Layered Graph Security Games

Jakub Černý[†] , **Chun Kai Ling**[†] , **Christian Kroer** and **Garud Iyengar**
Department of Industrial Engineering and Operations Research, Columbia University
{jakub.cerny, chunkai.ling, christian.kroer}@columbia.edu, garud@ieor.columbia.edu

Abstract

Security games model strategic interactions in adversarial real-world applications. Such applications often involve extremely large but highly structured strategy sets (e.g., selecting a distribution over all patrol routes in a given graph). In this paper, we represent each player's strategy space using a *layered graph* whose paths represent an exponentially large strategy space. Our formulation entails not only classic pursuit-evasion games, but also other security games, such as those modeling anti-terrorism and logistical interdiction. We study two-player zero-sum games under two distinct utility models: linear and binary utilities. We show that under linear utilities, Nash equilibrium can be computed in polynomial time, while binary utilities may lead to situations where even computing a best-response is computationally intractable. To this end, we propose a practical algorithm based on incremental strategy generation and mixed integer linear programs. We show through extensive experiments that our algorithm efficiently computes ϵ-equilibrium for many games of interest. We find that target values and graph structure often have a larger influence on running times as compared to the size of the graph per se.

1 Introduction

Security games model strategic interactions between a defender, typically representing governmental entities, and an attacker engaged in illicit activities. They have served as the foundation for deployed solutions in numerous real-world scenarios, spanning both physical and cyber security domains, such as scheduling air marshals to protect flights [Tsai *et al.*, 2009], or protecting wildlife in natural parks [Fang *et al.*, 2017]. These applications feature complex strategy spaces and reward functions that are application specific. In this paper, we introduce *Layered Graph Security Games* (LGSGs), a class of games where each player selects a path in a layered directed acyclic graph (henceforth *layered graph*) and receive payoffs depending on how "close" these two paths were. LGSGs strike a good balance between model expressiveness and computational complexity. On one hand, many security games and their variants can be easily reframed as LGSGs, despite not being explicitly defined in such terms. These include patrolling games, which have a natural time-component as well as cybersecurity applications, where layered graphs model dependencies in attack chains. Yet, being relatively compact structures, layered graphs retain, and in some cases expose much of the "nice" combinatorial aspects of the underlying security game, resulting in practically efficient game solvers. This stands in contrast to more heavy-handed formulations such as extensive form games.

Our contributions are summarized as follows. (i) We introduce Layered Graph Security Games (LGSG), and show how they lead to compact representations of an otherwise exponentially large space (Section 3). (ii) We demonstrate how many security problems may be reformulated as LGSGs, including various pursuit-evasions games and two novel settings relating to anti-terrorism and logistical interdiction (Section 3.3). (iii) We study the computational complexity of solving LGSGs in two regimes: linear and binary utilities (Section 4), give polynomial-time algorithms for the former, and prove hardness results for the latter. (iv) For binary utilities, we propose a solver based on incremental strategy generation and efficient best-response oracles formulated as mixed integer linear programs. (v) Experiments on a range of applications using both synthetic and real-world maps from various cities and parks (Section 5), show that our strategy generation method scales favorably. We find that equilibria exhibit a tiny support relative to the number of paths, validating our hypothesis that in practical domains, it is structure and not game size that governs computational costs.

2 Related Work

This paper is related to several fields spanning across disciplines. Since these are huge research areas in and of themselves, we focus on those most related to security games.

Pursuit-Evasion games (PEGs) model scenarios when one group (e.g., robots) locates and captures members of another group, often within a specified timeframe. This rich line of work goes by many names (e.g., Cops and Robbers, Differential Games, Games of Pursuit), dealing with a variety of environments, such as those exhibiting perfect or imperfect information, and on discrete and continuous time/space [Isaacs, 1999; Friedman, 2013; Weintraub *et al.*, 2020; Bopardikar *et al.*, 2008; Bonato, 2011; Parsons, 2006]. The mathematics

behind PEGs is deep and attractive. Nonetheless, computational costs become a hindrance in all but the simplest environments. Furthermore, models in PEGs can be fairly rigid; indeed, research in PEGs is often centered around the underlying geometry of the environment or proving theoretical bounds on metrics such as task-completion time.

Extensive-Form games (EFGs) are played on a game tree, with each player choosing actions to take at each of their information sets [Shoham and Leyton-Brown, 2008]. Extremely expressive and successful in practice, EFG solvers are responsible for superhuman poker bots today [Brown and Sandholm, 2018; Brown and Sandholm, 2019]. However, as trees, computing exact equilibria in EFGs incurs computational costs that is exponential in time horizon. Conversely, while the number of possible paths in LGSGs taken is also exponential in horizon, its reward functions are constrained by the layered graph structure, allowing for more efficient computation of best-responses and equilibrium. Recently, there have been significant work scaling up EFG solvers by incorporating machine learning [Lanctot *et al.*, 2017; Perolat *et al.*, 2022; Moravčík *et al.*, 2017; Wang *et al.*, 2019; Xue *et al.*, 2021]. While powerful, such approaches rarely yield theoretical guarantees on quality of the equilibria, making them less suited for high-stakes security applications.

Network Interdiction games (NIGs) study the optimal arcs in a network to remove or interdict in order to prevent an evader from traversing the graph. First studied by Wollmer [1964], NIGs now come in a variety of objectives, such as increasing the evader's shortest path to an exit, minimizing the maximum flow between two vertices, as well as a range of applications, including cybersecurity, cyberphysical security and supply-chain attacks [Washburn and Wood, 1995; Smith and Song, 2020; Smith and Lim, 2008]. Most of the existing literature consider attacker paths, but allow the defender to select arbitrary vertices or edges to interdict. This is unrealistic, particularly in physical patrolling such as anti-poaching, since it amounts to the defender "teleporting" to a location rather moving in to intercept the attacker.

Security games are the namesake of this paper, and have enjoyed much attention owing to a number of successful deployments in the real-world [Jain *et al.*, 2013; Pita *et al.*, 2008; Shieh *et al.*, 2012; An *et al.*, 2017]. In its vanilla form, security games feature a defender choosing a distribution over targets to defend and an attacker choosing one of them to attack. This simple setting enjoys polynomial-time solvers, even in the general-sum case [Kiekintveld *et al.*, 2009; Conitzer and Sandholm, 2006]. Much developments have been made to account for large, but structured strategy spaces such as defender target schedules [Korzhyk *et al.*, 2010] and repeated interactions [Fang *et al.*, 2015]. Unfortunately, just like NIGs, most security games focus on the special case where only the defender possesses structured strategies — the attacker still "teleports" to the target. This assumption reins in computational costs, but at the price of realism.

One of the few works which do handle structured strategies for both players is the Escape Interdiction Game (EIG) studied by Zhang *et al.* [2017] who investigate interdiction specifically in transport networks. LGSGs partially generalizes EIGs by allowing for richer interdiction, reward functions and most importantly goes beyond interdiction to include applications such as anti-terrorism, delayed interdiction and advanced persistent threats (Section 3.3).

