designed for a special class of strategies, and it can quickly construct (sub)optimal solutions even if the number of targets reaches hundreds of millions.

1 Introduction

A *security game* is a non-cooperative game where the Defender (leader) commits to some strategy and the Attacker (follower) first observes this strategy and then selects a best response. In *adversarial* security games, it is assumed that the Attacker not only knows the Defender's strategy, but can also observe the current positions and moves of the patrollers. This worst-case assumption is adequate also in situations when the actual Attacker's abilities are *unknown* and robust defending strategies are required.

In this paper, we concentrate on adversarial patrolling in the *Internet environment*, where the protected targets are fully connected by a network, and the patrollers are software agents freely moving among the targets trying to discover/prevent dangerous ongoing activities. We start by presenting two concrete scenarios[1] illustrating the considered class of security problems.

1. Large-scale surveillance systems. Contemporary surveillance systems may comprise thousands (or even millions[2]) of cameras watching complex scenes where real-time detection and alert are crucial. For example, crime detection systems assume fast response in case of crime or suspect detected. Also, they typically need sophisticated and computation intensive analytics (see, e.g., [Cotton, 2015]) such as object detection, database retrieval of image data, etc. Simple object detection tools running on high-end GPUs are capable of processing hundreds full scale images per second [Redmon *et al.*, 2016]. More detailed analysis, such as search of a detected face in a large database of suspects, causes even smaller number of processed images per second. This means that the delay caused by the analytics may prevent *simultaneous* real-time analysis of *all* video streams in a large-scale surveillance system, and the system must intelligently switch among the streams in real-time. Since the image processing time is substantially shorter than the intrusion time, there is a chance of achieving a good level of protection even if the system runs only a limited number of analytical processes concurrently. The crucial question is how to schedule the "visits" of these processes (patrollers) to the individual cameras (targets) so that the chance of discovering an ongoing intrusion is maximized. Since the analytics is not perfect, a patroller detects an ongoing intrusion in a currently visited target only with certain probability. Hence, the chance of successful intrusion detection increases if the target under attack is visited repeatedly before completing the attack.

2. Remote software protection. Protecting software from man-at-the-end (MATE) tampering is a hard problem in general. In client-server systems, where the server part is considered trusted, *continuous software updates* of the client software have been proposed as a promising technique for achieving the protection [Ceccato and Tonella, 2011; Collberg *et al.*, 2012]. Since completing a MATE attack requires a substantial amount of time and effort, the idea is to update some crucial components of the client software regularly (by the trusted server) so that the malicious ongoing analysis becomes useless and it must be restarted. If these updates are performed frequently enough, a MATE attack cannot be completed. However, the server's capacity is limited and the number of clients is typically very large (especially when the clients are split into independent submodules to decrease the update overhead). The question is how to design a suitable update policy for the server achieving a good protection against a MATE actively seeking for a weakly protected

*Supported by grant No. P202/12/G061, Czech Science Foundation. Access to computing and storage facilities owned by parties and projects contributing to the National Grid Infrastructure MetaCentrum provided under the programme "Projects of Large Research, Development, and Innovations Infrastructures" (CESNET LM2015042), is greatly appreciated.

[1]These scenarios should be seen just as *examples* of possible application areas for our results, not as an exhaustive list.

[2]According to IHS, there were 245 million professionally installed video surveillance cameras active and operational globally in 2014.

client. A detailed discussion of all relevant aspects can be found in [Basilico *et al.*, 2016a] where a *security game model* of the problem is designed. The patrollers are the update processes managed by the server, and the targets are the client software modules. Each target is assigned the time needed to complete a MATE attack and another natural number specifying its *importance*. It is assumed that performing an update takes a constant time. Although the patrollers cannot detect an ongoing MATE attack at the currently updated target, it is assumed that a possible ongoing attack is always interrupted by the update. Furthermore, it is shown how to compute a *positional* Defender's strategy for updating the targets, where the Defender's decisions depend only on the current positions (i.e., the tuple of currently visited modules) of the patrollers. In [Basilico *et al.*, 2016a], it is explicitly mentioned[3] that positional strategies are *weaker* than general *history-dependent* strategies taking into account the whole history of previous updates. Hence, positional strategies are generally not optimal, and the computational framework of [Basilico *et al.*, 2016a] does not allow for efficient construction of history-dependent strategies. This limitation is overcome in the presented work (see below).

Our contribution. Before explaining our results, let us briefly summarize the key *assumptions* about the considered class of problems which are reflected in the adopted game model (see Section 2):

1. *The environment is fully connected*. This is the basic property of Internet underpinning our (novel) approach to strategy synthesis based on game decomposition. The use of (non)linear programming is completely avoided.

2. *The patrollers are centrally coordinated.* The patrollers are software processes fully controlled by a server.

3. *The probability p of recognizing an ongoing intrusion at a currently visited target is not necessarily equal to* 1. That is, the intrusion detection is not fully accurate in general, as in Scenario 1. Note that is Scenario 2, an intrusion (MATE attack) is recognized (interrupted) with probability 1.

