Layered Graph Security Games

Jakub Černý , **Chun Kai Ling** , **Christian Kroer** and **Garud Iyengar**
Department of Industrial Engineering and Operations Research, Columbia University
{jc6224, cl4488, ck2945}@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, even though in reality one would expect the defender 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 be working with is the *layered directed graph*. Let $\mathcal{V}$ be a finite set of vertices, and $\mathcal{G}_a = (\mathcal{V}, \mathcal{E}_a)$, and $\mathcal{G}_d = (\mathcal{V}, \mathcal{E}_d)$ graphs for the attacker and defender respectively, 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 $\mathcal{V}_1, \dots, \mathcal{V}_L$ 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 same vertex set $\mathcal{V}$. Furthermore, 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 $(v_1, v_2, \dots v_L) \in \mathcal{V}^1 \times \mathcal{V}^2 \times \dots, \times \mathcal{V}^L$ 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 this is a key contribution of this paper.

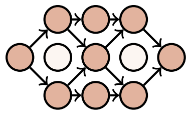
Targets and interdiction function. We assume that each target has an associated value given by $r^\odot : \mathcal{V}^\odot \to \mathbb{R}$. Typically, this value will be nonnegative. For some path $p \in \mathcal{P}_a \cup \mathcal{P}_d$, we write $r^\odot(p)$ as a shorthand for $r^\odot(p(L))$. We 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.

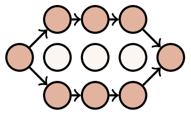
3.1 Player Utilities

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,

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


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


(b) $\mathcal{G}_d$

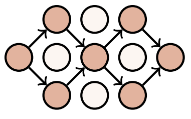
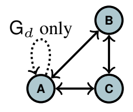
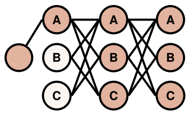

(c) $\mathcal{G}_a$

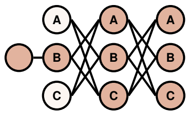
Figure 1: Layered graphs of Example 1. $\mathcal{V}$ has 5 layers with a single source and sink. Disconnected vertices in $\mathcal{G}_a$ and $\mathcal{G}_d$ are in white.



(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

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

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,* ***U****P or* ***D****OWN. 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. We will see shortly that interesting domains may be described by slightly modifying this unrolling process, potentially adding some application-specific auxiliary vertices and edges to this DAG. Due to space constraints, we will defer the detailed conversion from G_i to $\mathcal{G}_i$ for each application to the appendix (each conversion is quite natural).

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

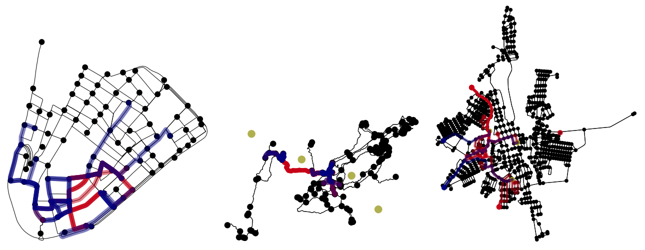


Figure 3: Real-world physical graphs used to generate LGSGs for our application domains, together with examples of defender's (red) and attacker's (blue) equilibrial paths in (a) PE, (b) AT, and (c) LI.

our framework admits many modifications, e.g., allowing the 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,

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

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*

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

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.

$$\min_{f_d \in \Gamma_d, g \in \mathbb{R}^{|\mathcal{V}|}} \quad g(v_{\text{source}})$$

$$g(e_a^+) - g(e_a^-) \geq \sum_{e_d \in \mathcal{E}_d} f_d(e_d) \cdot Q(e_d, e_a) \qquad \forall e_a \in \mathcal{E}_a$$

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 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 ATTACKERBR$(\widetilde{x}_d)$. ATTACKERBR can also be written as a MILP, the details of which are deferred to the