3 Layered Graph Security Games

The fundamental structure that we will work with is the *layered directed graph*. Let $\mathcal{V}$ be a finite set of vertices, and $\mathcal{G}_a = (\mathcal{V}, \mathcal{E}_a)$, $\mathcal{G}_d = (\mathcal{V}, \mathcal{E}_d)$ graphs for the attacker and defender, where the sets $\mathcal{E}_a$ and $\mathcal{E}_d$ contain *directed* edges. $\mathcal{V}$ comprises $L > 1$ layers, meaning that $\mathcal{V}$ can be partitioned into non-empty sets $\mathcal{V}_1, \ldots, \mathcal{V}_L$ where all edges $e_a \in \mathcal{E}_a$ lie in $\mathcal{V}^\ell \times \mathcal{V}^{\ell+1}$ for some $\ell \in [1, L-1]$. The same holds for all $e_d \in \mathcal{E}_d$. For an edge $e = (v_l, v_{l+1})$, we denote the vertex v_l by e^- and the vertex v_{l+1} by e^+. For a player $i \in \{a, d\}$, the incoming edges for a vertex v are denoted as $\mathcal{E}_i^+(v)$ and the outgoing edges as $\mathcal{E}_i^-(v)$. Note that $\mathcal{G}_a$ and $\mathcal{G}_d$ share the vertex set $\mathcal{V}$, though we allow $\mathcal{G}_a$ and $\mathcal{G}_d$ to be disconnected.

We assume that the first layer is a singleton, i.e., $|\mathcal{V}^1| = 1$; this restriction does not impose any significant modeling restrictions. However, the last layer may comprise multiple vertices. We call these *terminal states* or *targets* $\mathcal{V}^\odot = \mathcal{V}^L$.

Player strategies. We denote by $\mathcal{P}_a$ and $\mathcal{P}_d$ the set of paths for the attacker and defender. Each path for the attacker is of the form $(v_1, v_2, \ldots v_L) \in \mathcal{V}^1 \times \mathcal{V}^2 \times \ldots, \times \mathcal{V}^L$ where $(v_\ell, v_{\ell+1}) \in \mathcal{E}_a$. Consider some path $p \in \mathcal{P}_a \cup \mathcal{P}_d$. It traverses vertices in increasing order of their layer number and terminates at a target $v \in \mathcal{V}^\odot$. We denote by $p(i)$ the i-th vertex of p and $p[i]$ its i-th edge, i.e., (v_i, v_{i+1}). With a slight abuse of notation we write $v \in p$ and $e \in p$ if vertex v or edge e lies in p. In general, $|\mathcal{P}_a|, |\mathcal{P}_d|$ are exponential in L; dealing with these large strategy spaces is our key contribution.

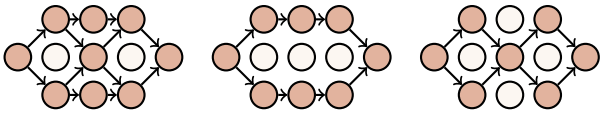
Targets and interdiction function. We assume that each target has an associated value given by $r^\odot : \mathcal{V}^\odot \to \mathbb{R}$. Typically, this value will be nonnegative. For some path $p \in \mathcal{P}_a \cup \mathcal{P}_d$, we write $r^\odot(p)$ as a shorthand for $r^\odot(p(L))$. We also introduce an interdiction function $R : \mathcal{E}_d \times \mathcal{E}_a \to \{0, 1\}$, which is equal to 1 when two edges are in "close proximity" such that the defender is able to interdict the attacker. While R is usually defined in terms of distance metrics (e.g., the range of a sensor and/or physical distance between two edges), we impose no such restriction in our framework.

3.1 Player Utilities

In general, utilities are a function of paths $u_a : \mathcal{P}_d \times \mathcal{P}_a \to \mathbb{R}$. We assume that the game is *zero-sum*, i.e., $u_a(p_d, p_a) = -u_d(p_d, p_a)$. For simplicity, we will write $u = u_a$. Solving a game with arbitrary u is computationally intractable (in terms of the size of the graph $|\mathcal{V}|, |\mathcal{E}_a|, |\mathcal{E}_d|$); indeed, it takes an exponential amount of space to even specify u. Nonetheless, for many security games u takes more compact forms involving edges and possibly $\mathcal{V}^\odot$ and $r^\odot$. The form of u greatly influences equilibrium structure and its computation.

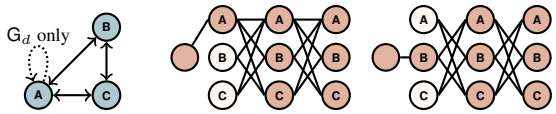
- *Linear utilities* may be additively decomposed in terms of pairs of edges shared between p_d and p_a,

$$u_{\text{LIN}}(p_d, p_a) = \sum_{e_d \in p_d} \sum_{e_a \in p_a} Q(e_d, e_a), \tag{1}$$



(a) $\mathcal{V}$ and $\mathcal{E}_d \cup \mathcal{E}_a$ (b) $\mathcal{G}_d$ (c) $\mathcal{G}_a$

Figure 1: Layered graphs of Example 1. $\mathcal{V}$ has 5 layers with a single source and sink. Disconnected vertices in $\mathcal{G}_a$ and $\mathcal{G}_d$ are in white.



(a) G_d and G_a (b) $\mathcal{G}_d$ (c) $\mathcal{G}_a$

Figure 2: Physical graph and the layered graphs $\mathcal{G}_d, \mathcal{G}_a$ obtained by unrolling over 3 steps. The defender and attacker starts at A and B. Note G_d has an extra loop at A. Unreachable vertices are in white.

where $Q : \mathcal{E}_d \times \mathcal{E}_a \to \mathbb{R}$ maps payoffs between edges between $\mathcal{E}_a$ and $\mathcal{E}_d$. We are particularly interested in the case where $Q = -R(e_d, e_a)$, i.e., the attacker incurs a penalty of 1 each time it is interdicted. Linear utility models allow the attacker to be caught repeatedly. For example, a driver fined for speeding may be fined again in the future, with penalties accumulating additively.

- *Binary utilities* avoid rewarding multiple interdictions,

$$u_{\text{BIN}}(p_d, p_a) = r^{\odot}(p_a) \cdot \mathbb{1}\left[\sum_{e_d \in p_d} \sum_{e_a \in p_a} R(e_d, e_a) = 0\right],$$

where $\mathbb{1}$ is the indicator function, i.e., the attacker receives a reward of $r^{\odot}(p_a)$ if and only if p_a and p_d do not share *any* edge that are "close". Binary utilities are often more suited for security applications. For example, a driver is arrested for drink driving, not released back to the public.