4. *The time needed to complete an intrusion depends on a concrete target, and targets may have varying importance.*

5. *The time needed to complete a patroller's activity is almost constant*[4]. This is satisfied in both scenarios.

6. *The number of targets is very large*. This influences mainly the methodology of algorithmic solutions. Algorithms for solving security games are mostly based on mathematical programming (see *Related work*). This approach becomes *infeasible* in our setting where the number of targets can easily reach a billion.

[3] A rigorous proof revealing insufficiency of positional strategies can be found in [Kučera and Lamser, 2016].

[4] Technically, this means that time is random but strongly concentrated around its expected value. Note that in Scenario 1, the real-time detection consists of two phases where a given image is first quickly classified as either harmless or potentially dangerous (in almost constant time), and dangerous images are subsequently enqueued for a more advanced analysis, such as face recognition in a database of suspects. This queue is processed separately (possibly using special hardware), and hence the second phase does not influence the assignment of patrolling processes.

We adopt the adversarial setting, i.e., assume that the Attacker knows the Defender's strategy (the server program assigning the patrolling processes to targets) and can also determine the targets currently visited (e.g., by analyzing the network traffic or deploying malware into the server). Since there are no principal bounds on the duration of the patrolling task, our games are of infinite horizon. The adopted solution concept is *Stackelberg equilibrium* (see, e.g, [Yin *et al.*, 2010]) where the Defender/Attacker corresponds to the leader/follower.

Our **main results** can be summarized as follows:

A. We give an *upper bound* on the level of protection achievable for a given game structure and a given number of patrollers. Consequently, we can also derive a *lower bound* on the number of patrollers needed to achieve a given level of protection. These bounds are valid for *general* (i.e., history-dependent and randomized) strategies. They are generally not tight, but good enough to serve as a "yardstick" for measuring the quality of the constructed Defender's strategies.

B. We develop a novel *compositional* approach to constructing Defender's strategies in Internet patrolling games. The method is based on splitting a given game into disjoint subgames, solving them recursively, and then combining the obtained solutions into a strategy for the original game. We evaluate the quality of the constructed strategies against the bounds described in A., and show that our algorithm produces *provably* (sub)optimal solutions. A precise formulation is given in Section 5.1.

The running time of our algorithm is low. Instances with millions of targets are processed in units of seconds (see Section 5.1). The only potentially costly part is solving a certain system of polynomial equations constructed by the algorithm, but this was always achieved in less than a second for all instances we analyzed (using Maple). The constructed strategies are randomized and history-dependent. We call them *modular* because they use a bounded counter to count the units of elapsed time modulo certain constant. Hence, modular strategies are still easy to implement.

One may also ask whether some "naive" strategy synthesis method can produce strategies of comparable quality (i.e., whether our decomposition method is really worth the invested effort). Perhaps, the most straightforward way of constructing *some* reasonable Defender's strategy is to compute a positional strategy where the probability of selecting a given vertex depends only on its importance and attack length, and it is chosen so that all vertices are protected equally well. We demonstrate that the strategies computed by our algorithm are *substantially* better than these naively constructed ones.

Due to space constraints, we defer proofs, examples, and some additional experimental results to the full version of this paper [Brázdil *et al.*, 2018].

Related work. Most of the existing works about security games study either the problem of computing an optimal static allocation of available resources to the targets, or the problem of computing an optimal movement strategy for a mobile Defender. Security games with static allocation have been studied in, e.g., [Jain *et al.*, 2010; Kiekintveld *et al.*, 2009; Pita *et al.*, 2008; Tsai *et al.*, 2009].

In patrolling games, the focus was primarily on finding locally optimal strategies for robotic patrolling units either on restricted graphs such as circles [Agmon *et al.*, 2008a; 2008b], or arbitrary graphs with weighted preference on the targets [Basilico *et al.*, 2009a; 2009b]. Alternatively, the work focused on some novel aspects of the problem, such as variants with moving targets [Bosansky *et al.*, 2011; Fang *et al.*, 2013], multiple patrolling units [Basilico *et al.*, 2010], or movement of the Attacker on the graph [Basilico *et al.*, 2009b] and reaction to alarms [de Cote *et al.*, 2013; Basilico *et al.*, 2016b]. Most of the existing literature assumes that the Defender is following a positional strategy that depends solely on the current position of the Defender in the graph and they seek for a solution using mathematical programming [Basilico *et al.*, 2012]. Few exceptions include duplicating each node of the graph to distinguish internal states of the Defender (e.g., in [Agmon *et al.*, 2008a] authors consider a direction of the patrolling robot as a specific state; in [Bosansky *et al.*, 2012], this concept is further generalized), or seeking for higher-order strategies in [Basilico *et al.*, 2009a]. In [Abaffy *et al.*, 2014], an exponential-time algorithm for computing an ε-optimal strategy for the Defender is designed. The existing works on multi-agent patrolling mostly assume autonomous patrollers without a central supervision (see, e.g., [Almeida *et al.*, 2004]).

A patrolling game model for remote software protection has been proposed in [Basilico *et al.*, 2016a]. The model is similar to ours, but the probability of detecting (interrupting) an ongoing intrusion is set to 1, which simplifies the strategy synthesis (repeated visits to the same target during an ongoing intrusion do not increase its protection). The constructed Defender's strategies are positional and computed by standard methods based on mathematical programming, which limits their scalability and does not lead to optimal solutions.

