An Extended Study on Addressing Defender Teamwork while Accounting for Uncertainty in Attacker Defender Games using Iterative Dec-MDPs

Eric Shieh
Computer Science, University of Southern California
Los Angeles, CA, USA
eshieh@usc.edu

Albert Xin Jiang
Computer Science, Trinity University
San Antonio, TX, USA
xjiang@trinity.edu

Amulya Yadav
Computer Science, University of Southern California
Los Angeles, CA, USA
amulyaya@usc.edu

Pradeep Varakantham
Information Systems, Singapore Management University
Singapore
pradeepv@smu.edu.sg

Milind Tambe
Computer Science, University of Southern California
Los Angeles, CA, USA
tambe@usc.edu

November 30, 2015

Abstract

Multi-agent teamwork and defender-attacker security games are two areas that are currently receiving significant attention within multi-agent systems research. Unfortunately, despite the need for effective teamwork among multiple defenders, little has been done to harness the teamwork

research in security games. The problem that this paper seeks to solve is the coordination of decentralized defender agents in the presence of uncertainty while securing targets against an observing adversary. To address this problem, we offer the following novel contributions in this paper: (i) New model of security games with defender teams that coordinate under uncertainty; (ii) New algorithm based on column generation that utilizes Decentralized Markov Decision Processes (Dec-MDPs) to generate defender strategies that incorporate uncertainty; (iii) New techniques to handle global events (when one or more agents may leave the system) during defender execution; (iv) Heuristics that help scale up in the number of targets and agents to handle real-world scenarios; (v) Exploration of the robustness of randomized pure strategies. The paper opens the door to a potentially new area combining computational game theory and multi-agent teamwork.[1]

Keywords — Game theory; Dec-MDP; Security; Stackelberg Games; Security Games

1 Introduction

Security games have recently emerged as an important research area in multi-agent systems, leading to successful deployments that aid security scheduling at ports, airports and other infrastructure sites, while also aiding in anti-poaching efforts and protection of fisheries [24, 45, 46, 49, 63, 65]. In this paper, when we refer to security games, we do not address the domain of computer security such as cybersecurity. The definition of a security game is a game where there are two players, a defender and an attacker. The players can be individuals or groups that cooperate to execute a strategy, where the leader (defender player) moves first while the follower (attacker player) observes the leader's strategy before moving (known as a Stackelberg game)[31]. The challenge addressed in security games is the optimization of the allocation of a defender's limited security agents (for example by determining randomized patrol routes or checkpoints). Such allocation is optimized taking into account the presence of an adversary who can conduct surveillance before planning an attack[12, 31, 42].

[1] An initial version of this work appeared in [51]. We extend this initial work with the following contributions with two new algorithms and extensive new analyses that improve our understanding of issues such as the relationship in security games of payoff covariance, graph structure, and execution uncertainty. More specifically: (i) In Section 4.2 we present a new heuristic to improve scale up to significantly larger defender teams than was possible in [51]; (ii) In Section 4.3 we propose and analyze a new approach that finds a locally optimal joint strategy; (iii) In Section 5.4 we provide additional analysis of the importance of addressing execution uncertainty; (iv) In Section 5.7.3 we further explore the relationship of deterministic versus randomized pure strategies under varying payoff structures - specifically we explore the relationship in the correlation between defender/attacker payoffs and performance of pure versus randomized defender strategies; (v) In Section 5.7.1 we evaluate the performance of the deterministic-based patrol strategy algorithm under varying graph structures and probabilities of delay to show the effect that graphs on the defender's expected utility. In addition to these contributions, three further new sections were added: Section 5.1 to discuss the metro rail domain, Section 6 for related work, and Section 7 which includes future work.

Unfortunately, previous work in security games has mostly ignored the challenge of defender teamwork; while the deployment of multiple defenders is optimized, most previous research has not focused on coordination among these agents (one exception is our previous work [50] which we build on and discuss in the Related Work section, Section 6.1). Additionally, no prior work has explored the effect of uncertainty in the coordination of multiple defender agents in security games.

This paper focuses on this challenge of computing an optimal agent allocation strategy for a defender team while also considering uncertainty in coordination of multiple defender agents. To that end, this paper combines two areas of research in multi-agent systems: security games and multi-agent teamwork under uncertainty. In many security environments, teamwork among multiple defender agents of possibly different types (e.g., joint coordinated patrols of aerial, motorized vehicles and canines) is important to the overall effectiveness of the defender. However, teamwork is complicated by the following three factors that we choose to address in this paper. First, multiple defenders may be required to coordinate their activities under uncertainty, e.g., delays that may arise from unexpected situations may lead different agents to miscoordinate, making them unable to act simultaneously. Second, some agents may leave the system unexpectedly requiring others to fill in the gaps that are created. Third, defenders may need to act without the ability to communicate, e.g., in security situations, communication may sometimes be intentionally switched off. We provide detailed motivating scenarios in Section 2 outlining these challenges.