3.2 Nash Equilibrium

Our goal is to find a Nash equilibrium (NE), possibly mixed, over player paths. Denote by Δ_a and Δ_d the probability simplices over $\mathcal{P}_a$ and $\mathcal{P}_d$ respectively. Then, for some distribution over paths $x_i \in \Delta_i$, $x_i(p_i)$ is the probability that p_i is played by player $i \in \{a, d\}$. The NE problem reduces to solving the bilinear saddle point problem

$$\min_{x_d \in \Delta_d} \max_{x_a \in \Delta_a} \mathbb{E}_{p_d \sim x_d, p_a \sim x_a}\left[u(p_d, p_a)\right] \tag{2}$$
$$= \min_{x_d \in \Delta_d} \max_{x_a \in \Delta_a} \sum_{p_d \in \mathcal{P}_d} \sum_{p_a \in \mathcal{P}_a} x_a(p_a) \cdot x_d(p_d) \cdot u(p_d, p_a).$$

Since Δ_d and Δ_a are convex and compact and the objective is convex-concave, the minimax theorem [v. Neumann, 1928] holds. Thus, the game has a unique value.

Example 1. *Consider the game in Figure 1, $r^{\odot} = 1$ and $R(e_d, e_a) = \mathbb{1}[e_d = e_a]$, i.e., interdiction occurs when p_a and p_d share an edge. There are 2 non-trivial "decision points" for $\mathcal{G}_a$, one in layer 1 and another in layer 3. Each has two (independent) decisions, **UP** or **DOWN**. This gives $\mathcal{P}_a = \{UU, UD, DU, DD\}$. Conversely, $\mathcal{G}_d$ has only one nontrivial decision point at the source, and $\mathcal{P}_d = \{U, D\}$.*

We can derive the following: (i) For both u_{LIN} and u_{BIN}, the defender Nash strategy is $x_d^*(U) = x_d^*(D) = 0.5$. (ii) Under u_{LIN}, the attacker Nash strategies are exactly distributions that satisfy $x_a^*(UU) = x_a^*(DD)$. This includes the uniform strategy $x_a^*(UU) = x_a^*(UD) = x_a^*(DU) = x_a^*(DD) = 0.25$. (iii) Under u_{BIN} the unique attacker Nash strategy is $x_a^*(UU) = x_a^*(DD) = 0.5$.

Example 1 illustrates the equilibrium differences between u_{BIN} and u_{LIN}. Under u_{LIN} there exists an equilibrium where x_a and x_d are "Markovian", i.e., the strategy can be expressed as a distribution over actions at each vertex, independently of the path. Such an equilibrium does not exist for u_{BIN}. We discuss Markovianity in more detail in Section 4.

3.3 Applications

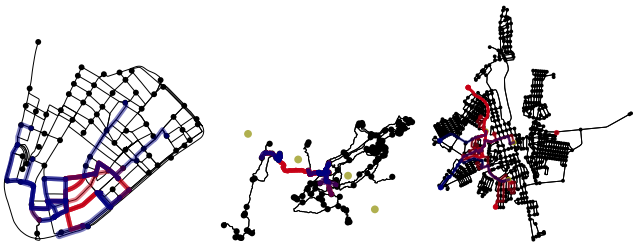
Now we give three examples of layered directed graphs and how the rewards u_{LIN} and u_{BIN} arise. These examples involve *physical graphs* for each player $\mathsf{G}_a = (\mathsf{V}, \mathsf{E}_a)$ and $\mathsf{G}_d = (\mathsf{V}, \mathsf{E}_d)$ (note that we use G for physical graphs and $\mathcal{G}$ for layered graphs). They may be directed, undirected, or include loops. For simplicity, we assume that G_a and G_d share vertices, but not necessarily edges. The vertices in $\mathsf{G}_a, \mathsf{G}_d$ represent locations in the physical world which the attacker and defender traverse over a finite set of timesteps $T \geq 1$. For example, Figure 3 illustrates the road networks used as $\mathsf{G}_a, \mathsf{G}_d$ in the case of Lower Manhattan, Minnewaska State Park, and part of the Ukrainian city of Bakhmut, respectively.

For $i \in \{a, d\}$, the $(T+1)$-layered graph $\mathcal{G}_i$ is obtained from G_i by first fixing a source vertex $\mathsf{v}_{i,\text{source}} \in \mathsf{G}_i$ (or a set of possible sources the player may choose from). We then "unroll" G_i over time. Each layer of $\mathcal{G}_i$ contains vertices which represent where the player is at a given timestep, while edges between layers represent an action taken in the physical world. Figure 2 shows a simple example of how a graph is unrolled over 3 timesteps. Crucially, this unrolling process *does not* result in a tree (whose size is exponential in T), but rather, a compact layered DAG. Note that in general, layers in $\mathcal{G}_i$ need not have the same edges across layers; in fact, vertices in each layer $\mathcal{V}^{\ell}$ need not have a 1-1 mapping with V. Many interesting domains may be described by slightly modifying this unrolling process and/or adding application-specific auxiliary vertices/edges to this DAG. The detailed conversion of G_i to $\mathcal{G}_i$ for each application is deferred to the appendix. [1]

Remark. The graphs $\mathsf{G}_a, \mathsf{G}_d$ are just a convenient way to generate $\mathcal{G}_a$ and $\mathcal{G}_d$. Our framework and algorithms do not exploit properties of $\mathsf{G}_a, \mathsf{G}_d$. In fact, depending on the application, one may choose to sidestep G_a and G_d entirely.

Pursuit-Evasion (PE). The simplest problem described by our formulation is finite-horizon pursuit-evasion games played on graphs [Parsons, 2006]. The pursuer plays the role of the defender, while the evader is the attacker. Players start at some vertex $\mathsf{v}_{a,\text{source}}, \mathsf{v}_{d,\text{source}} \in \mathsf{V}$. At each timestep $t < T$, players select an edge to traverse (loops are allowed in V). We define R such that the attacker is interdicted if both players share the same vertex v at the same time. We reiterate that our framework admits many modifications, e.g., allowing the

[1] Full paper and source code: https://arxiv.org/abs/2405.03070.



(a) L. Manhattan (b) Minnewaska SP (c) City of Bakhmut

Figure 3: Real-world physical graphs used to generate LGSGs for our application domains, together with examples of defender's (red) and attacker's (blue) equilibrial paths in (a) PE, (b) AT, and (c) LI.

attacker to be interdicted when traversing the same *edge* as the defender, or if both players are physically "close" to each other. Note that the game is one-shot: no information is revealed to either player after each step. Nonetheless, this captures a wide variety of pursuit-evasion games. For example, when the goal of the attacker is simply to evade capture, we set $r^{\odot}(p_a) = 1$ and use binary utilities u_{BIN} (we adjust $r^{\odot}$ appropriately if final attacker locations matter).

Anti-Terrorism (AT). An extension to PE games has the attacker playing the role of a terrorist, which seeks to plant an explosive at some target node $\mathsf{v} \in \mathsf{V}$ while evading capture. The explosive device requires $T_{\text{setup}} \geq 0$ time to setup, during which the attacker must remain at that same vertex. Once the setup is complete, the explosive detonates and the attacker receives utility equal to $r(\mathsf{v}) \geq 0$. The attacker gets 0 utility if the explosive was not planted by the time limit, or the attacker was interdicted while moving around or planting the explosive. As with PE games, the interdiction function R may be defined in various meaningful ways. This game can be formulated using the same layered graph formulation by introducing additional "waiting" vertices for each target which represent how long an attacker has been setting up up the explosive at each vertex. This results in a layered graph of $L = T + 1$ layers, each roughly of size $\mathcal{O}(|\mathsf{V}| \cdot T)$. Unlike PE games, the representation of the interdiction function in terms of $\mathcal{G}_a, \mathcal{G}_d$ is slightly more complicated.