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:     $\widetilde{x}^*_d, \widetilde{x}^*_a \leftarrow$ NashEquilibrium$(\widetilde{\mathcal{P}}_d, \widetilde{\mathcal{P}}_a)$
4:     $p^{\text{BR}}_d, p^{\text{BR}}_a \leftarrow$ DefenderBR$(\widetilde{x}^*_a)$, AttackerBR$(\widetilde{x}^*_d)$
5:     $\widetilde{\mathcal{P}}_d, \widetilde{\mathcal{P}}_a \leftarrow \widetilde{\mathcal{P}}_d \cup \{p^{\text{BR}}_d\}, \widetilde{\mathcal{P}}_a \cup \{p^{\text{BR}}_a\}$
6: until EquilibriumGap$(\widetilde{x}^*_d, \widetilde{x}^*_a, p^{\text{BR}}_d, p^{\text{BR}}_a) < \epsilon$
```

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 $r^{\odot}$.

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.

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+I) 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

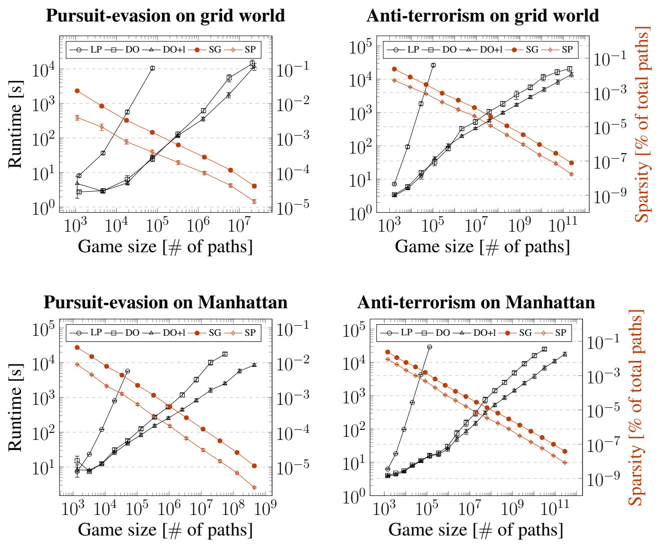


Figure 4: Computation times (black lines) and sparsity metrics (orange lines, for vanilla DO with exact best-responses) for PE and AT domains on grid world and Lower Manhattan.

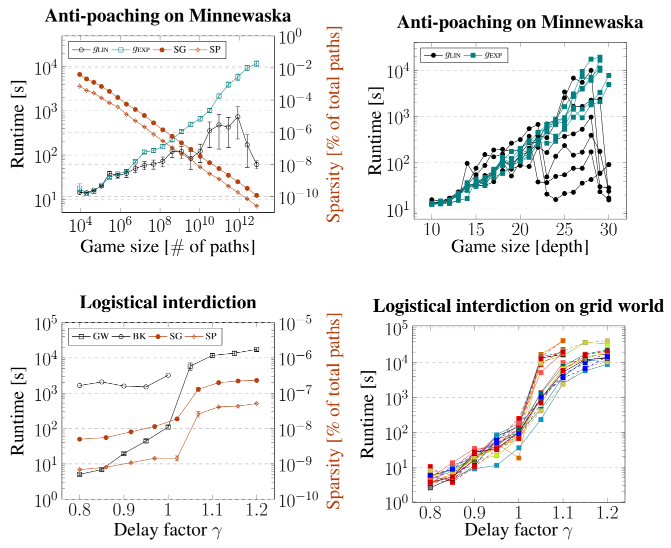


Figure 5: Computation times and sparsity metrics for AP and LI domains demonstrating the effects of additional factors on the performance of our DO algorithm with approximate best-responses.

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 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 highlight the importance of structure over game size when estimating computational costs.

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.

[Hagberg *et al.*, 2008] Aric Hagberg, Pieter Swart, and Daniel S Chult. Exploring network structure, dynamics, and function using networkx. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.

[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.

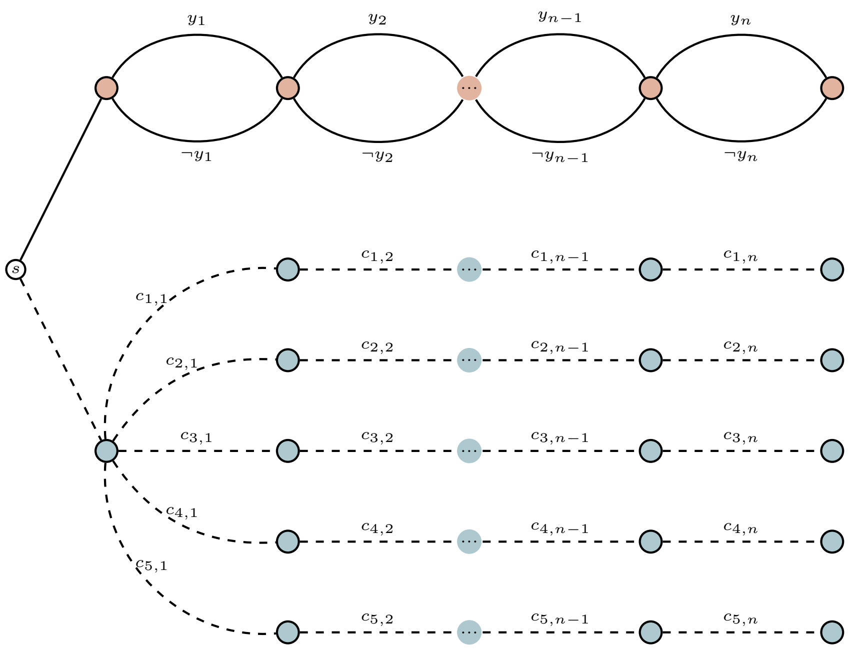


Figure 6: The graph described in Proposition 1. Full and dashed edges denote edges present in $\mathcal{G}_d$ and $\mathcal{G}_a$ respectively. Edges between orange vertices "belong" to the defender and correspond to variable assignments, while paths in the blue vertices denote a single clauses.

A Proofs

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

Proof. The proof is very similar the reduction of 3-SAT to the Escape Interdiction Game (EIG) of Zhang *et al.* [2017], but much simpler because of the generality of R admitted. Let there be n variables and m clauses. They are denoted by $Y_1, \ldots, Y_n$ and $C_1, \ldots, C_m$ respectively.

The layered graphs $\mathcal{G}_d$ and $\mathcal{G}_a$ can be broken into two parts, the vertices and edges of which are shown in Figure 6. Lines drawn in solid edges belong to the defender, who selects an variable assignment in a "chain graph" with vertices given in orange. The chain graph has $n+1$ vertices (including a dummy terminal vertex), the first n of which contains two outgoing edges represents y_i or $\neg y_i$ respectively. Clearly, every path formed this way corresponds to an assignment to $y_1, \ldots y_n$. The dotted lines are for the attacker, and the reachable vertices given in blue. The attacker (tries to) selects a clause which the variable assignment violates.

We now define $R(y_i, c_{j,i}) = 1$ if and only if the variable Y_i belongs to clause C_j, and correspondingly $R(\neg y_i, c_{j,i})$ if and only if $\neg Y_i$ belongs to clause C_j. All other edges across different layers, or belonging to the same player (e.g., both dotted or solid) have $R = 0$. All terminal vertices have a value of 1, i.e., $r(v) = 1$ for all $v \in \mathcal{V}^{\odot}$. We claim that the value of the game is 0 if and only if the formula is satisfiable.

($\Longleftarrow$). If the formula is satisfiable, then the defender can choose that path in solid edges, and whatever the attacker chooses will at least be interdicted once (by the definition of satisfiability and R). This gives 0 payoff to the attacker.

($\Longrightarrow$). Now, suppose the value of the game is 0. We look at the defender's strategy. If it was pure, then it is a single path and therefore corressponds to a variable assignment. Since that strategy constitutes a (defender) NE in a game with value 0, any best-response from the attacker cannot give any attacker payoff greater than 0. This also means there is no clause-path the attacker can choose that can ever avoid being interdicted (i.e., all clauses are satisfied). Now, if the defender's strategy is *not* pure, then it must be some mixture of paths x_d^*. Take any path p_d that is played with strictly positive probability, i.e., $x_d^*(p_d) > 0$; we claim playing p_d deterministically must also constitute a pure Nash. This is because 0 is already the best possible outcome for the defender (who is the min-player). That is, if it was not Nash, then there must be a best-response $p_a \in \mathcal{P}_a$ that interdicts p_d. This implies that the attacker could have played p_a against x_d^* and obtained some strictly positive payoff, contradicting the claim that x_d^* is a NE (and that 0 is the value of the game). Either way, there exists some pure strategy NE for the defender, implying the formula is satisfiable.

$\square$

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

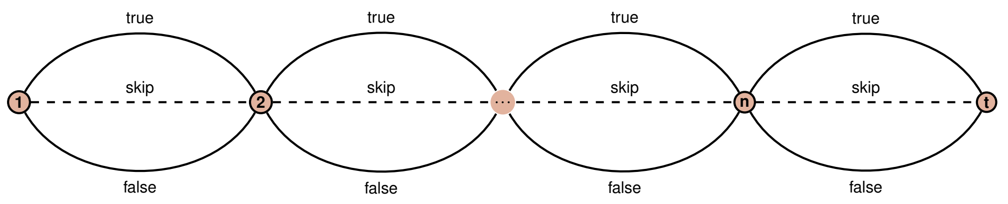


Figure 7: The graph described in Proposition 5, representing both $\mathcal{G}_i$ (indicated by full lines) and $\mathcal{G}_{-i}$ (represented by full and dashed lines), serving as a game for the SAT problems we aim to reduce from. Each vertex, denoted by j, corresponds to a variable j in the SAT formula, which contains a total of n variables. Paths within these graphs function as encodings for either an assignment (pertaining to player i) or a clause (pertaining to player $-i$).

in their flows $f_d(x_d)$ and $f_a(x_a)$. Specifically,

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

Proof. We proceed by rearranging the formula for computing the expected utility given distributions over paths $x_d \in \Delta_d$ and $x_a \in \Delta_a$.

$$\begin{aligned}
u_{\text{LIN}}(x_d, x_a) &= \sum_{p_d \in \mathcal{P}_d} \sum_{p_a \in \mathcal{P}_a} x_a(p_a) x_d(p_d) u_{\text{LIN}}(p_d, p_a) \\
&= \sum_{p_d \in \mathcal{P}_d} \sum_{p_a \in \mathcal{P}_a} \sum_{e_d \in p_d} \sum_{e_a \in p_a} x_a(p_a) x_d(p_d) Q(e_d, e_a) \\