To handle teamwork of defender agents in security games, our work makes the following contributions. First, this paper provides a new model of a security game where the defender team's strategy incorporates coordination under uncertainty. Second, we present a new algorithm that uses column generation and decentralized Markov Decision Problems (Dec-MDPs) to efficiently generate defender strategies in solving this new model of a security game. Third, global events among defender agents (e.g., a defender agent stops patrolling due to a bomb threat) are modeled in handling teamwork. Fourth, we contribute heuristics within our algorithm that help scale-up to real-world scenarios. Fifth, while exploring randomized pure strategies previously seen to converge faster, we discovered that they were not as fast but instead were more robust than deterministic pure strategies.

While the work presented in this paper applies to many of the application domains of security games, including the security of flights, ports and rail [56], we focus on the metro rail domain for a concrete example, given the increasing amount of rail related terrorism threats [47]. The challenges from interruptions, teamwork, or limited communication is not specific to only the metro rail domain and can be applied to other domains as well.

This paper is organized as follows: Section 2 starts with presenting the game theoretic model to address uncertainty among defender agents in a security game. Section 3 describes the algorithm to solve and compute the defender strategy. Section 4 presents heuristics to improve the runtime. Section 5 provides experimental results for all of our algorithms and heuristics. Section 6

explores the related work on security games and Dec-MDPs. Section 7 summarizes the contributions of the paper and future work.

2 Game Model of Patrolling Defender and Attacker Agent

This paper presents a game theoretic model of effective teamwork among multiple decentralized defender agents with execution uncertainty against an attacker agent. We are generalizing the security game model (background information on this model is in Section 5.2) to multiple defender agents coordinating under uncertainty. This section starts with preliminary background on Dec-MDPs (Section 2.1). The following section then gives an overview of the defender team and attacker model (Section 2.2). Next, the paper goes into detail of the defender's effectiveness at each target-time pair (Section 2.3). Then, the defender's pure strategy along with the attacker and defender's expected utility is discussed (Section 2.4). Finally, global events are explained and addressed in the model (Section 2.5).

2.1 Preliminary Knowledge on Dec-MDP

In this paper, we enhance security games by allowing complex defender strategies where multiple defenders coordinate under uncertainty. Attempting to find the optimal defender mixed strategy in such a setting is computationally extremely expensive, as discussed later. To speed up computation, we exploit advances in previous work in Decentralized Markov Decision Process (Dec-MDP) algorithms [14, 23, 54, 59], in one key component of our algorithm, and hence this section provides relevant background on Dec-MDPs.

Markov Decisions Processes (MDPs) are a useful framework to address problems that involve sequential decision making under uncertainty. In situations where there is only partial information of the system's state, a more general framework of Partially Observable Markov Decision Processes (POMDPs) are used. When there is a team of agents, where each one is able to make its own local observations, the framework is known as a Decentralized Markov Decision Process (Dec-MDP) when there is joint full observability (at a given time step, the total observation of all agents uniquely determine the state) [6, 7, 14, 15, 54], and a Decentralized Partially Observable Markov Decision Process (Dec-POMDP) when the agents together may not fully observe the state of the system and thus have uncertainty in their state [1, 2, 7, 40, 41, 62]. As we will explain later, when solving the security game model introduced in this paper, we use Dec-MDPs in one key component of our algorithm to attempt to optimize defender mixed strategies. Informally, in this component, we are faced with a problem involving multiple agents in a team, with uncertainty in their actions, and only local knowledge of states.

More specifically, we employ the transition independent Dec-MDP model [6] that is defined by the tuple: $\langle Ag, S, A, T, R \rangle$. $Ag = \{1, \ldots, n\}$ represents the set

of n agents [7]. $S = S_u \times S_1 \times \cdots \times S_n$ is a finite set of world states of the form $s = \langle s_u, s_1, \cdots, s_n \rangle$. Each agent i's local state s_i is a tuple (t_i, τ_i) where t_i is the target and τ_i is the time at which agent i reaches target t_i. Time is discretized (as explained in Section 5.1) and there are m decision epochs $\{1, \ldots, m\}$. s_u is the *unaffected state*, meaning that it is not affected by the agents' actions. It is employed to represent occurrences of global events (bomb threats, increased risk at a location, etc.) that are not dependent on the state or actions of the agents. This notion of unaffected states is equivalent to the one employed in Network Distributed POMDPs [37].