Logistical Interdiction (LI) and Persistent Threats (PT). PE games may be modified to contain select exit vertices which end the game when the attacker reaches them. We introduce a *delay factor* $\gamma \geq 0$. If the attacker reaches one of these exit vertices at time t_{exit} it obtains a payoff of $\gamma^{t_{\text{exit}}}$. If it does not, or gets captured, it gets a payoff of 0. When $\gamma = 1$, this reduces to PE games with the introduction of a special 'exit' vertex with a self-loop only reachable by the attacker. When $\gamma < 1$, the delay factor encourages the attacker to exit as soon as possible. We call these logistical interdiction (LI) games, which model supply lines in warfare where delays in shipments result in casualties. When $\gamma > 1$, the attacker delays exiting as long as possible. We call these persistent threat (PT) games. A raiding party seeks to cause damage for as long as possible (without being captured). Similarly, Advanced Persistent Threats (APTs) in cybersecurity procure classified information for as long as possible (without being evicted) before leaving [Rass *et al.*, 2017].

4 Computing Equilibrium in LGSGs

This expressiveness of LGSGs comes at a cost: finding a NE is intractable. This is unsurprising since layered graph games generalize the EIGs of Zhang *et al.* [2017].

Proposition 1. *It is NP-hard to find a NE for a layered graph security game with general utilities given in Equation 2.*

The proof is essentially identical to the reduction from 3-SAT in Zhang *et al.* [2017]. Their reduction uses multiple defenders, but may be adapted to LGSGs with $u = u_{\text{BIN}}$, $r^{\odot} = 1$ and suitably designed R (see appendix for details). We now delve deeper and discuss subclasses of layered games and their respective computational complexities.

4.1 LGSGs with Linear Utility Models

In the case of linear utilities u_{LIN}, equilibrium computation is greatly simplified. This arises from the use of network *flows* as a polynomial-size representation of strategies which is payoff equivalent to Δ_d, Δ_a. Given a layered graph (or any single-source DAG), the unit-flow polytopes Γ_d, Γ_a are given by flow conservation constraints,

$$\Gamma_i = \left\{ f_i \geq 0 \;\middle|\; \begin{array}{ll} \sum_{e \in \mathcal{E}_i^-(v)} f_i(e) = 1 & v \in \mathcal{V}_i^1 \\ \sum_{e \in \mathcal{E}_i^-(v)} f_i(e) = \sum_{e \in \mathcal{E}_i^+(v)} f_i(e) & v \in \mathcal{V}_i \backslash \mathcal{V}_i^1 \end{array} \right\}$$

Flows describe for each player $i \in \{a, d\}$ the marginal probability that a particular edge is traversed. Crucially, every flow is the convex combination of *some* set of paths. In particular, flows embody *Markovian strategies*. For internal vertices $v \in \mathcal{V}_i \backslash \mathcal{V}_i^1$, the conditional probability of taking edge $e = (v, v')$ is $f_i(e|v) = f_i(e) / \sum_{e' \in \mathcal{E}_i^+(v)} f_i(e')$, where by convention we set $0/0 = 0$, while $f_i(e|v) = f_i(e)$ for $v \in \mathcal{V}_i^1$. The probability $x_i(p)$ of choosing a path $p \in \mathcal{P}_i$ is $\prod_{e:(v,v') \in p} f_i(e, v)$. Conversely, any distribution over paths $x_i \in \Delta_i$ (not necessarily Markovian), maps to a flow $f_i : \mathcal{E}_i \mapsto [0, 1]$ where $f_i(e_i) = \sum_{p_i \in P_i(e_i)} x_i(p_i)$. In fact, the vertices of Γ_i correspond precisely to paths, simplifying linear utility models.

Proposition 2 (Kuhn's theorem for u_{LIN})**.** *Suppose $x_d \in \Delta_d, x_a \in \Delta_a$ in a layered graph security game. Then u_{LIN} is bilinear in their flows $f_d(x_d)$ and $f_a(x_a)$. Specifically,*

$$u_{\text{LIN}}(x_d, x_a) = \sum_{e_d \in \mathcal{E}_d} \sum_{e_a \in \mathcal{E}_a} Q(e_d, e_a) f_d(e_d) f_a(e_a).$$

Proposition 2 implies that rather than optimizing over Δ_i, we can optimize over Γ_i instead. Computing a Nash equilibrium (2) may be expressed as a *bilinear saddle point problem*

$$\min_{f_d \in \Gamma_d} \max_{f_a \in \Gamma_a} \sum_{e_d \in \mathcal{E}_d} \sum_{e_a \in \mathcal{E}_a} Q(e_d, e_a) f_d(e_d) f_a(e_a). \quad (3)$$

We remark that, since Markovian strategies do not capture every distribution over paths (i.e., there may be multiple $x_i \in \Delta_i$ mapping onto the same flow), the solutions to the above optimization will not necessarily capture *all* equilibria.

Nonetheless, it will provide *some* Markovian equilibrium. In fact, we argue that this is desirable since Markovian strategies are compactly represented. The optimization problem in (3) lends itself well to computation. Since Γ_d and Γ_a are compact and convex sets (being polytopes), the minimax theorem [v. Neumann, 1928] holds. In particular, (3) resembles the classic min-max formulation for solving zero-sum normal-form games, except that we optimize over Γ_i instead of over the probability simplex. Taking the dual of the inner maximization problem yields the following linear program.

$$\min_{f_d \in \Gamma_d, g \in \mathbb{R}^{|\mathcal{V}|}} g(v_{\text{source}})$$
$$g(e_a^+) - g(e_a^-) \geq \sum_{e_d \in \mathcal{E}_d} f_d(e_d) \cdot Q(e_d, e_a) \quad \forall e_a \in \mathcal{E}_a$$

Here, the g variables are dual variables corresponding to values of *vertices*. We remark that other methods such as those involving regret minimization over Γ_i [Takimoto and Warmuth, 2003; Farina *et al.*, 2019; Farina *et al.*, 2022] may be more efficient in practice than this LP. Regardless, since the number of variables and constraints is linear in the sizes of $\mathcal{G}_i$ and LPs are solvable in polynomial time, we have:

Proposition 3. *A NE for a LGSG with linear utilities as in Equation* (1) *may be found in polynomial time.*

Since LGSGs with linear utilities are less suited for security applications and also computationally uninteresting, we focus on binary utility models for the rest of the paper.

4.2 The Role of Flows in Binary Utility Models

In games with linear utilities, restricting the space of strategies to flows resulted in polynomial-time algorithms. Such optimism is perhaps unwarranted in binary utilities given the intractability result of Proposition 1. It turns out that, not only are polynomial-time algorithms unlikely, even the restriction of strategies from $\mathcal{P}_i$ to Γ_i itself may not contain any NE.