&= \sum_{e_d \in \mathcal{E}_d} \sum_{e_a \in \mathcal{E}_a} \sum_{p_d \in P_d(e_d)} \sum_{p_a \in P_a(e_a)} x_a(p_a) x_d(p_d) Q(e_d, e_a) \\
&= \sum_{e_d \in \mathcal{E}_d} \sum_{e_a \in \mathcal{E}_a} Q(e_d, e_a) f_d(e_d) f_a(e_a),
\end{aligned}$$

□

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.*

Proof. For the defender's best-response, we employ reductions from MAX-SAT. Consider a CNF with n variables and m clauses. The LGSG we construct has $n+1$ vertices organized in a row, one for each variable and a final terminal vertex. For $\mathcal{E}_d$, each non-terminal layer contains two outgoing parallel edges to the next vertex, these signify positive or negative assignments. Clearly, each assignment is a valid defender path in $\mathcal{P}_d$ and vice versa. $\mathcal{E}_a$ is the same, except that there is an extra 'skip' edge between adjacent variables. See Figure 7 for an illustration. $\widetilde{\mathcal{P}}_a$ comprises m paths played uniformly at random. Each of these paths corresponds to a clause: if variable j is True in the clause, the attacker takes the top edge; if False, it takes the bottom path; if j is not in the clause, it takes the skip edge. Since exactly one of the 3 edges per layer is taken, this describes a path in $\mathcal{P}_a$. We set $R(e_d, e_a) = \mathbb{1}[e_d = e_a]$ and $r^{\odot} = m$. Hence, each attacker path in $\widetilde{\mathcal{P}}_a$ interdicted by the defender signifies a satisfied clause, and reduces the attacker payoff by 1 (starting from m). Since the defender seeks to interdict as many paths in $\widetilde{\mathcal{P}}_a$ as possible, this corresponds to solving the MAX-SAT problem. A similar reduction from MIN-SAT shows NP-hardness of the attacker best-response.

□

B Attacker's Best Response MILP

We overload the function R to denote for a given defender's path p_d the subset of attacker's edges the path p_d can interdict as $R_d(p_d)$, formally

$$R_d(p_d) = \{e_a : \exists e_d \in p_d R(e_d, e_a) = 1\}.$$

The attacker's best-response MILP has then the following form:

$$
\begin{aligned}
&\arg\max_{f_a, y, z} \sum_{v \in \mathcal{V}^L} r(v) \cdot z_v \\
&z_v \le \sum_{e_a \in E_a^+(v)} f_a(e_a) && \forall v \in \mathcal{V}^L \\
&z_v \le 1 - \sum_{p_d \in \widetilde{\mathcal{P}}_d} y(p_d) \cdot x_d^*(p_d) && \forall v \in \mathcal{V}^L \\
&y(p_d) \ge f_a(e_a) && \forall p_d \in \widetilde{\mathcal{P}}_d, e_a \in R_d(p_d) \\
&1 = \sum_{e_a \in \mathcal{E}_a^-(\mathcal{V}^1)} f_a(e_a) \\
&0 = \sum_{e_a \in \mathcal{E}_a^-(v)} f_a(e_a) - \sum_{e_a \in \mathcal{E}_a^+(v)} f_a(e_a) && \forall v \in \mathcal{V} \backslash \{\mathcal{V}^1, \mathcal{V}^L\} \\
&f_a(e_a) \in \{0, 1\} && \forall e_a \in \mathcal{E}_a \\
&0 \le z_v, y(p_d) \le 1 && \forall p_d \in \widetilde{\mathcal{P}}_d, v \in \mathcal{V}^L
\end{aligned}
$$

The MILP structure for the attacker mirrors that of the defender. Once again, we represent the best-responding path as a binary flow, along with corresponding conservation constraints. The variable $y(p_d)$ serves as an indicator of the interdiction of the attacker's best-response path by the defender's path p_d. The first two constraints set the probability z_v of safely reaching the target v in the last layer of the layered graph in relation to the defender's interdicting paths. It is noteworthy that only one z_v will have a non-zero value. The objective then focuses on maximizing the attacker's utility by seeking to maximize z_v weighted by the target's value.

This MILP also grows in size with the opponent's subgame $|\widetilde{\mathcal{P}}_d|$, and becomes harder to solve as the subgame $|\widetilde{\mathcal{P}}_d|$ increases.

C Implementation Details

In principle, the subgame LP solved in NASHEQUILIBRIUM$(\widetilde{\mathcal{P}}_d, \widetilde{\mathcal{P}}_a)$, as well as the best-response MILP oracles that are solved in DEFENDERBR$(\widetilde{x}_a^)$ and ATTACKERBR$(\widetilde{x}_d^)$ in Algorithm 1, can be reconstructed from scratch in every iteration of the double oracle. However, in practical terms, it proves advantageous to update the existing Gurobi models and re-solve them. This approach allows Gurobi to leverage the previously computed results, resulting in decreased computation time.

The update of the Nash LP is straightforward, involving the addition of an extra best-response constraint or appending a new variable to the existing set of best-response constraints and the probability simplex constraint. However, updating the oracles is a more intricate task. Given that neither oracle relies on the player's own subgame, each oracle needs to be updated on two events: when the opponent's subgame increases and when the opponent's subgame strategy undergoes a change. In most cases, both events occur in each iteration of the algorithm.

Introducing a new attacker's path p_a to the defender's MILP oracle leaves the existing constraints unchanged. However, it necessitates the creation of a new variable $y(p_a)$, the construction of a new constraint

$$y(p_a) \le \sum_{e_d \in \{e_d : \exists e_a \in p_a R(e_d, e_a) = 1\}} f_d(e_d),$$

and the inclusion of the variable $y(p_a)$ in the objective. Any change in the attacker's subgame strategy x_a^* then involves a modification of the coefficients in the objective.

In contrast, an increase in the defender's subgame and a change in their subgame strategy affect even the already existing constraints in the attacker's MILP. First, for a new path p_d, a new variable $y(p_d)$ is created. Similarly to the defender's MILP, the new set of constraints

$$y(p_d) \ge f_a(e_a) \qquad \forall e_a \in R_d(p_d)$$

is added to the MILP. The change in the defender's subgame strategy x_d^* then implies modifications of coefficients in all the interdiction-probability constraints

$$z_v \le 1 - \sum_{p_d \in \widetilde{\mathcal{P}}_d} y(p_d) \cdot x_d^*(p_d) \qquad \forall v \in \mathcal{V}^L.$$

Speedups for MILP Solvers

As we mentioned in Section 4.4, we attempted several speedups for the MILP solver, including (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 now describe briefly how these were performed (with an focus on (a)).