Continuous-time adversarial patrolling games with one patroller in fully connected environment have recently been studied in [Kempe *et al.*, 2018]. Based on the Defender's strategy, the Attacker selects which vertex to attack and for how long. The Attacker expected utility grows linearly with the time spent in the vertex but drops to 0 if being caught. The Defender's goal is to minimize the Attacker's utility. This setup is technically different from ours, although it is also motivated by possible applications in Internet security problems (in [Kempe *et al.*, 2018], no concrete scenarios illustrating the applicability of the considered model are given).

2 The Game Model

In this section we present our game-theoretic model of adversarial patrolling in the Internet environment reflecting Assumptions 1.-6. formulated in Section 1.

Preliminaries. We use $\mathbb{N}_0$ and $\mathbb{N}$ to denote the sets of non-negative and positive integers, respectively. The set of all probability distributions over a finite set M is denoted by $\Delta(M)$. The lower and upper integer approximations of a real number a are written as $\lfloor a \rfloor$ and $\lceil a \rceil$, respectively. The number of elements of a set A is denoted by $|A|$. A k-subset of A is a subset of A with precisely k elements, and we use $A^{\langle k \rangle}$ to denote the set of all k-subsets of A. For $f : A \to B$ and $X \subseteq A$, we use $f|_X$ to denote the restriction of f to X.

Game structures. A *game structure* is a tuple $\mathcal{G} = (V, d, \alpha, p)$, where V is a finite set of *vertices* (targets), $d : V \to \mathbb{N}$ specifies the number of time units needed to complete an intrusion at a given vertex, $\alpha : V \to \mathbb{N}$ is a *cost* function specifying the importance of each vertex (a higher number means higher importance), and $p \in (0, 1]$ is the probability of discovering an ongoing intrusion by a patroller visiting a vertex under attack. We assume that a patroller spends one unit of time when moving from vertex to vertex, which corresponds to performing the patroller's activity at the previously visited vertex.

Defeder's strategy. Assume $\mathcal{G}$ is protected by $k \in \mathbb{N}$ patrollers (where $k \leq |V|$) centrally coordinated by the Defender who has a complete knowledge about the history of previously visited vertices. Based on the history, the Defender selects a k-subset of vertices where the patrollers are sent in the next round, and this decision can be randomized.

Formally, a *Defender's strategy* is a function η assigning to every history $U_1, \ldots, U_\ell$, where $\ell \geq 0$ and U_i is the k-subset of vertices visited in round i, a probability distribution over $V^{\langle k \rangle}$. Note that a k-subset of vertices visited in the first round is determined by $\eta(\varepsilon)$, where ε is the empty history. A strategy η is *positional* if $\eta(U_1, \ldots, U_\ell)$ depends only on U_ℓ.

Each strategy η determines a unique probability space over all *walks*, i.e., infinite sequences $U_1, U_2, \ldots$ where $U_i \in V^{\langle k \rangle}$ for all $i \in \mathbb{N}$, in the standard way.

Attacker's strategy. Depending on the observed history of visited vertices, the Attacker may choose to attack some vertex or wait. Formally, an *Attacker's strategy* is a function π assigning to every history an element of $V \cup \{\bot\}$ such that whenever $\pi(U_1, \ldots, U_\ell) \neq \bot$, then for all $j < \ell$ we have that $\pi(U_1, \ldots, U_j) = \bot$, i.e., the Attacker may attack at most once along a walk.

Level of protection. The aim of the Attacker is to maximize the *expected damage*, i.e., the expected cost of a successfully attacked target, and the Defender aims at the opposite. Let us fix an Attacker's strategy π, and let $w = U_1, U_2, \ldots$ be a walk. The damage achieved by π in w, denoted by $Damage^\pi(w)$, is defined as follows:

- If the Attacker does not attack along w, i.e., $\pi(U_1, \ldots, U_\ell) = \bot$ for all $\ell \in \mathbb{N}_0$, then $Damage^\pi(w)$=0.
- Otherwise, there is $\ell \in \mathbb{N}_0$ such that $\pi(U_1, \ldots, U_\ell) = v$, where $v \in V$ is the attacked vertex. For a given $i \in \{1, \ldots, d(v)\}$, we say that v is *visited in round* i (since the moment of initiating the attack) if $v \in U_{\ell+i}$. For each such i, the probability of discovering the ongoing attack is p. The attack is successful if it remains undiscovered after *all* visits to v in the next $d(v)$ rounds (the individual trials are considered independent). That is, the probability of performing the attack successfully is equal to $(1-p)^c$, where c is the total number of all $i \in \{1, \ldots, d(v)\}$ such that v is visited in round i. We put

$$Damage^\pi(w) \;=\; (1-p)^c \cdot \alpha(v)$$

to reflect the importance of the attacked vertex v (if $p = 1$ and $c = 0$, we put $Damage^\pi(w) = \alpha(v)$).