Proposition 4. *There exist LGSG with* u_{BIN}, $r^{\odot} = 1$, $R(e_d, e_a) = \mathbb{1}[e_d = e_a]$ *where no NE is Markovian.*

Proposition 4 follows directly from Example 1, whose unique attacker Nash strategy $x_a^*(UU) = x_a^*(DD)$ is clearly not Markovian since the decision at the second vertex depends on the previous action taken. This dependency is intuitive: imagine the attacker has already taken the top path and is deciding where to go in the second decision vertex. The fact that it was not interdicted implies that the defender did not play U, implying that it should continue to play UU.

Proposition 4 seems trivial in hindsight, but has important ramifications. For example, one may try running deep reinforcement learning methods with self-play to arrive at an equilibrium [Wang *et al.*, 2019]. Proposition 4 implies that the state features fed into such a policy or Q-function *must* describe the player's history and not just features of that vertex. This stands in contrast to Markov games, and is analogous to the importance of perfect recall in EFGs.

There exists similar examples to Example 1 where $\mathcal{E}_d = \mathcal{E}_a$ but with $r^{\odot}(v)$ varying over targets. These negative examples lead us to believe that searching purely in the space of Γ_i is unlikely to yield fruitful results. Nevertheless, our experimentation seem to substantiate the following special case.

Conjecture 1. *LGSGs with* $u = u_{\text{BIN}}$, $r^{\odot} = 1$, $R(e_d, e_a) = \mathbb{1}[e_d = e_a]$ *and* $\mathcal{E}_d = \mathcal{E}_a$ *have a Markovian NE.*

4.3 Best-Responses in Binary Utility Models

Recall that defender's and attacker's (pure) best-responses to fixed strategies x_a and x_d are defined as $p_d^{\text{BR}} = \arg\min_{p_d \in \mathcal{P}_d} u(p_d, x_a)$ and $p_a^{\text{BR}} = \arg\max_{p_a \in \mathcal{P}_a} u(x_d, p_a)$, where $u(x_d, p_a) = \mathbb{E}_{p_a \sim x_a} u(p_d, p_a)$ and $u(p_d, x_a) = \mathbb{E}_{p_d \sim x_d} u(p_d, p_a)$. One may hope that with u_{BIN}, computing best-responses is tractable even if equilibrium solving is not. Unfortunately, it turns out that this too is intractable.

Proposition 5. *Let* $\widetilde{\mathcal{P}}_i \subseteq \mathcal{P}_i$ *be of size* k *(possibly much smaller than* $|\mathcal{P}_i|$*) and* $\widetilde{x}_i$ *be a distribution with support* $\widetilde{\mathcal{P}}_i$*. Finding a best response of player* $-i$ *against* $\widetilde{x}_i$ *in a LGSG with* $u = u_{\text{BIN}}$ *is NP-hard in terms of* $|\mathcal{G}_a|, |\mathcal{G}_d|$ *and* k*.*

We stress that while best-responses are closely related to NE computation, these are generally distinct problems: it is possible to devise classes of games where best-responses are difficult to compute but finding NE is easy and vice versa [Xu, 2016]. Proposition 5 suggests that even *verifying* that a given pair (x_d^*, x_a^*) is a NE may be difficult. Thankfully, it turns out that in many games of interest a best-response can be formulated as a mixed-integer linear program (MILP) which can be practically tackled by commercial solvers, at least when the opponent distribution $\widetilde{x}_i$ has small support.

Let $\widetilde{\mathcal{P}}_a \subseteq \mathcal{P}_a$ be the support of an attacker strategy $\widetilde{x}_a(p_a)$. The following Mixed-integer linear program (MILP) solves for the best defender response in the form of a path $p_d \in \mathcal{P}_d$.

$$\max_{f_d \in \Gamma_d} \sum_{p_a \in \widetilde{\mathcal{P}}_a} -r^{\odot}(p_a) \cdot (1 - y(p_a)) \cdot \widetilde{x}_a(p_a)$$
$$y(p_a) \leq \sum_{e_d \in \{e_d : \exists e_a \in p_a \, R(e_d, e_a) = 1\}} f_d(e_d) \qquad \forall p_a \in \widetilde{\mathcal{P}}_a$$
$$f_d(e_d) \in \{0, 1\} \qquad \forall e_d \in \mathcal{E}_d$$
$$0 \leq y(p_a) \leq 1 \qquad \forall p_a \in \widetilde{\mathcal{P}}_a.$$

In the above, $y(p_a)$ is the number of times the attacker is interdicted when playing $p_a \in \widetilde{\mathcal{P}}_a$; note that this will be integeral as elements of f_d are binary. Observe that f_d lies in the intersection of the unit-flow polytope Γ_d and the $\{0,1\}^{|\mathcal{E}_d|}$ integer lattice, which is precisely the set of all paths in $\mathcal{G}_d$. The objective minimizes the attacker utility by trying to achieve large $y(p_a)$. The first constraint ensures that $y(p_a)$ can only be set to 1 if at least one edge is shared between p_a and the path given by f_d, while the last constraint limits the defender to a maximum of one interdiction. This MILP grows in size with $|\widetilde{\mathcal{P}}_a|$, and is likely to be hard when $|\widetilde{\mathcal{P}}_a|$ is large.

This MILP serves as the defender's *best-response oracle* to a given distribution of attacker paths $\widetilde{x}_a$. We denote this by $\textsc{DefenderBR}(\widetilde{x}_a)$. Similarly, the attacker's best response to a distribution of defender paths $\widetilde{x}_d$ with support in $\widetilde{\mathcal{P}}_d \subseteq \mathcal{P}_d$ is given by $\textsc{AttackerBR}(\widetilde{x}_d)$. $\textsc{AttackerBR}$ can also be written as a MILP, the details of which are deferred to the appendix. These best-responses are similar in spirit to those in Zhang *et al.* [2017], except that we account for a wider class of interdiction functions R and target values $r^{\odot}$.

Algorithm 1 Double Oracle for LGSGs

Require: $\mathcal{G}_d, \mathcal{G}_a, r^{\odot}, R, u, \epsilon > 0$
1: $\widetilde{\mathcal{P}}_d, \widetilde{\mathcal{P}}_a \leftarrow$ INITIALSUBGAME$(\mathcal{G}_d, \mathcal{G}_a)$
2: **repeat**
3: $\quad \widetilde{x}^*_d, \widetilde{x}^*_a \leftarrow$ NASHEQUILIBRIUM$(\widetilde{\mathcal{P}}_d, \widetilde{\mathcal{P}}_a)$
4: $\quad p^{\text{BR}}_d, p^{\text{BR}}_a \leftarrow$ DEFENDERBR$(\widetilde{x}^*_a)$, ATTACKERBR$(\widetilde{x}^*_d)$
5: $\quad \widetilde{\mathcal{P}}_d, \widetilde{\mathcal{P}}_a \leftarrow \widetilde{\mathcal{P}}_d \cup \{p^{\text{BR}}_d\}, \widetilde{\mathcal{P}}_a \cup \{p^{\text{BR}}_a\}$
6: **until** EQUILIBRIUMGAP$(\widetilde{x}^*_d, \widetilde{x}^*_a, p^{\text{BR}}_d, p^{\text{BR}}_a) < \epsilon$

4.4 Approximating NE Using Strategy Generation in LGSGs with Binary Utility Models

The negative results of Proposition 1 compels us to work directly in the space of path distributions instead. Fortunately, despite the number of paths being exponential, we find that in practical applications such as those in Section 3.3, equilibria exhibit relatively small supports (in path space). This motivates our adoption of the *double oracle* framework.