$A = A_1 \times \cdots \times A_n$ is a finite set of joint actions $a = \langle a_1, \cdots, a_n \rangle$, where A_i is the set of actions to be performed by agent i. $T : S \times A \times S \to \mathbb{R}$ is the transition function where $T(s, a, s')$ represents the probability of the next joint state being s' if the current joint state is s and the joint action is a. Since transitions between agent i's local states are independent of actions of other agents, we have transition independence [6]. Formally, $T(s, a, s') = T_u(s_u, s'_u) \cdot \prod_i T_i(\langle s_u, s_i \rangle, a_i, s'_i)$, where $T_i(\langle s_u, s_i \rangle, a_i, s'_i)$ is the transition function for agent i and $T_u(s_u, s'_u)$ is the unaffectable transition function. The joint reward function for the Dec-MDP takes the form of $R : S \to \mathbb{R}$, where $R(s)$ represents the reward for reaching joint state s.

Unfortunately, we cannot directly apply the Dec-MDP model to solve the security game that incorporates defender teamwork under uncertainty. One issue is that in the security game, the defender and attacker have different payoffs, which is not possible to be modeled in Dec-MDPs. Another issue is that we are modeling game-theoretic interactions, in which the rewards depend on the strategies of both the defender and the attacker. Therefore the standard Dec-MDP model cannot be directly applied to model and solve this game-theoretic interaction between the defender and attacker. Nevertheless, as mentioned earlier, to speed up the computation of the optimal defender mixed strategy under uncertainty, we decompose the problem into a game theoretic component and a Dec-MDP component (that only models the interaction among defender agents and does not need to model the interaction with the attacker nor have to consider the different payoffs for the attacker).

2.2 Defender and Attacker Model

The main differences in the model that is presented in this section compared to common security games are: the use of a target-time pair for the state of the defender, the effectiveness of a single defender agent along with the effectiveness of multiple agents at a target-time pair, and a joint policy as the defender's strategy. Common security game representations simply use a target and do not consider the time element. We need to incorporate the time element as there are multiple defender agents that must coordinate together to defend a target. In addition, common security game models represent a target as either covered or not covered by a defender, whereas we add an effectiveness value to show the varying levels of coverage based on the number of agents at a given state. Prior security game models do not use a joint policy for the defender's strategy as it

b	Target-time pair composed of (t, τ) where t is the target and τ is the time
$U_d^c(b)$	Defender payoff if b is covered by the defender (100% effectiveness)
$U_d^u(b)$	Defender payoff if b is uncovered by the defender (0% effectiveness)
$U_a^c(b)$	Attacker payoff if b is covered by the defender (100% effectiveness)
$U_a^u(b)$	Attacker payoff if b is uncovered by the defender (0% effectiveness)
R	Total number of agents
s_r	State of agent r, composed of a location(target) t, and time τ
ξ	Effectiveness of a single defender agent
$\texttt{eff}(s, b)$	Effectiveness of the agents on target-time pair b, given the global state s
$\boldsymbol{\pi}^j$	The the defender team's j^{th} pure strategy (joint policy)
J	Set of indices of defender pure strategies
P_b^j	The expected effectiveness of target-time pair b from defender pure strategy $\boldsymbol{\pi}^j$
$U_d(b, \boldsymbol{\pi}^j)$	Expected utility of the defender given a defender pure strategy $\boldsymbol{\pi}^j$, and an attacker pure strategy of target-time pair b
$U_a(b, \boldsymbol{\pi}^j)$	Expected utility of the attacker given a defender pure strategy $\boldsymbol{\pi}^j$, and an attacker pure strategy of target-time pair b
$\mathbf{x}$	Mixed strategy of the defender (probability distribution over $\boldsymbol{\pi}^j$)
$\mathbf{c}$	Vector of marginal coverages over target-time pairs
$U_d(b, \mathbf{c})$	Expected utility of the defender given marginal coverage $\mathbf{c}$, and an attacker pure strategy of target-time pair b

Table 1: Notation for game formulation

typically is represented as a set of targets that the defender agent must visit. We use a joint policy for the defender's strategy to model the defender agents' coordination under uncertainty.

The model for the defender team is represented by the tuple similar to the one for Dec-MDP as described in Section 2.1: $\langle Ag, S, A, T, U \rangle$. The main difference between this tuple and the one presented in Section 2.1 is the last element, U, which represents the utility or reward of the state. The reward is no longer just based on the state or action, as in traditional Dec-MDPs, but now is based on the interaction between the defender and attacker.