Let η be a Defender's strategy for k patrollers and π an Attacker's strategy. The *expected damage* caused by π when the Defender commits to η, denoted by $\mathbb{E}^{\eta}[Damage^{\pi}]$, is the expected value of $Damage^{\pi}$ in the probability space over the walks determined by η (see above). Note that $\mathbb{E}^{\eta}[Damage^{\pi}] \leq \alpha_{\max}$, where $\alpha_{\max}$ is the maximal cost assigned to a vertex of $\mathcal{G}$.

In Section 1, we used the term "protection" instead of "damage", and said that the Defender aims at maximizing the protection rather than minimizing the damage. This original terminology seems more intuitive, and it will be used also in the rest of this paper. Formally, the *level of protection* achieved by η against π is defined as

$$Lev(\eta, \pi) \;=\; \alpha_{\max} - \mathbb{E}^{\eta}[Damage^{\pi}]\,.$$

Note that maximizing the level of protection is *equivalent* to minimizing the expected damage.

Furthermore, the level of protection achieved by η (against *any* π) is defined by $Lev(\eta) = \inf_{\pi} Lev(\eta, \pi)$. Finally, the maximal level of protection achievable with k patrollers is defined as $Lev_k = \sup_{\eta} Lev(\eta)$, where η ranges over all Defender's strategies for k patrollers. The underlying $\mathcal{G}$ will always be clearly determined by the context. A Defender's strategy η for k patrollers is *δ-optimal*, where $\delta \geq 0$, if $Lev(\eta) \geq Lev_k - \delta$. A 0-optimal strategy is called *optimal*.

3 The Bounds

In this section we give an upper bound on Lev_k and a lower bound on the number of patrollers needed to achieve a given level of protection. These bounds are not always tight, but good enough for evaluating the efficiency of Defender's strategies computed by the algorithm of Section 5.

The bounds are obtained by solving a non-trivial system of exponential equations constructed for a given game structure. In general, the solution can only be computed by numerical methods, which is fully sufficient for our purposes. Contemporary mathematical software such as Maple can perform the required computations very efficiently even if for high parameter values.

Let $\mathcal{G} = (V, d, \alpha, p)$ be a game structure, and $\alpha_{\max}$ the maximal cost assigned to a vertex of $\mathcal{G}$. Furthermore, let ϱ be a fresh variable. For every $v \in V$, we construct the equation

$$\varrho = \alpha_{\max} - \alpha(v) \cdot (1-p)^{\lfloor Q_v \rfloor} \cdot (1 - p \cdot (Q_v - \lfloor Q_v \rfloor))$$

where Q_v is a fresh variable (if $p = 1$, the equation is simplified into $\varrho = \alpha_{\max} - \alpha(v) \cdot (1 - Q_v)$). Let $\mathcal{L}_{\mathcal{G}}$ be the resulting system of equations. Note that $\mathcal{L}_{\mathcal{G}}$ has $|V|$ equations and $|V| + 1$ variables.

Theorem 1. *Let $\mathcal{G} = (V, d, \alpha, p)$ be a game structure.*

a) The level of protection achievable with a given number of patrollers k is bounded from above by ϱ obtained by solving the system $\mathcal{L}_{\mathcal{G}}$ extended with the equation $k = \sum_{v \in V, Q(v) > 0} Q_v / d(v)$.

b) Let $\tau \geq 0$ be a desired level of protection. Consider the system of equations obtained by extending $\mathcal{L}_{\mathcal{G}}$ with the equation $\tau = \varrho$. If this system has no solution such that $Q_v \leq d(v)$ for all $v \in V$, then the level of protection τ is not achievable for an arbitrarily large number of patrollers (if $p = 1$, the condition is restricted to $Q_v \leq 1$). Otherwise, the number of patrollers needed to achieve the level of protection τ is bounded from below by $\left\lceil \sum_{v \in V, Q(v) > 0} Q_v / d(v) \right\rceil$, where the value of each Q_v is obtained by solving the system.

A proof of Theorem 1 is non-trivial and it is based on a careful analysis of "uniform coverage" of all vertices with k-patrollers reflecting the importance of individual vertices. The details are omitted due to the lack of space. Let us note that when solving the systems considered in Theorem 1, it may happen that some Q_v's become negative. This happens if (and only if) the importance of some vertices is so low that even if the patrollers do not visit them at all, they are still protected better than the other vertices (this also explains why the sum $\sum_{v \in V, Q(v) > 0} Q_v / d(v)$ is taken only over positive $Q(v)$'s). The system of Theorem 1(b) may have no eligible solution in situations when p is so small that the protection τ is not achievable even if each v is visited with probability one in every step. If $p = 1$, no eligible solution is induced only by $\tau > \alpha_{\max}$.

4 Modular Strategies

In this section, we introduce *modular strategies* and the associated *compositional principle* which are the cornerstones of our strategy synthesis algorithm.