1. We set the appropriate parameter in Gurobi telling it to terminate when the computation time exceeds 1s. When that happens, it is almost always the case that we have found a better (but not best) response. We add this to the strategy set anyway, as if it was a best response. This works because often times, a relatively good solution to the MILP is found quickly, and Gurobi is spending most of its effort proving optimality. In addition, we set a parameter $\alpha \geq 1$. This is how frequent we force ourselves to solve the MILP exactly. For our experiments, we set $\alpha = 20$. Lastly, we stress that when using early termination it is crucial that we *do not* exit the loop in Algorithm 1 using strategies from the partially solved MILP. This is because the equilibrium gap will seem tighter than they actually are. One method is to only terminate when we are solving the MILP exactly, which occurs once every α iterations. In practice, when we notice that the equilibrium gap computed using approximate best-responses is very small, e.g., 10ϵ, we will double the running time and resolve the MILP. This is repeated until the gap becomes large, or the MILP is solved fully and still gives a gap $< \epsilon$. This resolving does not significantly increase running times, since we do not resolve from scratch: Gurobi is able to continue the MILP solver from when it left off (we have not made any changes the the model).

2. In our initial implementation, we removed paths (strategies) from $\widetilde{\mathcal{P}}_i$ when it has not been included in any Nash equilibrium for more than 50 iterations. By keeping the set $\widetilde{\mathcal{P}}_i$ small, we hope to keep the resultant MILPs small. We also made sure we do not cycle around by repeatedly removing the same path—this is done by keep track of paths removed and ensuring that each path is removed from $\widetilde{\mathcal{P}}_i$ at most once. Contrary to what we may hope for, apart from the first 50 or so paths, most paths turn out to be not terribly bad in practice and are frequently included in *some* NE, albeit not all of them. Therefore, almost no paths were removed. We noticed that different combinations of paths "work well" together. Often times DO will alternate between improving these different combinations, making all of them useful in some sense.

3. We attempted to add some additional constraints that we know are true (from our problem formulation) but are not immediately derived from the MILP. One such constraint is to set $y(p_a)$ to be binary (in the defender best response). At first glance, we are increasing the number of binary variables and could make the problem more difficult. However, this should not be the case since we are indicating to Gurobi that y is a variable that could be branched on (instead of just x's). This sometimes yielded faster performance, but we did not see a noticeable difference. Another interesting example is to constrain the number of nonzero x's in each layer to be no more than one (in fact it will be exactly 1). This can be derived from the flow constraints in Γ_i. Interestingly, when implemented as a Special Ordered Set of type 1 (SOS1), the solver performs significantly faster ($\sim$ 15 %) most of the time. We suspect the inclusion of this SOS constraint helps Gurobi perform selection for variable branching slightly differently. Unfortunately, this improvement was not consistent over all our experiments (though including it seems to at least cause no harm)

D Additional Experimental Data

In this section, additional details and experimental results are presented, which were excluded from the main text due to space limitations. The complete code for all experiments will be released upon acceptance of the paper.

D.1 Detailed Description of Physical Graphs

Recall that G_d and G_a are graphs that we "unroll" over time to obtain the layered graphs $\mathcal{G}_d$ and $\mathcal{G}_a$. This unrolling may depend be application specific, and we may add additional vertices to allow for dependencies on a small portion of history. This is particularly important for Anti-Terrorism(AT) and Logistical Interdiction/Persistent Threats(LI/PT). The physical worlds G_d and $\mathcal{G}_a$ are designed to be independent of this unrolling routine (both in theory and in implementation).

We will first discuss the physical worlds and how they are parameterized and generated in the context of our experiments. After that, we will move on to describe the individual unrolling process for PE, AT and LI/PT applications.

Grid World. The grid world is simply a $S \times S$ grid of vertices arranged in a rectangle of length $(S-1) \times (S-1)$. Each vertex is *potentially* adjacent to its 4-neighbors, with no wraparound for vertices at the edges. The vertices are identical, but edges could be different between G_d and G_a. This models situations where some roads are available to one player but not the other: e.g., security personnel have shortcuts they may take, while the attacker may take illegal and dangerous actions (e.g., driving against traffic) that the defender has to avoid.

The generation process for the Grid World (GW) takes two parameters, an integer $S > 1$, the width and height of the grid, and optional parameters $q_{d,\text{drop}}, q_{a,\text{drop}} \in [0, 1]$, the probability that an edge in the grid is independently dropped. Typically, $q_{i,\text{drop}}$ is quite low, often around 0.1. An example of G_i (for either player) is shown in Figure 3. There $q_{d,\text{drop}} = 0$ and $q_{a,\text{drop}} > 0$, i.e., some edges are off limits to the attacker.

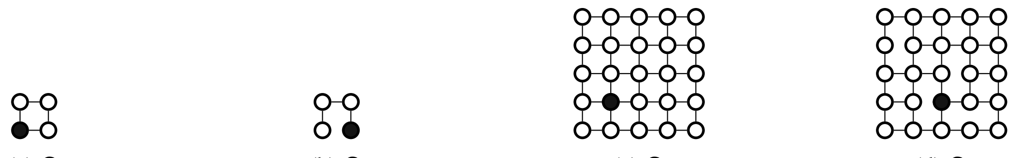

(a) G_d (b) G_a (c) G_d (d) G_a

Figure 8: Defender and attacker physical graphs for Grid Worlds (GW). Figure 8a and 8b give graphs for a toy 2×2 map, while Figure 8c and 8d give show an instance of size 5×5. Black dots represent starting positions for each player. We set $q_{a,\text{drop}} = 0$ and $q_{a,\text{drop}} > 0$.