The double oracle (DO) algorithm is a variant of concurrent column and row generation. It is commonly used to solve large saddle-point problems which admit efficient (in a practical sense) best-response oracles. The DO algorithm is an iterative algorithm that incrementally builds a subgame — a subset of pure strategies for each player — with the hope that "weak" strategies outside the equilibrium's support are never included in the subgame. DO guarantees that at termination, the subgame (ideally a small fraction of the entire game) has a NE which mirrors the NE of the original game. In our setting, pure strategies are paths $p_i \in \mathcal{P}_i$ and subgames are specified via subsets $\widetilde{\mathcal{P}}_i \subseteq \mathcal{P}_i$ for each i. An outline of our proposed DO implementation is shown in Algorithm 1.

The algorithm starts from a small subgame for each player $\widetilde{\mathcal{P}}_d, \widetilde{\mathcal{P}}_a$. At each iteration, it computes the equilibrium $(\widetilde{x}^*_d, \widetilde{x}^*_a)$ within the current subgame (i.e., player i may only choose distributions of paths in $\widetilde{\mathcal{P}}_i$) as a normal-form game. For each player $i \in \{a, d\}$ we compute best-responses p^{BR}_i (of the full game) against their opponent's subgame equilibrium strategy $\widetilde{x}^*_{-i}$. This is done using the best-response oracles. These best-responses give paths which are added into the subgame, and the procedure repeats.

The DO algorithm terminates when the best-response oracle for *both* players returns responses that do not improve the player's return over the subgame value. This implies that the current subgame equilibrium constitutes an equilibrium in the full game, and further addition of strategies will not result in less exploitable strategies for either player. In practice, rather than converging to an exact equilibrium, we compute the *equilibrium gap* $\nabla = u(\widetilde{x}^*_d, p^{\text{BR}}_a) - u(p^{\text{BR}}_d, \widetilde{x}^*_a)$ and terminate when $\nabla \leq \epsilon$ for some pre-specified threshold $\epsilon > 0$.

Speeding up DO. The efficiency of DO hinges on three factors, (i) the time needed to compute a NE of the subgame, (ii) the number of outer iterations of DO, i.e., the number of strategies added to the subgame and (iii) the best response oracles for each player. The first factor is typically negligible, since computing the NE of a zero-sum matrix game may be done in polynomial time. The second factor depends on the underlying game (e.g., does it have a sparse equilibrium). The third factor depends on how hard the MILP is.

The component most within our control is (iii), since MILP solvers can be tuned via parameters and reformulation. We consider the following speedups, (a) admitting approximate best-responses (or better responses) rather than solving MILPs to completion, (b) strategy management by periodically removing "weak" strategies in $\widetilde{\mathcal{P}}_i$ that absent in $\widetilde{x}^*_i$, (c) tightening of MILPs by adding cuts/implied constraints or lifting, and (d) tuning the MILP solver, using warm-starts, or heuristics. Unfortunately, we find that only (a) yielded consistently better results. We discuss implementation details and other speedups in the appendix.

5 Empirical Evaluation

We seek to answer the following questions. (i) Performance: how does DO compare with the full LP solver? (ii) Sparsity: how do sizes of the (approximate) NE's support and DO subgames compare to the game size itself? (iii) Factors: how does performance scale with other game parameters?

The experiments were conducted on an Intel Xeon Gold 6226, 2.9Ghz on a Linux 64-bit platform. We used Gurobi 10.0.3 [Gurobi Optimization, LLC, 2023] for MILPs and LPs. Each run was restricted to 8 threads and 32GB of RAM. Our DO algorithm was implemented in Python 3.7.9 and configured with a tolerance of $\epsilon = 10^{-3}$. The real-world physical graphs were obtained using OSMnx [Boeing, 2017]. For experiments with randomness, 20 instances were generated and solved. We report their standard errors in plots, *noting that in almost all cases these are negligible*. We limit solvers to 6 hours for each instance . Game sizes are defined as $|\mathcal{P}_d|+|\mathcal{P}_a|$ when reporting runtimes. We allow loops in G_i and set R such that the attacker is interdicted when players share a vertex. Application specific instantiations of R and $r^{\odot}$, as well as additional setup and results are deferred to the appendix. A link to the source code is included in the full paper.

Performance and sparsity. We compare linear programming for the full game (denoted LP, in the figures) against DO with (i) exact (denoted DO) and (ii) approximate (denoted DO+l) best-responses achieved by halting best-response computations after 1s. For sparsity, we report subgame and support sizes (denoted SG and SP) summed over players and as a fraction of the total number of paths $|\mathcal{P}_d| + |\mathcal{P}_a|$.

We experiment on (a) a synthetic 4-connected 5×5 *grid world*, with a few edges randomly removed per player (making the grids unique), and (b) the map of *Lower Manhattan* in New York City, USA, with 87 nodes and 191 edges (depicted in Figure 3). In each domain, we ran pursuit-evasion (PE) and anti-terrorism (AT) scenarios for varying horizons (i.e., different game sizes). For each of the 4 settings, we report in Figure 4 running times (over 3 algorithms) and the final DO subgame (SG) and support (SP) sizes. Note that for purposes of comparison, target values are identical over PE and AT.

As expected, both variants of DO outperform linear programming, which is unable to solve modestly-sized games within the time limit of 6 hours. On Lower Manhattan, the use of approximate best-responses yields up to half an order-of-magnitude speedup over exact best-responses, despite final subgame sizes being virtually identical. We report in the

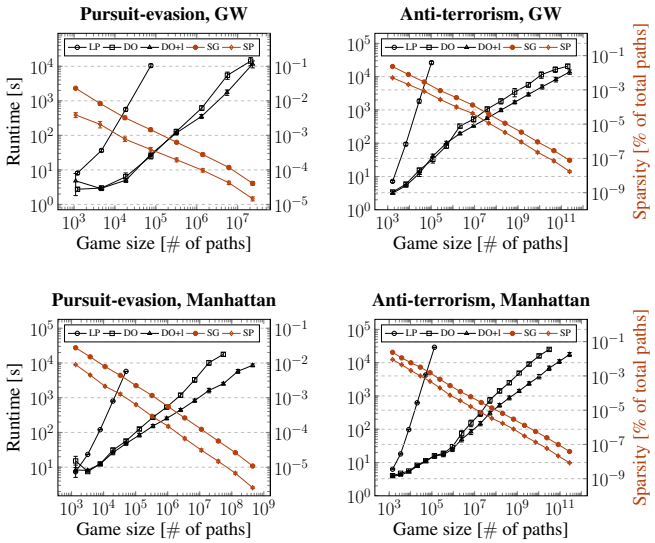


Figure 4: Computation times (black lines) and sparsity metrics (orange lines, for vanilla DO with exact best-responses) for PE and AT domains on grid world and Lower Manhattan.