Definition 1. *Let $\mathcal{G} = (V, d, \alpha, p)$ be a game structure, and $c_{\mathcal{G}}$ the least common multiple of all $d(v)$ where $v \in V$. A Defender's strategy η for $\mathcal{G}$ is* modular *if for every $h \in \mathcal{H}$, the distribution $\eta(h)$ depends only on $\ell \bmod c_{\mathcal{G}}$, where ℓ is the length of h.*

Hence, a modular strategy "ignores" the precise structure of a history and takes into account only $\ell \bmod c_{\mathcal{G}}$. At first glance, this looks like a severe restriction substantially limiting the efficiency of modular strategies. Surprisingly, this intuition turns out to be largely incorrect. As we shall see in Section 5.1, the level of protection achievable by modular strategies approaches and in some cases even *matches* the upper bound of Theorem 1 which proves their *(sub)optimality* among *general* strategies. Also note that modular strategies require only a bounded counter as auxiliary memory, which makes them easy to implement. To simplify our notation, we formally consider modular strategies as functions $\eta : \mathbb{N}_0 \to \Delta(V^{\langle k \rangle})$, where $\eta(\ell)$ depends just on $\ell \bmod c_{\mathcal{G}}$.

The main advantage of modular strategies is their *compositionality*. A modular strategy for a given game structure $\mathcal{G}$ can be constructed by decomposing $\mathcal{G}$ into pairwise disjoint substructures $\mathcal{G}_1, \ldots, \mathcal{G}_m$, computing modular strategies for these substructures recursively, and then combining them into a modular strategy for $\mathcal{G}$. The algorithm presented in Section 5 works in this way. Now we describe the composition step in greater detail.

Let $\mathcal{G} = (V, d, \alpha, p)$, and let k be the number of patrollers protecting $\mathcal{G}$. Assume that we already managed to decompose $\mathcal{G}$ into smaller disjoint substructures $\mathcal{G}_1, \ldots, \mathcal{G}_m$, where $\mathcal{G}_i = (V_i, d|_{V_i}, \alpha|_{V_i}, p)$, so that, for all $n \leq k$ and $i \leq m$,

an efficient modular strategy $\eta_i[n]$ for $\mathcal{G}_i$ and n patrollers together with $Lev(\eta_i[n])$ can already be efficiently computed. In order to combine the constructed strategies into a strategy for $\mathcal{G}$, we need to solve the following problems:

- How to assign the k available patrollers to $\mathcal{G}_1, \ldots, \mathcal{G}_m$?
- How to compose the modular strategies constructed for $\mathcal{G}_1, \ldots, \mathcal{G}_m$ into a modular strategy η for $\mathcal{G}$?

Assigning patrollers to substructures. Clearly, the number of patrollers assigned to $\mathcal{G}_i$ should not exceed $|V_i|$. It is also clear that if k is smaller than m, the assignment cannot be deterministic, because otherwise some substructures would remain completely unprotected, and the Attacker could attack them without any risk (recall the Attacker knows the Defender's strategy). So, we need to assign the k patrollers to $\mathcal{G}_1, \ldots, \mathcal{G}_m$ *randomly* in general. Formally, an *assignment* for k patrollers and $\mathcal{G}_1, \ldots, \mathcal{G}_m$ is a probability distribution β over the set of all *eligible* allocations $\vec{\kappa} \in \mathbb{N}_0^m$, where the components of $\vec{\kappa}$ sum up to k and $\vec{\kappa}_i \leq |V_i|$ for all $i \leq m$.

Composing strategies constructed for substructures. Assume we already constructed a suitable assignment β for k patrollers and $\mathcal{G}_1, \ldots, \mathcal{G}_m$. By our assumption, for all $n \leq k$ and $i \leq m$, an efficient modular strategy $\eta_i[n]$ for $\mathcal{G}_i$ and n patrollers can already by constructed. Our task is to construct a suitable modular strategy η for $\mathcal{G}$ and k patrollers. For every $\ell \in \mathbb{N}_0$, the outcome of $\eta(\ell)$ is determined as follows:

- First, some eligible allocation $\vec{\kappa} \in \mathbb{N}_0^m$ is selected randomly according to β (independently of ℓ).
- Then, for each $i \in \{1, \ldots, m\}$, we independently select a $\vec{\kappa}_i$-subset U_i of V_i according to $\eta_i[\vec{\kappa}_i](\ell)$. Thus, we obtain a k-subset $U_1 \cup \cdots \cup U_m$ of V, which is the (random) outcome of $\eta(\ell)$.

Observe that the above definition makes a good sense because all $\eta_i[\vec{\kappa}_i]$ are *modular* strategies[5].

Hence, a concrete strategy synthesis algorithm based on the introduced decomposition principle (such as the one presented in Section 5) must implement the following:

- a *decomposition procedure* which, for given k and $\mathcal{G}$, either decomposes $\mathcal{G}$ into disjoint substructures to be solved recursively, or computes a suitable modular strategy for $\mathcal{G}$ and k patrollers;
- an *assignment procedure* which, for given k and $\mathcal{G}_1, \ldots, \mathcal{G}_m$, computes an assignment for k patrollers and $\mathcal{G}_1, \ldots, \mathcal{G}_m$.

These procedures may reflect different decomposition tactics apt for specific classes of instances.