Open Street Maps. Real-world maps, represented as Networkx graphs [Hagberg *et al.*, 2008] and obtained through the OSMnx framework [Boeing, 2017], undergo a series of processing steps to improve their applicability to our application domains. The initial step is to download the original graph using the GRAPH_FROM_PLACE method. Subsequently, the CONSOLIDATE_INTERSECTIONS method is used to merge nearby intersections, with a specified tolerance parameter set to 30. This merging process serves a dual purpose: firstly, it models the ability to interdict the attacker at nodes that are in close proximity to each other. Secondly, it contributes to reducing the overall number of nodes in the graphs, streamlining the computation while maintaining the essential features of the real-world layout. Following the intersection consolidation, loops are added to the graph, enabling players to stay at the same vertex for multiple timesteps. Next, the edges of the graph are discretized using the UTILS_GEO.INTERPOLATE_POINTS method into sub-edges of place-specific unit length. This discretization serves the purpose of equalizing the distance that can be traversed in a single timestep across different locations in the graph.

In Lower Manhattan (place="Financial district, NYC, USA"), the unit distance is set to 80. In the case of Minnewaska State Park (place="Minnewaska State Park, NY, USA"), a unit distance of 300 is set. Lastly, for Bakhmut (place="Bakhmut, Ukraine"), the designated unit distance is 200. These variations in unit distance cater to the unique geographical characteristics of each location, e.g., on Lower Manhattan it is set to a typical width of a Manhattan block of houses.

D.2 Detailed Description of Application Domains and Unrolling them Over Time

Recall that we need to go from the physical maps G_i and the underlying task (e.g., pursuit-evasion) to a LGSG, which requires the following components.

- Layered graphs $\mathcal{G}_d$ and $\mathcal{G}_a$
- Target values $r^{\odot}(v)$ for terminal states $v \in \mathcal{V}^{\odot}$
- Interdiction function R.

In most of our use cases, we find that the the number of layers corresponds to the time horizon T. Indeed, since trees with depth T are also layered graphs of depth T, we can trivially convert G_i to $\mathcal{G}_i$ by unrolling a search *tree* (including any extra actions needed, e.g., planting an explosive in AT games) from its root (which would correspond to the starting location in the physical graph, taking the union of the vertices produced, and by setting r and R accordingly based solely on the leaves and the edges from the second-last layer (both have a 1-1 correspondence to paths for each player). Of course, this is a bad idea since $|\mathcal{V}^\ell|$ now grows exponentially in ℓ: we are no longer utilizing the fact that the utility u has some "nice" structure. We now describe the unrolling process for PE, AT and LI/PT. We remind the reader that we are focusing on binary reward models.

Pursuit-Evasion (PE)

Pursuit evasion is the simplest variant of unrolling. For each player $i = \{d, a\}$, we perform the following. In each layer (except for the first, which is a singleton) we have $|\mathsf{V}|$ vertices, which we call $v_{\ell,\mathsf{v}}$, for layers $\ell \in \{2, ...T+1\}$ and $\mathsf{v} \in \mathsf{V}$. For layer $\ell \geq 2$, we write their outgoing edge sets for player i

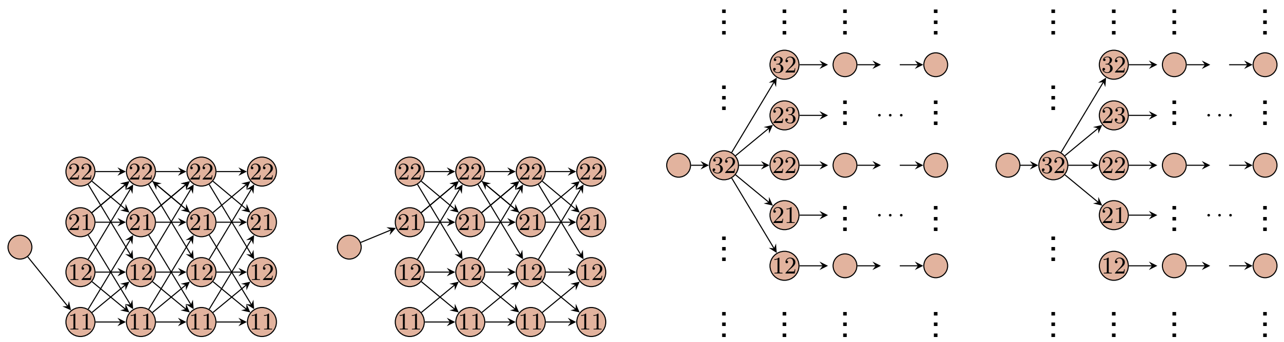
$$\mathcal{E}_i^\ell = \left\{ (v_{\ell,\mathsf{v}}, v_{\ell+1,\mathsf{v}'}) \middle| \underbrace{(\mathsf{v},\mathsf{v}') \in \mathsf{E}_i}_{\text{move } \mathsf{v} \to \mathsf{v}'} \vee \underbrace{(\mathsf{v} = \mathsf{v}')}_{\text{stay in } \mathsf{v}} \right\}, \tag{4}$$

and for the first layer (a singleton containing the source)

$$\mathcal{E}_i^1 = \left\{ (v_{\text{source}}, v_{2,\mathsf{v}}) \middle| \mathsf{v} \in \mathsf{V}_i^{\text{start}} \right\},$$

where $\mathsf{V}_i^{\text{start}}$ is the set of possible starting vertices in the physical graph. Note that in (5) we allow for either player to "stay" in its current vertex if it desires (this is equivalent to having G_i contains self-loops). Finally, the set of edges is given by the union over these disjoint sets

$$\mathcal{E}_i = \bigcup_{\ell=1}^{T} \mathcal{E}_i^\ell.$$



(a) $\mathcal{G}_d$ for 2×2 GW (b) $\mathcal{G}_a$ for 2×2 GW (c) $\mathcal{G}_d$ for 5×5 GW (d) $\mathcal{G}_a$ for 5×5 GW

Figure 9: Unrolled layered graphs for the grid world (GW) under pursuit evasion (PE). Vertices are labeled according to their xy-coordinate, with the bottom left of the grid being 11. Figures 9a and 9b show $\mathcal{G}_d$ and $\mathcal{G}_a$ for the 2×2 grid worlds of Figures 8a and 8b respectively for a horizon of $T = 4$. Figures 9c and 9d show the same for the 5×5 variant shown in Figures 8c and Figures 8d. Note that the $\mathcal{G}_d$ and $\mathcal{G}_a$ differ.

Figure 9 shows examples of how this is done for the grid worlds of Figure 8 for grid worlds of size 2×2 and 5×5 respectively.

Next we will specify R. Recall that in our experimental setup of Section 5, we set R to capture the idea that interdiction occurs when players share a vertex, rather than edge. This is in order to be consistent with other related work, such that that of Zhang *et al.* [2017]. This interdiction function may be written as

$$R\left(e_d := (u_d, v_d), e_a := (u_a, v_a)\right) = \begin{cases} 1 & v_d = v_a \\ 0 & \text{otherwise} \end{cases}$$

or equivalently

$$R\left(e_d, e_a\right) = \begin{cases} 1 & e_d^+ = e_a^+ \\ 0 & \text{otherwise.} \end{cases}$$

Lastly, we specify $r^{\odot}(v)$, where v lies in the final layer, i.e., $v^{\odot} \in \mathcal{V}^{\odot} = \mathcal{V}^L$. If the goal is simply to evade capture, we can set $v^{\odot} \equiv 1$, since this means that the evader (attacker) is not concerned with the vertex it ends up in at the end of the game, as long as it was not interdicted. If, however, there is some preference based on some non-negative function $r(\mathsf{v})$ for $\mathsf{v} \in \mathsf{V}$ (which is the case we use in our experiments), we set

$$r^{\odot}(v := (L, \mathsf{v})) = r(\mathsf{v}).$$

Anti-Terrorism (AT)

For AT, the situation is similar; however, we will introduce T_{setup} extra vertices in order to keep track of how long the explosive has been set up. These vertices are only reachable by the attacker, and while in them, the only action that may be taken is to advance time. Therefore, the vertices in layers $\ell \in \{2, ..., T+1\}$ are given by $v_{\ell,\mathsf{v},w}$, where $w \in \{0, 1, \dots, T_{\text{setup}}\}$, and will be 0 if the explosive has not yet been placed. This is a significant increase in the size of $\mathcal{G}_d, \mathcal{G}_a$ compared to the previously discussed PE, but still manageable overall (and *much* better than expanding a tree).