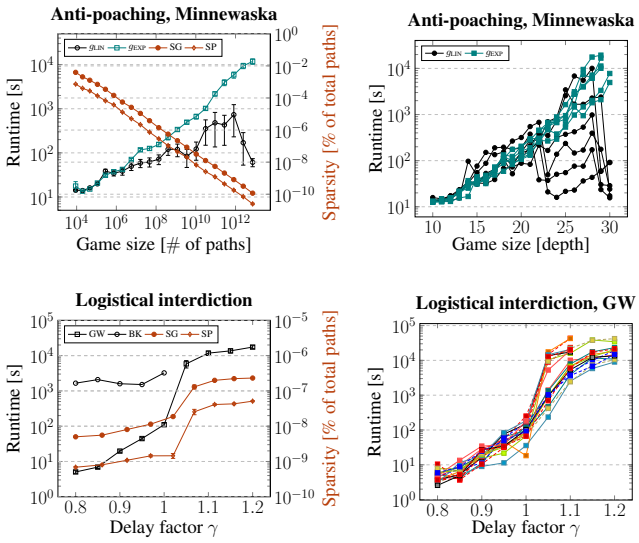


Figure 5: Computation times and sparsity metrics for AP and LI domains demonstrating the effects of additional factors on the performance of our DO algorithm with approximate best-responses.

appendix how equilibrium gaps evolve with running time.

Even though PE and AT utilize identical physical graphs and target values, solving PE games appears harder than AT games. This stems from more involved best-response computations. Furthermore, the size of the NE support remains no more than 2-5 times less than the subgame sizes. This means DO avoids adding too many unnecessary strategies. We discuss the qualitative behavior of equilibrium in the appendix.

Effect of diameter and horizon. Consider *Minnewaska State Park* in NY, USA (138 nodes, 201 edges, Figure 3). Here, we apply AT for anti-poaching, i.e., planting explosives $\approx$ poaching. To model spatial correlations in animal density, we randomly assign 4 "animal habitats" (yellow dots in figure), each associated with a positive animal score. Each node's value is the sum of animal scores, attenuated by $g_{\text{LIN}}(z) = 1/z$, where z is the node's euclidean distance to the habitat.We report results in the top row of Figure 5.

The results on DO+l in Figure 5 (black line, top left) exhibit an anomaly where increasing horizon (i.e., larger games) leads to a *decrease* in running time around the 10^{10} mark. Furthermore, the standard deviations become extremely high (see Figure 5, top right for individual runs). It turns out that at lower horizons, parts of the G_i were not accessible and are only "unlocked" at a deeper horizons. Sometimes, these locations lie close to a habitat (hence g_{LIN} is high), unlocking them causes the attacker's strategy to almost always move to these rich locations. This "phase transition" is accentuated as g_{LIN} has a singularity at $z = 0$. Hence, even though the game is *larger*, the NE can be *simpler*. To validate our hypothesis, we ran the same experiment (same habitats) using $g_{\text{EXP}}(z) = \exp(-z)$, avoiding the singularities in g_{LIN}. This anomaly and large standard deviations then vanish.

Effect of delay factor in LI and PT. Again, we consider (i) the 5×5 grid world with 4-connectivity (denoted GW), and (ii) the city of *Bakhmut*, Ukraine (denoted BK, 721 nodes, 1229 edges, Figure 3), a frontline in the Russo-Ukrainian 2022 war. The latter simulates logistical interdiction (LI) involving the delivery of Ukrainian supplies to three frontline locations (yellow dots in figure, playing the role of exits) from two entry points (west-most blue dots) while being susceptible to Russian attacks in the contested territory. For each domain we study DO+l runtimes on the same physical graph G_i, varying only the delay factor γ. In GW we consider $\gamma \in [0.8, 1.2]$, i.e., both persistent threats and interdiction, while in Bakhmut we only consider $\gamma \leq 1$, i.e., interdiction.

We report results in the bottom row of Figure 5. For GW, games with high γ are more difficult to solve. When γ is small, the attacker is incentivized to rush to an exit with less consideration for being caught; when γ is closer to 1, more unpredictable NE (hence larger support) are employed. This occurs to a lesser degree in Bakhmut. Because players start far apart, earlier actions are often inconsequential: players simply move closer to the frontline where meaningful decisions are actually made (illustrations in appendix). For $\gamma > 1$ in grid worlds (PT), running times skyrocket alongside the subgame and support sizes. This is expected as being very unpredictable is necessary to delay departure without being interdicted (see bottom right of Figure 5 for individual runs).

6 Conclusion

This paper introduced Layered Graph Security Games (LGSGs). LGSGs offer a balance between model expressiveness and computational complexity. We study the complexity of solving LGSGs and propose a solver for the challenging case of binary utilities. Our experiments demonstrate scalability of our method and highlights the importance of structure over game size when estimating computational costs.

Acknowledgments

This research was supported by the Office of Naval Research award N00014-23-1-2374. Christian Kroer was additionally supported by the Office of Naval Research award N00014-22-1-2530, and the National Science Foundation awards IIS-2147361 and IIS-2238960. We would like to thank Daniel Bienstock for the useful discussions on mixed-integer programming, as well as Sven Bednář for his assistance with formalizing the Bakhmut scenario.

Contribution Statement

Equal contribution between authors Jakub Černý[†] and Chun Kai Ling[†].

References

[An *et al.*, 2017] Bo An, Milind Tambe, and Arunesh Sinha. Stackelberg security games (ssg) basics and application overview. *Improving Homeland Security Decisions*, page 485, 2017.

[Boeing, 2017] Geoff Boeing. Osmnx: New methods for acquiring, constructing, analyzing, and visualizing complex street networks. *Computers, Environment and Urban Systems*, 65:126–139, 2017.

[Bonato, 2011] Anthony Bonato. *The game of cops and robbers on graphs*. American Mathematical Soc., 2011.

[Bopardikar *et al.*, 2008] Shaunak D Bopardikar, Francesco Bullo, and Joao P Hespanha. On discrete-time pursuit-evasion games with sensing limitations. *IEEE Transactions on Robotics*, 24(6):1429–1439, 2008.

[Brown and Sandholm, 2018] Noam Brown and Tuomas Sandholm. Superhuman ai for heads-up no-limit poker: Libratus beats top professionals. *Science*, 359(6374):418–424, 2018.

[Brown and Sandholm, 2019] Noam Brown and Tuomas Sandholm. Superhuman ai for multiplayer poker. *Science*, 365(6456):885–890, 2019.

[Conitzer and Sandholm, 2006] Vincent Conitzer and Tuomas Sandholm. Computing the optimal strategy to commit to. In *Proceedings of the 7th ACM conference on Electronic commerce*, pages 82–90, 2006.