5 A Strategy Synthesis Algorithm

In this section we design and evaluate a concrete strategy synthesis algorithm based on the compositional method presented in Section 4. This algorithm is particularly apt for game structures where large subsets of vertices share the same attack length and the same importance weight. This applies to, e.g., surveillance systems and remote software protection systems[6] discussed in Section 1.

Decomposition procedure. For a given $\mathcal{G} = (V, d, \alpha, p)$, the decomposition starts by splitting the vertices of V into pairwise disjoint subsets $U_{D,\tau}$ consisting of all $v \in V$ such that $d(v) = D$ and $\alpha(v) = \tau$ (for all D's and τ's in the range of d and α, respectively). Then, each $U_{D,\tau}$ is further split into $\lceil |U_{D,\tau}|/D \rceil$ pairwise disjoint subsets of size precisely D (these are called *full*), and possibly one extra set with $(U_{D,\tau} \bmod D)$ elements (if D does not divide $|U_{D,\tau}|$). These sets, constructed for all eligible D's and τ's, are called the *basic sets of* $\mathcal{G}$, and they are not decomposed any further. Note that the decomposition of $\mathcal{G}$ into basic sets is independent of the number of patrollers assigned to protect $\mathcal{G}$.

For every basic set $U = \{v_0, \ldots, v_{q-1}\}$ and every k, we need to compute a modular strategy μ for U and k patrollers. Recall that all vertices of U have the same importance τ and the same attack length D, where $q \leq D$. Let us first consider the simpler case when q divides D. Imagine there are k tokens, initially put on the vertices $v_0, \ldots, v_{k-1}$, which are then moved simultaneously around a circle formed by the vertices of U, where the successor of v_i is $v_{(i+1) \bmod D}$. That is, after ℓ steps, the first token resides at $v_{\ell \bmod D}$, and the other tokens reside at the next $k-1$ vertices of the circle. The strategy $\mu(\ell)$ deterministically selects the k-tuple of vertices occupied by the tokens after ℓ steps. Note that each token visits each vertex precisely D/q times in D consecutive steps.

Now consider the general case when q does not necessarily divide D. In the first $\lfloor D/q \rfloor \cdot q$ steps, μ simulates k tokens moving simultaneously around a circle similarly as above. In the remaining (D mod q) steps, a k-subset of U is chosen uniformly at random (independently in each step). Formally, for every $\ell \in \{0, \ldots, D-1\}$, we have the following:

- if $\ell < \lfloor D/q \rfloor \cdot q$, then $\mu(\ell)$ returns $\{v_{j_0}, \ldots, v_{j_{k-1}}\}$ with probability one, where $j_n = (\ell + n) \bmod q$ for all $n \in \{0, \ldots, k-1\}$;
- otherwise, $\mu(\ell)$ returns a k-subset of U chosen uniformly at random.

Assignment procedure. This is the most advanced part of our algorithm. We need to construct an assignment β for k patrollers and the basic sets. We aim to construct β so that the number of patrollers assigned to a given basic set U is either K_U or $K_U + 1$, where $K_U \in \mathbb{N}_0$ is a suitable constant depending only on k, the attack length, importance level, and the number of vertices of U. The reason is that a wider variability in the number of patrollers assigned to U would actually *decrease*[7] the protection achieved for basic sets.

Now we explain how to compute the constant K_U and the probability λ_U of sending precisely $K_U + 1$ patrollers to U.

[5] If $\eta_i[\vec{\kappa}_i]$ were general strategies, they could not be combined in such a simple way, because in each step, a different number of patrollers (including zero) can be assigned to a given $\mathcal{G}_i$, and hence the history produced by η would not have to contain a valid history of $\eta_i[\vec{\kappa}_i]$. So, it would not be clear how to simulate $\eta_i[\vec{\kappa}_i]$.

[6] The number of IP camers or software modules can easily reach hundreds of millions, while the time needed to complete an intrusion or a software update typically ranges over a small interval of discretized time values. Similarly, the importance of a target usually ranges over a small set of discrete levels.

[7] This claim follows from the structure of modular strategies constructed for basic sets, and it can be proven rigorously.

Let us fix a basic set $U = \{v_1, \ldots, v_q\}$ where $\alpha(v_i) = \tau$ and $d(v_i) = D$ for all $v_i \in U$. If β sends K_U patrollers to U with probability $(1 - \lambda_U)$ and $K_U + 1$ patrollers with probability λ_U, then the expected number of patrollers sent to U, denoted by E_U, is equal to $K_U + \lambda_U$. Furthermore, the protection achieved for the vertices of U is given by:

$$\alpha_{\max} - \tau \cdot (1-p)^{K_U \cdot \lfloor D/q \rfloor} \cdot (1 - p \cdot \lambda_U)^{\lfloor D/q \rfloor} \cdot (1 - p \cdot (K_U + \lambda_U)/q)^{(D \bmod q)} \quad (1)$$

Expression (1) is obtained by a straightforward calculation omitted in here (possible subexpressions of the form 0^0 are interpreted as 1). Since we wish to protect all basic sets equally well, the above expression should produce the *same* value for all $U \in W$. Hence, we can consider a system of equations stipulating that the above expressions are equal for all U, and the sum of all positive E_U's is equal to k. The values for K_U and λ_U are obtained by solving this system. Observe that (1) is parameterized by *two* unknowns K_U and λ_U, so we have more unknowns than equations. This is overcome by observing that $K_U = \lfloor E_U \rfloor$ and $\lambda_U = E_U - \lfloor E_U \rfloor$, which allows to use just *one* variable E_U instead of K_U and λ_U. Furthermore, it may happen that a solution contains some negative E_U's, which indicates that the importance of the vertices in U is so low that U is protected better than the other basic sets even if zero patrollers are assigned to U. In this case, we simply remove U from $\mathcal{G}$ and restart the algorithm.