As with before, we write for layers $\ell \geq 2$, their outgoing edge sets for player i, *assuming the explosive has not been planted*

$$\mathcal{E}_i^\ell = \left\{ (v_{\ell,\mathsf{v},0}, v_{\ell+1,\mathsf{v}'}) \middle| \underbrace{(\mathsf{v}, \mathsf{v}', 0) \in \mathsf{E}_i}_{\text{move } \mathsf{v} \to \mathsf{v}'} \vee \underbrace{(\mathsf{v} = \mathsf{v}')}_{\text{stay in } \mathsf{v}} \right\}, \tag{5}$$

and for the first layer (a singleton containing the source)

$$\mathcal{E}_i^1 = \left\{ (v_{\text{source}}, v_{2,\mathsf{v}}, 0) \middle| \mathsf{v} \in \mathsf{V}_i^{\text{start}} \right\},$$

where $\mathsf{V}_i^{\text{start}}$ is the set of possible starting vertices in the physical graph; this essentially begins the game. So far, $\mathcal{E}_i^\ell$'s are formed the same way (though they may differ since E_i differs). However, recall that the attacker has the additional actions of setting up up the explosive. Hence for $\ell \geq 2$, the attacker has the option of planting the explosive, giving

$$\hat{\mathcal{E}}_a = \left\{ (v_{\ell,\mathsf{v},0}, v_{\ell+1,\mathsf{v},1}) | \ell \in \{2, ..., T\} \right\},$$

or, if the explosive is already planted, but not gone off the attacker has to wait there, incrementing the waiting count by one

$$\check{\mathcal{E}} = \left\{ (v_{\ell,\mathsf{v},w}, v_{\ell+1,\mathsf{v},w+1}) | \ell \in \{2, ..., T\}, w \in \{1, \dots, T_{\text{setup}}\} \right\}.$$

If the explosive has already gone off, i.e., $w = T_{\text{setup}}$, then we do not increment w

$$\check{\mathcal{E}}' = \{(v_{\ell,\mathsf{v},w}, v_{\ell+1,\mathsf{v},w}) | \ell \in \{2, ..., T\}, w = T_{\text{setup}}\}.$$

Finally, the set of edges for each player is given by the union over these disjoint sets,

$$\mathcal{E}_d = \bigcup_{\ell=1}^{T} \mathcal{E}_d^{\ell}, \quad \mathcal{E}_a = \left(\bigcup_{\ell=1}^{T} \mathcal{E}_a^{\ell}\right) \cup \hat{\mathcal{E}}_a \cup \check{\mathcal{E}}_a \cup \check{\mathcal{E}}'_a.$$

We illustrate the unrolled graph in Figure 10. For simplicity, we will only show this for the toy grid world of size 2 (Figure 8b), and only for the attacker. The defender graph is essentially the same as Figure 9a, except it has many additional vertices that are not reachable. Also, we assume that $T_{\text{setup}} = 1$.

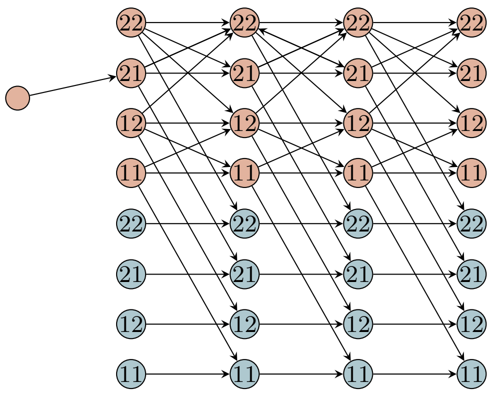


Figure 10: $\mathcal{G}_a$ by unrolling the 2×2 grid-world in Figure 8b under the AT domain with $T_{\text{setup}} = 1$ and $T = 3$. Vertices are labeled according their physical location. The color represents w, the time that has been spent setting up the explosive; orange vertices are when $w = 0$, blue vertices are when $w = 1$.

.

In Figure 10, the blue vertices are duplicates of the orange ones. These represent vertices where the explosive has been planted — once planted, the attacker stays in that physical location for the rest of the game. Also, since $T_{\text{setup}} = 1$, the outgoing edges in the blue vertices go horizontally (if $T_{\text{setup}} > 1$), then as w increases it will move to the next set of duplicated edges.

Now, we will specify R. If the explosive has not been planted, the interdiction function is easy. Let $e_d = (u_d, v_d), e_a = (u_a, v_a)$, where $v_d = (\ell_d + 1, \mathsf{v}_d, 0)$, $v_a = (\ell_d, \mathsf{v}_d, w)$ and $v_a = (\ell_d + 1, \mathsf{v}'_d, w')$. Then

$$R(e_d, e_a) = \begin{cases} 1 & \mathsf{v}_d = \mathsf{v}'_d \wedge \ell_d = \ell_a \wedge w = 0 \\ 1 & \mathsf{v}_d = \mathsf{v}'_d \wedge \ell_d = \ell_a \wedge w \neq w' \\ 1 & u_a = v_a \wedge \ell_d = \ell_a = 1 \\ 0 & \text{otherwise.} \end{cases}$$

The first case is the same as PE. The second case is when the defender interdicts the attacker *before* the explosive goes off (if it has already went off then we will have $w = w'$). The third case is the edge case where both players (choose to) start on the same vertex.

Finally, we need to specify r. Assume each physical target is worth $r(\mathsf{v}) \geq 0$. We simply set

$$r^{\odot}(v_{\ell,\mathsf{v},w}) = \begin{cases} r(\mathsf{v}) & w = T_{\text{setup}} \\ 0 & \text{otherwise,} \end{cases}$$

i.e., the explosive must have went off.

Logistical Interdiction (LI)

Recall that in LI, the utilities are dependent on the time that the attacker reached an escape point (if at all). Just like AT, we will have to add additional vertices that capture the notion of *time* that has passed since the escape t_{exit}.

Let the exit vertices in the physical graph be $\mathsf{V}_{\text{exit}} \subseteq \mathsf{V}$. In our layered graphs $\mathcal{G}_a$ and $\mathcal{G}_d$, we will once again have vertices for the physical location of each player. In addition, we have T "sink" vertices where the attacker is forced to enter once it exists. These special sink vertices may not be entered by the defender, and there is one of them for each timestep; this will help record when the escape occurred and will be used in $r^{\odot}$ to compute utilities u_{BIN}. We illustrate the unrolled *attacker* graph in

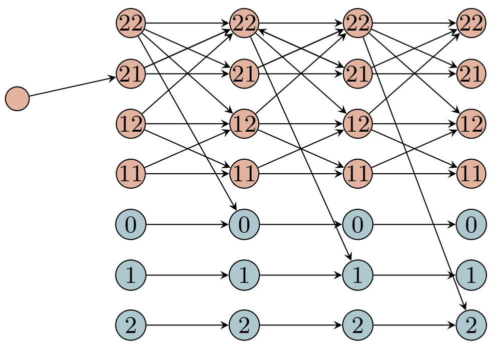


Figure 11: $\mathcal{G}_a$ by unrolling the 2×2 grid-world in Figure 8b under the LI domain with $T = 3$ and some $\gamma \geq 0$. Orange vertices are labeled according their physical location. Blue vertices (sinks) are labeled based on the time has passed when the attacker reached the exit.

.

Figure 11. We will assume that the escape point is in the top right vertex 22, which makes the problem trivial (the attacker can simply reach the exit by just moving up) but still worth illustrating.

From Figure 11, we can see several blue sinks denoting when the escape occurred. The vertex sets are of the form $v_{\ell,\mathsf{v}}$ *or*, for $\ell \geq 2$, the blue sinks $\hat{v}_{\ell,t}$ where t is the time that escape occurred. The edge sets include the usual edges in the physical world

$$\mathcal{E}_i^\ell = \left\{ (v_{\ell,\mathsf{v}}, v_{\ell+1,\mathsf{v}'}) \middle| \underbrace{(\mathsf{v},\mathsf{v}') \in \mathsf{E}_i}_{\text{move } \mathsf{v} \to \mathsf{v}'} \vee \underbrace{(\mathsf{v} = \mathsf{v}')}_{\text{stay in } \mathsf{v}} \right\}, \tag{6}$$