[Fang *et al.*, 2015] Fei Fang, Peter Stone, and Milind Tambe. When security games go green: Designing defender strategies to prevent poaching and illegal fishing. In *IJCAI*, pages 2589–2595, 2015.

[Fang *et al.*, 2017] Fei Fang, Thanh H Nguyen, Rob Pickles, Wai Y Lam, Gopalasamy R Clements, Bo An, Amandeep Singh, Brian C Schwedock, Milind Tambe, and Andrew Lemieux. Paws—a deployed game-theoretic application to combat poaching. *AI Magazine*, 2017.

[Farina *et al.*, 2019] Gabriele Farina, Chun Kai Ling, Fei Fang, and Tuomas Sandholm. Efficient regret minimization algorithm for extensive-form correlated equilibrium. *Advances in Neural Information Processing Systems*, 32, 2019.

[Farina *et al.*, 2022] Gabriele Farina, Chung-Wei Lee, Haipeng Luo, and Christian Kroer. Kernelized multiplicative weights for 0/1-polyhedral games: Bridging the gap between learning in extensive-form and normal-form games. In *International Conference on Machine Learning*, pages 6337–6357. PMLR, 2022.

[Friedman, 2013] Avner Friedman. *Differential games*. Courier Corporation, 2013.

[Gurobi Optimization, LLC, 2023] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2023.

[Isaacs, 1999] Rufus Isaacs. *Differential games: a mathematical theory with applications to warfare and pursuit, control and optimization*. Courier Corporation, 1999.

[Jain *et al.*, 2013] Manish Jain, Bo An, and Milind Tambe. Security games applied to real-world: Research contributions and challenges. In *Moving Target Defense II: Application of Game Theory and Adversarial Modeling*, pages 15–39. Springer, 2013.

[Kiekintveld *et al.*, 2009] Christopher Kiekintveld, Manish Jain, Jason Tsai, James Pita, Fernando Ordónez, and Milind Tambe. Computing optimal randomized resource allocations for massive security games. 2009.

[Korzhyk *et al.*, 2010] Dmytro Korzhyk, Vincent Conitzer, and Ronald Parr. Complexity of computing optimal stackelberg strategies in security resource allocation games. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 24, pages 805–810, 2010.

[Lanctot *et al.*, 2017] Marc Lanctot, Vinicius Zambaldi, Audrunas Gruslys, Angeliki Lazaridou, Karl Tuyls, Julien Pérolat, David Silver, and Thore Graepel. A unified game-theoretic approach to multiagent reinforcement learning. *Advances in neural information processing systems*, 30, 2017.

[Moravčík *et al.*, 2017] Matej Moravčík, Martin Schmid, Neil Burch, Viliam Lisỳ, Dustin Morrill, Nolan Bard, Trevor Davis, Kevin Waugh, Michael Johanson, and Michael Bowling. Deepstack: Expert-level artificial intelligence in heads-up no-limit poker. *Science*, 356(6337):508–513, 2017.

[Parsons, 2006] Torrence D Parsons. Pursuit-evasion in a graph. In *Theory and Applications of Graphs: Proceedings, Michigan May 11–15, 1976*, pages 426–441. Springer, 2006.

[Perolat *et al.*, 2022] Julien Perolat, Bart De Vylder, Daniel Hennes, Eugene Tarassov, Florian Strub, Vincent de Boer, Paul Muller, Jerome T Connor, Neil Burch, Thomas Anthony, et al. Mastering the game of stratego with model-free multiagent reinforcement learning. *Science*, 378(6623):990–996, 2022.

[Pita *et al.*, 2008] James Pita, Manish Jain, Fernando Ordónez, Christopher Portway, Milind Tambe, Craig Western, Praveen Paruchuri, and Sarit Kraus. Armor security for los angeles international airport. In *AAAI*, pages 1884–1885, 2008.

[Rass *et al.*, 2017] Stefan Rass, Sandra König, and Stefan Schauer. Defending against advanced persistent threats using game-theory. *PloS one*, 12(1):e0168675, 2017.

[Shieh *et al.*, 2012] Eric Shieh, Bo An, Rong Yang, Milind Tambe, Craig Baldwin, Joseph DiRenzo, Ben Maule, and Garrett Meyer. Protect: A deployed game theoretic system to protect the ports of the united states. In *Proceedings of the 11th international conference on autonomous agents and multiagent systems-volume 1*, pages 13–20, 2012.

[Shoham and Leyton-Brown, 2008] Yoav Shoham and Kevin Leyton-Brown. *Multiagent systems: Algorithmic, game-theoretic, and logical foundations*. Cambridge University Press, 2008.

[Smith and Lim, 2008] J Cole Smith and Churlzu Lim. Algorithms for network interdiction and fortification games. *Pareto optimality, game theory and equilibria*, pages 609–644, 2008.

[Smith and Song, 2020] J Cole Smith and Yongjia Song. A survey of network interdiction models and algorithms. *European Journal of Operational Research*, 283(3):797–811, 2020.

[Takimoto and Warmuth, 2003] Eiji Takimoto and Manfred K Warmuth. Path kernels and multiplicative updates. *The Journal of Machine Learning Research*, 4:773–818, 2003.

[Tsai *et al.*, 2009] Jason Tsai, Christopher Kiekintveld, Fernando Ordóñez, Milind Tamble, and Shyamsunder Rathi. Iris - a tool for strategic security allocation in transportation networks categories and subject descriptors. In *Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems*, 2009.

[v. Neumann, 1928] J v. Neumann. Zur theorie der gesellschaftsspiele. *Mathematische annalen*, 100(1):295–320, 1928.

[Wang *et al.*, 2019] Yufei Wang, Zheyuan Ryan Shi, Lantao Yu, Yi Wu, Rohit Singh, Lucas Joppa, and Fei Fang. Deep reinforcement learning for green security games with real-time information. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 1401–1408, 2019.

[Washburn and Wood, 1995] Alan Washburn and Kevin Wood. Two-person zero-sum games for network interdiction. *Operations research*, 43(2):243–251, 1995.

[Weintraub *et al.*, 2020] Isaac E Weintraub, Meir Pachter, and Eloy Garcia. An introduction to pursuit-evasion differential games. In *2020 American Control Conference (ACC)*, pages 1049–1066. IEEE, 2020.

[Wollmer, 1964] Richard Wollmer. Removing arcs from a network. *Operations Research*, 12(6):934–940, 1964.

[Xu, 2016] Haifeng Xu. The mysteries of security games: Equilibrium computation becomes combinatorial algorithm design. In *Proceedings of the 2016 ACM Conference on Economics and Computation*, pages 497–514, 2016.

[Xue *et al.*, 2021] Wanqi Xue, Youzhi Zhang, Shuxin Li, Xinrun Wang, Bo An, and Chai Kiat Yeo. Solving large-scale extensive-form network security games via neural fictitious self-play. *arXiv preprint arXiv:2106.00897*, 2021.

[Zhang *et al.*, 2017] Youzhi Zhang, Bo An, Long Tran-Thanh, Zhen Wang, Jiarui Gan, and Nicholas R Jennings. Optimal escape interdiction on transportation networks. 2017.