5.1 Evaluating the Constructed Strategies

For every Defender's strategy η for k patrollers, the *relative deviation of η from optimal strategy*, denoted by $Dev(\eta)$, is defined by $(Lev_k - Lev(\eta))/Lev(\eta)$.

First, let us note that when all basic sets of $\mathcal{G}$ are full, then the achieved level of protection matches the upper bound of Theorem 1, i.e., the strategy constructed by our algorithm is *provably optimal*. For every game structure $\mathcal{G}$, there are two "surrounding" game structures obtained by decreasing/increasing the number of vertices in each $U_{D,\tau}$ to the nearest multiple of D, for which our algorithm produces optimal strategies. With an increasing number of vertices, the levels of protection achieved for the surrouding game structures are closer and closer, which implies that $Dev(\eta)$ approaches zero. For a given $\delta > 0$, one can easily compute a threshold such that, for every game structure where the number of vertices exceeds the threshold, the strategy η constructed by our algorithm satisfies $Dev(\eta) < \delta$. Hence, our algorithm *provably* produces strategies which are either optimal or very close to optimal, and this claim does *not* require further experimental evidence. The aim of our experiments is to demonstrate that the time needed to solve the constructed system of equations is low (using Maple), and also to show that the protection achieved by our strategies is substantially better than the protection achieved by naively constructed strategies.

More concretely, we consider a surveillance system with three types of targets, where processing one image takes 0.1 sec, an on-going intrusion is detected with probability 0.7, the time needed to complete an intrusion is either 20 secs, 2 mins, or 15 mins, and the target values are \$100000, \$130000, and \$400000, respectively. We analyze large instances where the number of targets of each type is $7000000 \cdot x$, $500000 \cdot x$, and $300000 \cdot x$, respectively, where the x ranges from 1 to 3 with a step 0.01. Hence, the total number of analyzed game structures is 200 and the largest one has more than 23 million vertices. The number of patrollers is set to 6000. Since the time needed to compute η was always negligible (the constructed systems of equations were solved by Maple in less than a second on an average PC), we do not report the running time details. The levels of protection achieved by η strategies differ from the theoretical upper bound by less than one dollar (for all instances). When we compare these strategies against naively constructed strategies σ, where in each round, the patrollers are identically distributed among the vertices so that all vertices are protected equally well, the difference between the levels of protection achieved by σ strategies and the theoretical upper bound ranges between \$157 and \$740.

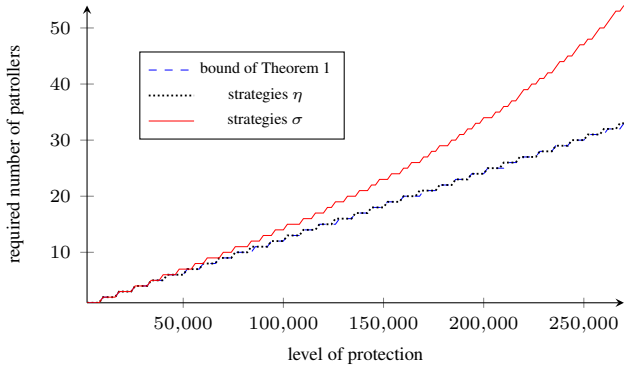

Figure 1: Achieving a given level of protection.

In Fig. 1, we show the number of patrollers needed to achieve a given level of protection bounded by 270000. Here we assume a fixed game with the same parameters as above where $x = 1$, i.e., we have 7000000, 500000, and 300000 vertices of the three types. The number of patrollers required by η differs from the theoretical bound by at most one, while the naive strategies σ need about 125% of this amount on average. In the plot of Fig. 1, the theoretical lower bound on the number of patrollers and the number of patrollers required by η are indistinguishable.

6 Conclusions

The presented method can also be extended to more general models. For example, adapting our algorithm to a generalized model where p is a *function* from V to $(0, 1]$ is easy—the compositional principle stays the same, and in the algorithm, the equations used to compute an optimal assignment for the patrollers are slightly adjusted. Another direction for future research is to consider Attackers with limited capabilities.

The algorithm of Section 5 works well for game structures where the range of d and α is relatively small compared to the number of vertices. For other classes of game structures, one may possibly develop other algorithms, but some decomposition principle similar to ours seems *unavoidable* when the number of vertices reaches a certain threshold. Hence, we believe that the presented approach may trigger the development of the whole spectrum of decomposition techniques applicable to Internet security problems.