as well as the extra edges from exit vertices leading to blue “sinks” for the attacker ($\mathcal{G}_a$ does not have edges leading to the blue sinks.

$$\bar{\mathcal{E}}_a = \left\{ (v_{\ell,\mathsf{v}}, \hat{v}_{\ell+1,\ell-2}) \middle| \mathsf{v} \in \mathsf{V}_{\text{exit}}, \ell \geq 2 \right\}.$$

Finally, once the attacker has escaped it will stay there, leaving the t index untouched,

$$\hat{\mathcal{E}}_a = \left\{ (\hat{v}_{\ell,t}, \hat{v}_{\ell+1,t}) \middle| \ell \geq 2 \right\}.$$

The edge sets are their respective unions for each player,

$$\mathcal{E}_d = \bigcup_{\ell=1}^{T} \mathcal{E}_d^\ell, \quad \mathcal{E}_a = \left(\bigcup_{\ell=1}^{T} \mathcal{E}_a^\ell \right) \cup \bar{\mathcal{E}}_a \cup \hat{\mathcal{E}}_a.$$

Now, R is exactly the same as PE.

$$R(e_d, e_a) = \begin{cases} 1 & e_d^+ = e_a^+ \\ 0 & \text{otherwise.} \end{cases}$$

Finally, the target values $r^\odot$ are given by

$$r^\odot(\hat{v}_{L,t}) = \gamma^t$$

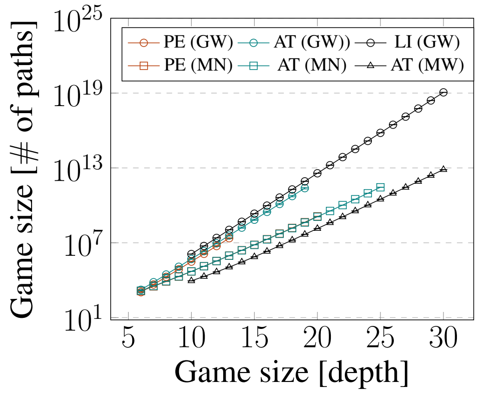
and

$$r^\odot(v_{L,\mathsf{v}}) = \begin{cases} \gamma^{L-2} & \mathsf{v} \in \mathsf{V}_{\text{exit}} \\ 0 & \text{otherwise.} \end{cases}$$

Where the second instance of $r^\odot$ is a special case which occurs when the attacker reaches the exit just when the game ends.

D.3 Application Domains Setups

In this subsection we explain how the individual physical graphs are setup to generate LGSGs for the application domains. For each scenario we depict the corresponding game size as a function of the depth of the layered graphs (i.e., the horizon) in Figure 12.



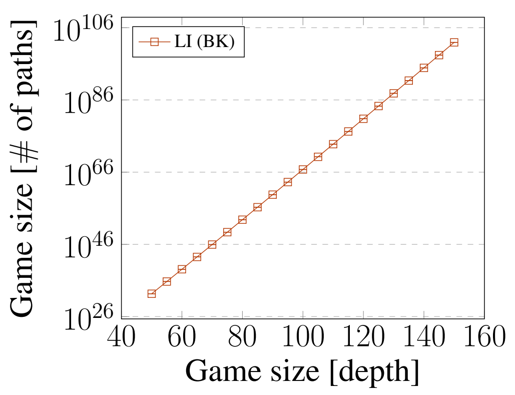


Figure 12: Game size as a function of the depth of the layered graph (i.e., the horizon) for each scenario we consider: pursuit-evasion (PE), anti-terrorism / anti-poaching (AT), and logistical interdiction / persistent threats (LI) on the grid world (GW), Lower Manhattan (MN), Minnewaska State Park (MW), and the city of Bakhmut (BK).

Pursuit-Evasion on grid world: In this scenario, players start from opposite corners, and target values are uniformly generated at random from the integral interval $[1, 10]$ for each node on the grid. For each player, edges are randomly dropped with a probability of 0.1.

Pursuit-Evasion on Lower Manhattan: In this setup, both players can select their starting points from the blue nodes shown in Figure 13. Similarly, target values are uniformly generated for each node from the integral interval $[1, 10]$. No edges are omitted for either player.

Anti-Terrorism on grid world: In this scenario, the defender starts from a grid corner, while the attacker starts from a node immediately adjacent. Target values are uniformly generated from the integral interval $[1, 10]$, and each edge is randomly dropped with a 0.1 probability for each player. The time required to detonate the explosive device is set to $T_{\text{setup}} = 2$.

Anti-Terrorism on Lower Manhattan: Here, the defender starts at the red node, and the attacker may choose a starting node from the adjacent blue nodes in Figure 13. Similar to previous scenarios, node targets have values generated uniformly from an integral interval $[1, 10]$. The time needed to set up the explosive device is $T_{\text{setup}} = 2$. No edges are omitted for either player.

Anti-Poaching on Minnewaska State Park: The defender begins at the red node in Figure 13, and the attacker selects from the set of blue nodes. Four habitats are sampled from the yellow diagonal area in the figure. Habitat animal scores are uniformly generated from the interval $(0, 1]$. The node scores are non-zero only on the intersections. In contrast to the attacker, the defender can traverse selected paved routes (indicated as paved by OSMnx) in one timestep, simulating vehicular movement, during which the attacker cannot be interdicted. It takes $T_{\text{setup}} = 2$ steps for the attacker to poach an animal.

Logistical Interdiction on grid world: In this setup, the defender starts at the center of the grid (unique in our 5×5 grid), and the attacker starts in a bottom corner. The only non-zero same-valued targets are the top corners, with values rescaled to the interval $(0, 10]$ based on the delay factor γ and reaching time. The defender can utilize the entire grid (no edges are dropped), while each edge in the attacker's graph is randomly omitted with a probability of 0.1.

Logistical Interdiction on Bakhmut: This scenario is inspired by the battlefront situation in Bakhmut on March 2, 2023. The Ukrainian supply forces can choose a starting node for the supply run from the blue node in Figure 13. Russian forces start in one of the red nodes. The objective for Ukrainian forces is to safely reach one of the yellow targets, the only non-zero same-valued targets on the map. Similar to previous scenarios, target values are rescaled to the interval $(0, 10]$ based on the delay factor γ and reaching time. The graph is the same for both players.

D.4 Detailed Experimental Setup

The experiments were conducted on an Intel Xeon Gold 6226 operating at 2.9Ghz, restricted to 8 threads and equipped with 32GB of RAM. The optimization tasks were undertaken with the Gurobi Optimizer version 10.0.3, build v10.0.3rc0 [Gurobi Optimization, LLC, 2023], on a Linux 64-bit platform. The double oracle algorithm was implemented in Python 3.7.9 and configured with a tolerance setting of $\epsilon = 10^{-3}$. The real-world graphs were obtained using OSMnx 1.8.1 [Boeing, 2017]. To ensure statistical robustness, 20 instances of each game were constructed and solved for any game involving randomness. In the reported graphs, we often depict the game size on one axis, defining it as the sum of the number of paths for each player.

D.5 Additional Experimental Results

In this section, we present additional empirical results that could not be accommodated within the main text's experimental section. Figure 14 illustrates how all the algorithms, including the full LP solver and the double oracle with exact or approximate

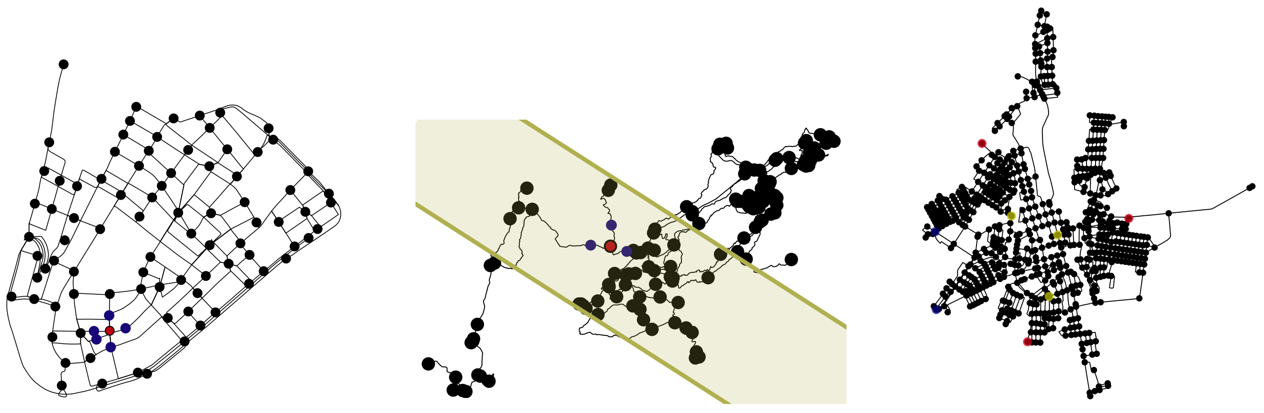

(a) Lower Manhattan, NYC, USA (b) Minnewaska State Park, NY, USA (c) City of Bakhmut, UKR

Figure 13: Starting points in our real-world physical graphs used to generate LGSGs for our application domains for the defender (red point(s)) and attacker (blue point(s)) in (a) anti-terrorism, (b) anti-terrorism, and (c) logistical interdiction.