References

[Abaffy *et al.*, 2014] M. Abaffy, T. Brázdil, V. Řehák, B. Bošanský, A. Kučera, and J. Krčál. Solving adversarial patrolling games with bounded error. In *Proceedings of AAMAS 2014*, pages 1617–1618, 2014.

[Agmon *et al.*, 2008a] N. Agmon, S. Kraus, and G. Kaminka. Multi-robot perimeter patrol in adversarial settings. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA2008)*, pages 2339–2345. IEEE, 2008.

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

[Almeida *et al.*, 2004] A. Almeida, G. Ramalho, H. Santana, P. Tedesco, T. Menezes, V. Corruble, and Y. Chevaleyre. Recent advances on multi-agent patrolling. In *Advances in Artificial Intelligence – SBIA 2004*, volume 3171 of *LNCS*, pages 474–483. Springer, 2004.

[Basilico *et al.*, 2009a] N. Basilico, N. Gatti, and F. Amigoni. Leader-follower strategies for robotic patrolling in environments with arbitrary topologies. In *AAMAS*, pages 57–64, 2009.

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

[Basilico *et al.*, 2010] N. Basilico, N. Gatti, and F. Villa. Asynchronous Multi-Robot Patrolling against Intrusion in Arbitrary Topologies. In *AAAI*, 2010.

[Basilico *et al.*, 2012] N. Basilico, N. Gatti, and F. Amigoni. Patrolling security games: Definitions and algorithms for solving large instances with single patroller and single intruder. *AI*, 184–185:78–123, 2012.

[Basilico *et al.*, 2016a] N. Basilico, A. Lanzi, and M. Monga. A security game model for remote software protection. In *Proceedings of ARES 2016*, pages 437–443, 2016.

[Basilico *et al.*, 2016b] N. Basilico, G. De Nittis, and N. Gatti. A security game combining patrolling and alarm-triggered responses under spatial and detection uncertainties. In *Proceedings of AAAI 2016*, pages 404–410, 2016.

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

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

[Brázdil *et al.*, 2018] T. Brázdil, A. Kučera, and V. Řehák. Synthesizing Efficient Solutions for Patrolling Problems in the Internet Environment. arXiv:1805.02861, 2018.

[Ceccato and Tonella, 2011] M. Ceccato and P. Tonella. Codebender: Remote software protection using orthogonal replacement. *IEEE Software*, 28(2):28–34, 2011.

[Collberg *et al.*, 2012] C. Collberg, S. Martin, J. Myers, and J. Nagra. Distributed application tamper detection via continuous software updates. In *Proceedings of the 28th Annual Computer Security Applications Conference*, pages 319–328. ACM Press, 2012.

[Cotton, 2015] B. Cotton. Enhancing a city's ability to plan, protect and manage with intelligent video analytics. Technical report, A Frost & Sullivan White Paper, 2015.

[de Cote *et al.*, 2013] E. Munoz de Cote, R. Stranders, N. Basilico, N. Gatti, and N. Jennings. Introducing alarms in adversarial patrolling games: extended abstract. In *AAMAS*, pages 1275–1276, 2013.

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

[Jain *et al.*, 2010] M. Jain, E. Karde, C. Kiekintveld, F. Ordóñez, and M. Tambe. Optimal defender allocation for massive security games: A branch and price approach. In *Workshop on Optimization in Multi-Agent Systems at AAMAS*, 2010.

[Kempe *et al.*, 2018] D. Kempe, L.J. Schulman, and O. Tamuz. Quasi-regular sequences and optimal schedulers for security games. In *Proceedings of SODA 2018*. SIAM, 2018.

[Kiekintveld *et al.*, 2009] C. Kiekintveld, M. Jain, J. Tsai, J. Pita, F. Ordóñez, and M. Tambe. Computing optimal randomized resource allocations for massive security games. In *Proceedings of AAMAS 2009*, pages 689–696, 2009.

[Kučera and Lamser, 2016] A. Kučera and T. Lamser. Regular strategies and strategy improvement: Efficient tools for solving large patrolling problems. In *Proceedings of AAMAS 2016*, pages 1171–1179, 2016.

[Pita *et al.*, 2008] J. Pita, M. Jain, J. Marecki, F. Ordónez, C. Portway, M. Tambe, C. Western, P. Paruchuri, and S. Kraus. Deployed ARMOR protection: The application of a game theoretic model for security at the Los Angeles Int. Airport. In *Proceedings of AAMAS 2008*, pages 125–132, 2008.

[Redmon *et al.*, 2016] J. Redmon, S.K. Divvala, R.B. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016*, pages 779–788. IEEE, 2016.

[Tsai *et al.*, 2009] J. Tsai, S. Rathi, C. Kiekintveld, F. Ordóñez, and M. Tambe. IRIS—a tool for strategic security allocation in transportation networks categories and subject descriptors. In *Proceedings of AAMAS 2009*, pages 37–44, 2009.

[Yin *et al.*, 2010] Z. Yin, D. Korzhyk, C. Kiekintveld, V. Conitzer, and M. Tambe. Stackelberg vs. Nash in security games: Interchangeability, equivalence, and uniqueness. In *AAMAS*, pages 1139–1146, 2010.