best-responses, scale with respect the number of paths in the corresponding layered graph. We conducted these experiments in scenarios where we controlled computational difficulty through a different parameter or when not all the algorithms were compared in the main text.

The observed results are generally as expected, with a few noteworthy observations. Particularly in the case of logistic interdiction on the map of Bakhmut, the difficulty does not immediately increase with size, although a slight upward trend is noticeable. Additionally, a higher delay factor tends to correlate with longer solving times, particularly for smaller layered graphs, although this is not as pronounced as in grid worlds.

Subsequently, in Figure 15, we contrast the memory requirements of the double oracle variants by assessing the relative sizes of the subgames and supports in all scenarios considered in this paper. Overall, the distinction between both variants is minimal across all games, as indicated by the teal lines (DO with approximate BRs) overlapping the black lines (DO with exact BRs). For logistic interdiction, we additionally examine the sparsity of the double oracle across two different delay factors. As anticipated, we note that a higher delay factor results in larger subgames and supports, as the attacker is motivated to prioritize safety over speed more than with lower values.

For completeness, Figure 16 illustrates the typical evolution of equilibrium gaps over computation time and iterations in the double oracle with exact best responses for pursuit-evasion and anti-terrorism scenarios on both grid world and Lower Manhattan. An immediate trend observed is the slowdown of iterations (depicted on the top axis), particularly noticeable on the grid world, attributed to the increasing size of the subgame despite the the number of binary variables in the best-response MILPs being constant. Additionally, equilibrium gaps often experience a sudden drop as they approach the value of $\epsilon = 10^{-3}$.

Finally, in Figures 17, 18 and 19 we showcase qualitative results as typical solutions in the scenarios under consideration. First, in Figure 17, we highlight the distinctions between solutions on the exact same grid world with identical target values for both pursuit-evasion and anti-terrorism. We find the defender moves around the attacker's starting nodes, patrolling the area. In contrast, for anti-terrorism, it is more advantageous for the defender to prioritize high-value targets, considering the attacker's need to set up the explosive.

In Figure 18, we illustrate the typical impact of increasing the horizon for the anti-poaching scenario on the map of Minnewaska State Park, where the values of individual nodes represent the sums of the animal scores, attenuated by $g_{\text{LIN}}(z) = 1/z$, with z being the node's Euclidean distance to the habitat. At depth 20, the attacker still prioritizes nodes in the center of the graph and deploys a complex strategy targeting multiple nodes in the area. Upon reaching depth 21, the node closest to the upper-left high-value animal habitat becomes accessible, resulting in a shift in equilibrium towards this node and noticeably simplifying the strategy.

Figure 19 shows the equilibrium strategies for the logistical interdiction scenario on the map of the city of Bakhmut. The depiction illustrates how the strategies evolve with an increase in the horizon from 85 to 110 and a rise in the delay factor from 0.9 to 0.99. The results indicate that an increase in either parameter generally leads to more complex strategies by the players, with the distinction being more noticeable on the side of the delay factor.

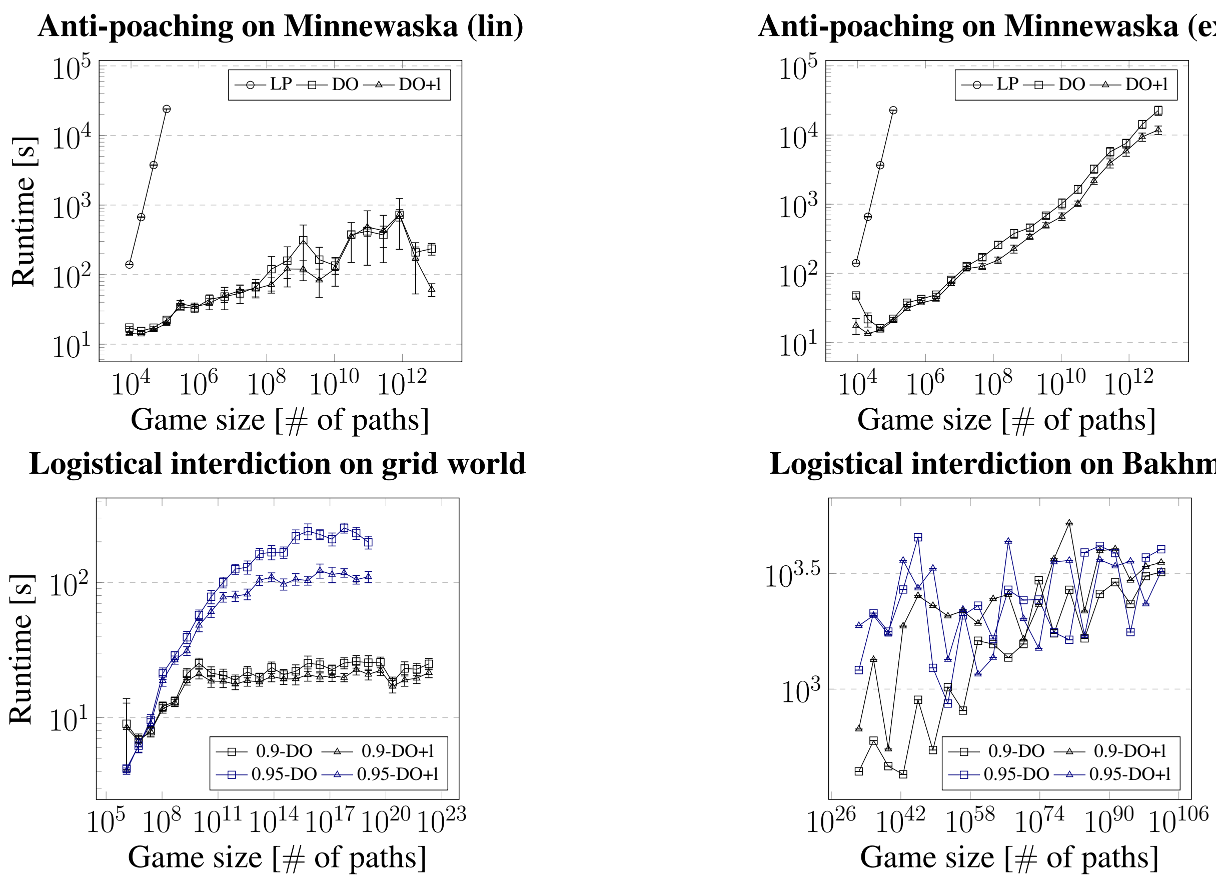


Figure 14: Computation times for anti-poaching on Minnewaska State Park and logistical interdiction on grid world and the city of Bakhmut. The complete LP is referred to as LP, while the double oracle is designated as DOfor the version incorporating exact best-responses and DO+lfor the version involving approximate best-responses. For logistic interdiction, we further compare how the algorithms scale for different values of fixed delay factor $\gamma \in \{0.9, 0.95\}$. The standard Nash LP is unable to compute equilibria even for the smallest logistical interdiction scenarios.

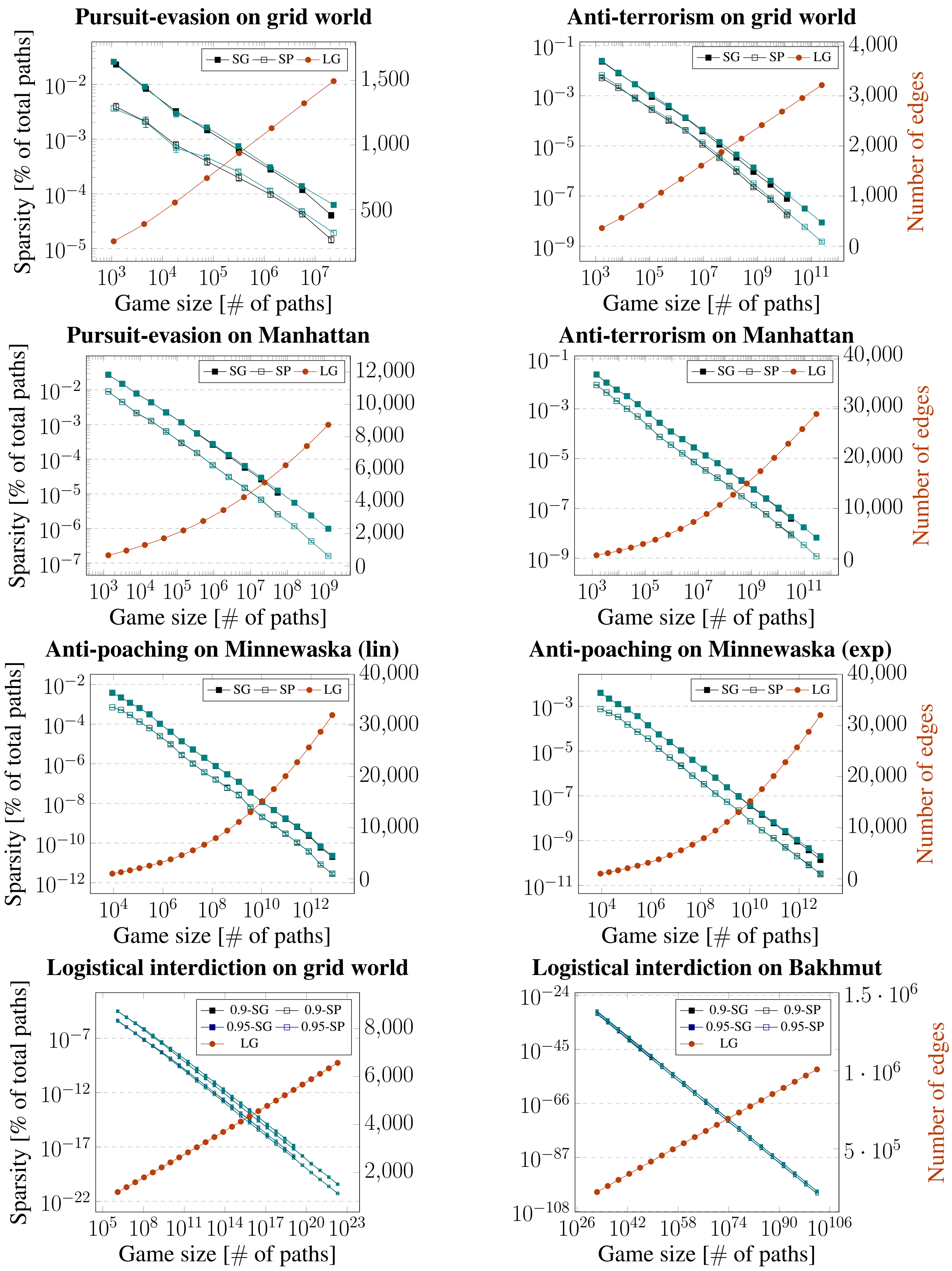


Figure 15: Comparison of sparsity metrics between vanilla double oracle with exact best response and double oracle with approximate best responses for each scenario we consider. Similarly as in the previous figure, for logistic interdiction, we further compare how the algorithms scale for different values of fixed delay factor $\gamma \in \{0.9, 0.95\}$. We depict also the size of the layered graph for each scenario by the number of its edges, corresponding to the number of binary variables in the best-response MILPs.

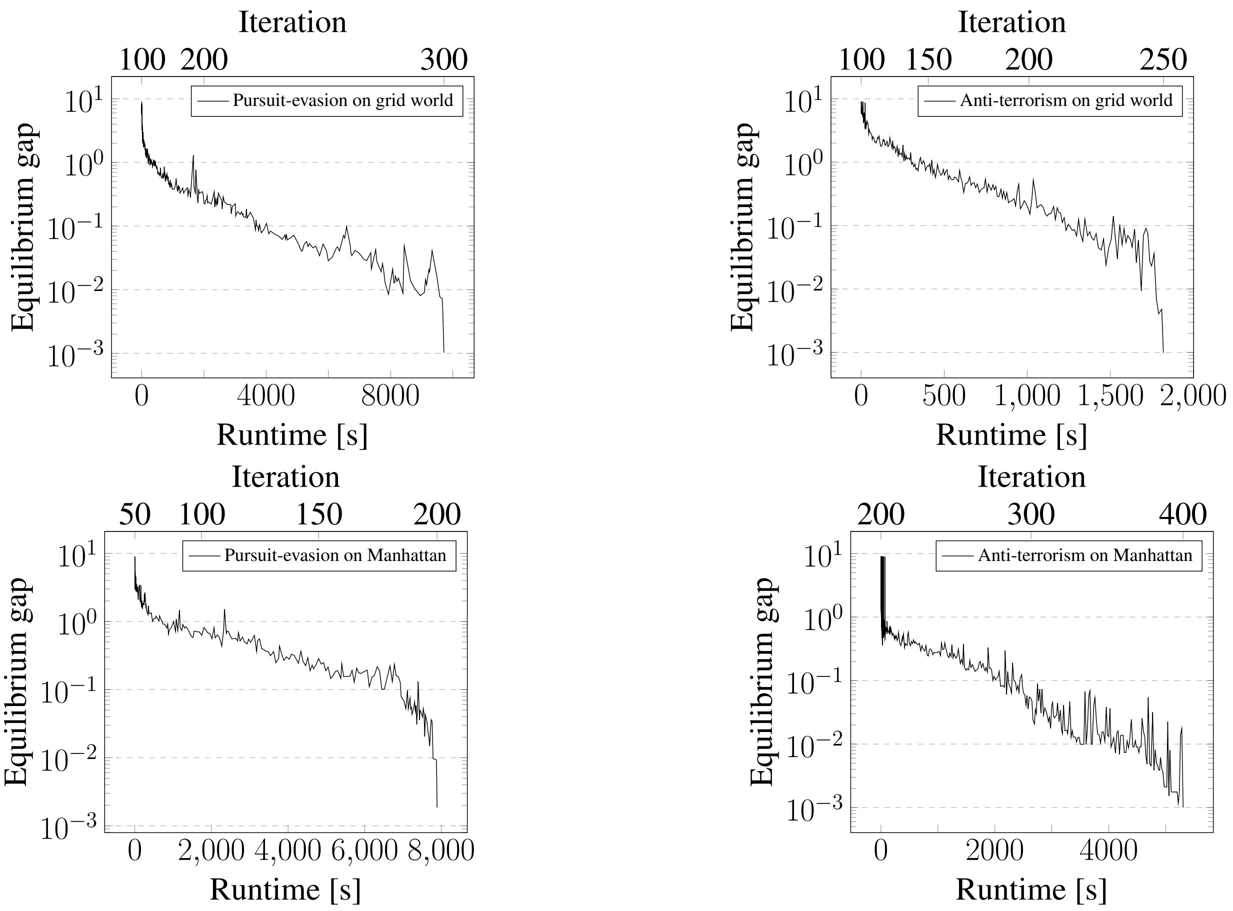
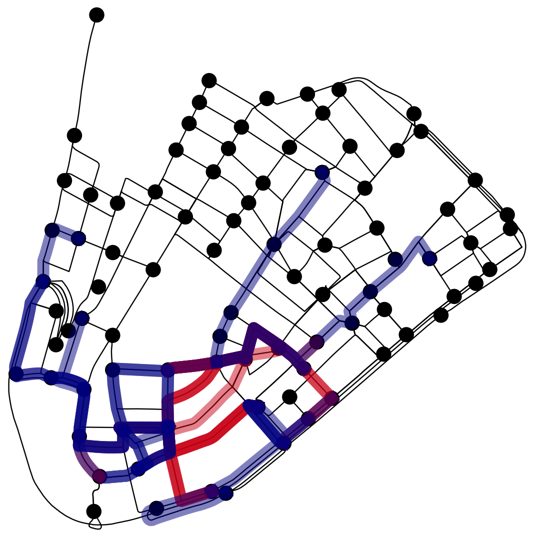


Figure 16: Vanilla double oracle with exact best-responses showcasing the evolution of equilibrium gaps over runtime and iterations in pursuit-evasion and anti-terrorism application domains across grid world and Lower Manhattan.


(a) Pursuit-evasion

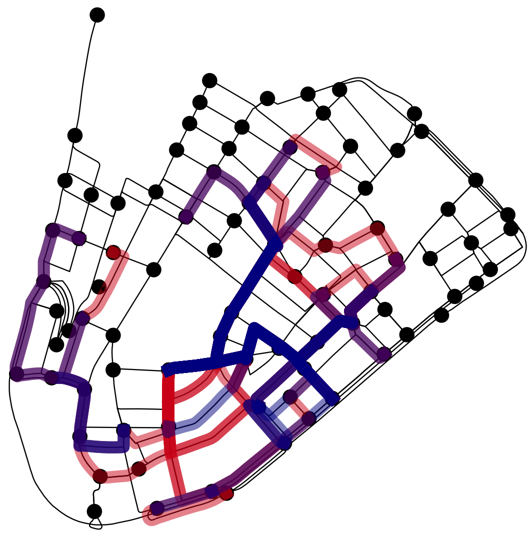
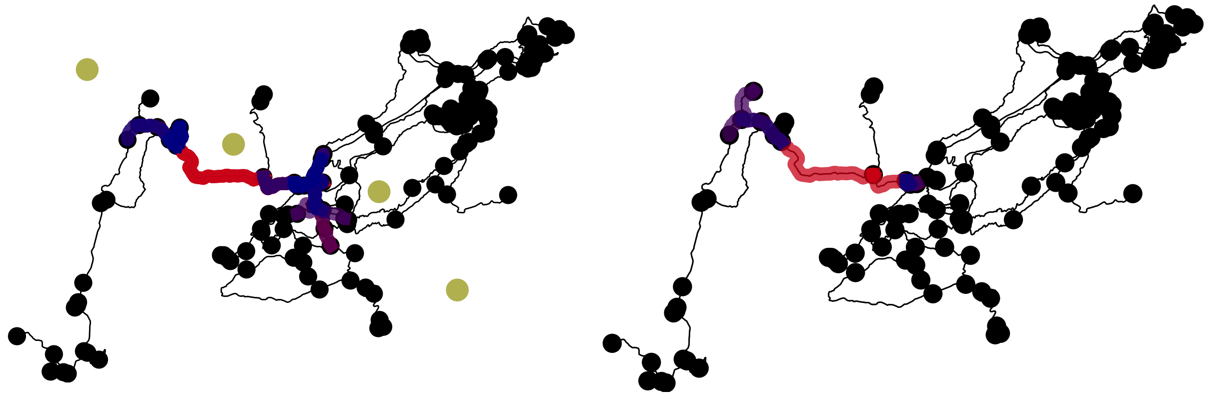

(b) Anti-terrorism

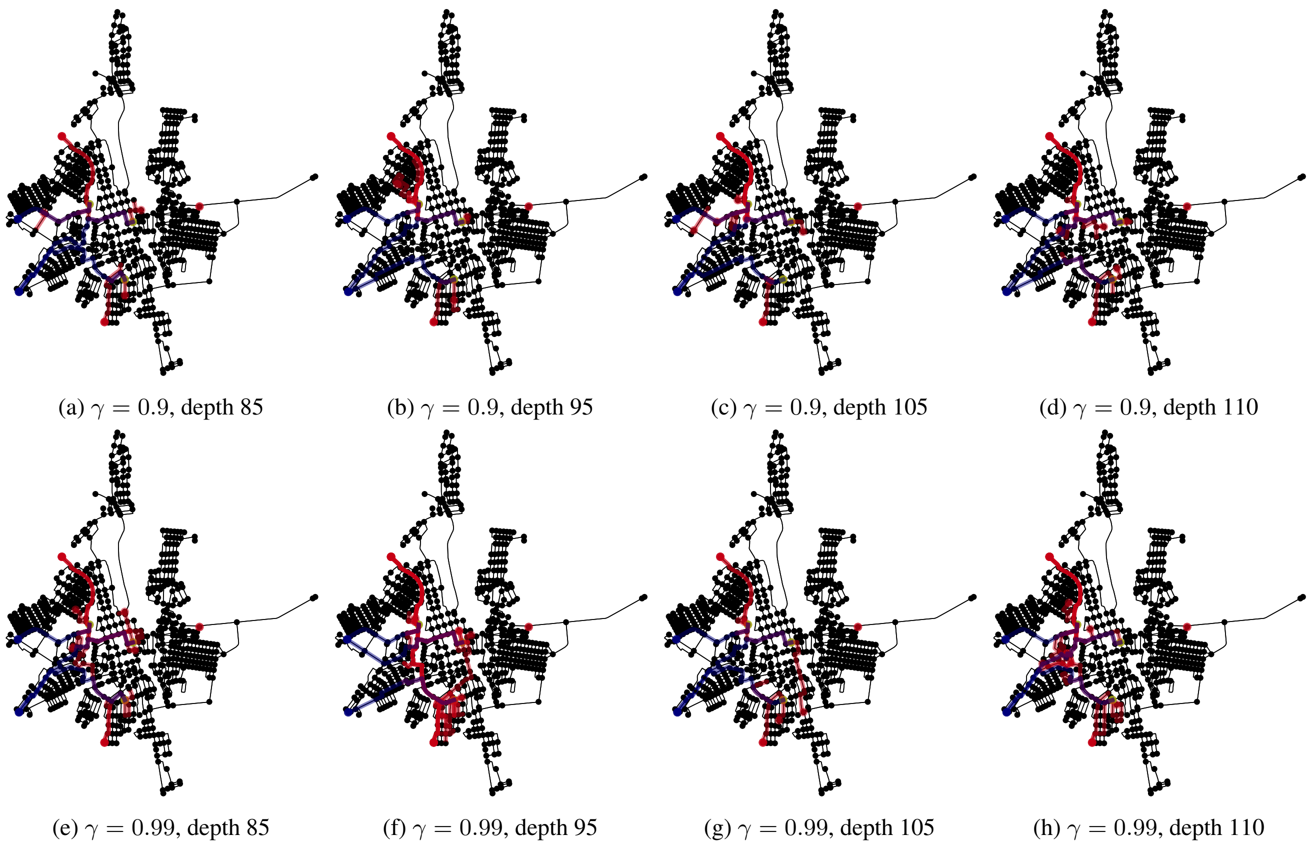
Figure 17: Illustration of defender's (red) and attacker's (blue) equilibrium paths in Lower Manhattan at a depth of 17.



(a) Depth 20

(b) Depth 21

Figure 18: Illustration of defender's (red) and attacker's (blue) equilibrium paths for anti-poaching in Minnewaska State Park using g_{LIN}. At depth 21, a high-value target becomes accessible.



(a) $\gamma = 0.9$, depth 85

(b) $\gamma = 0.9$, depth 95

(c) $\gamma = 0.9$, depth 105

(d) $\gamma = 0.9$, depth 110

(e) $\gamma = 0.99$, depth 85

(f) $\gamma = 0.99$, depth 95

(g) $\gamma = 0.99$, depth 105

(h) $\gamma = 0.99$, depth 110

Figure 19: Illustration of defender's (red) and attacker's (blue) equilibrium paths for logistical interdiction in the city of Bakhmut at depths $85, 95, 105$, and 110. The top row represents scenarios with a delay factor of 0.9, while the bottom row has a delay factor set to 0.99.