A (naive) patrol schedule for each agent consists of a sequence of commands; each command is of the form: at time τ, the agent should be at target t and execute action a. The action of the current command takes the defender agent to the location and time of the next command. In practice, each defender agent faces execution uncertainty, where taking an action might result in the defender agent being at a different location and time than intended. This type of execution uncertainty may arise due to unforeseen events. In our example metro rail domain, this uncertainty may arise due to questioning of suspicious individuals. The questioning of suspicious individuals results in the defender agent taking additional time to determine the motive and actions of the individuals, thereby taking a longer duration at the given location and potentially missing the next train and delaying the whole schedule.

The attacker is assumed to observe the defender's marginal coverage over the target-time pair (defined in detail later in this section). The defender's marginal coverage is based on the frequency and number of agents at each target-time pair. So in other words, the attacker cares about how often and with how many agents each target-time pair is visited by the defender team. The attacker's strategy is to choose which target and location to attack, and once that happens, the game terminates. For simplicity of exposition, we first focus on the case with no global events, in which case the unaffected state s_u never changes and can be ignored (we will consider these global events later in Section 2.5). Actions at s_r are decisions of which target to visit next. We consider the following model of delays that mirror the real-world scenarios of unexpected events: for each action a_r at s_r there are two states s'_r, s''_r with a nonzero transition probability: s'_r is the intended next state and s''_r has the same target as s_r but a later time. Next, we discuss the defender's effectiveness at each state and how this impacts defender coordination.

2.3 Defender Effectiveness

This section explains the value of the defender's effectiveness starting with a single defender agent and then how this changes with the inclusion of multiple defender agents. The defender's effectiveness of a single defender agent visiting a target-time pair is defined to be $\xi \in [0, 1]$. ξ can be less than 1 because visiting a target-time pair will not guarantee full protection. For example, if a defender agent visits a station while patrolling and walking through each of the platforms and the concourse, she will be able to provide some level of effectiveness, however she cannot guarantee that there is no adversary attack. Two or more defender agents visiting the same target-time pair provides an additional effectiveness. Given a global state s of defender agents, let $\texttt{eff}(s, b)$ be the effectiveness of the agents on target-time pair b. This effectiveness value, $\texttt{eff}(s, b)$, is similarly defined to be in the range $[0, 1]$ with 0 signifying no coverage and 1 representing full protection of the state b. We define the effectiveness of k agents visiting the same target-time pair to be $1 - (1 - \xi)^k$. This corresponds to the probability of catching the attacker if each agent independently has probability ξ of catching the attacker. Then

$$\texttt{eff}(s, b) = 1 - (1 - \xi)^{\sum_i I_{s_i = b}} \tag{1}$$

where $I_{s_i=b}$ is the indicator function that is 1 when $s_i = b$ and 0 otherwise. As more agents visit the same target-time pair, the effectiveness increases, up to the maximum value of 1. The rationale for the increase in effectiveness as additional agents visit the same target-time pair, b, is that as the attacker observes b, and notices multiple defender agents, this will provide further deterrence of the attacker choosing to target b. If the attacker observes just one defender agent, he can still choose to attack b, by first circumventing one defender agent. However if there are multiple defender agents, the attacker would either need additional help or decide to attack a different target-time pair. Although we

provide a function for the effectiveness value of $\texttt{eff}(s, b)$, our algorithm to solve this SSG would apply to other functions of effectiveness, including when different agents have different capabilities. The only constraint of other possible functions of the effectiveness given the global state s and target-time pair b, is that the value of the effectiveness is in the range $[0, 1]$. Other possibilities include representing defender agents that give an effectiveness value greater than 0 only when paired with another specialized type of defender agent. The next section explains the defender's pure strategy and the expected utility of both the defender and attacker.

2.4 Defender Pure Strategy and Expected Utility

This section first explains the model of the defender team's pure strategy and then describes how the defender and attacker's expected utility is computed based on the pure strategy, mixed strategy, and marginal coverage. Denote by $\boldsymbol{\pi}^j$ the defender team's j^{th} pure strategy (joint policy), and $\boldsymbol{\pi}^J$ the set of all defender pure strategies, where J is the corresponding set of indices. For example, if there are two defender agents, then a sample $\boldsymbol{\pi}^j$ includes a policy for defender agent 1 (r_1), and a policy for defender agent 2 (r_2). An example policy for r_1 is: $\{((t_1, 0)$:Visit $t_2), ((t_1, 1)$:Visit $t_2), ((t_2, 1)$:Visit $t_3)\}$, while an example policy for r_2 is: $\{((t_3, 0)$:Visit $t_2)$, $((t_3, 1)$:Visit $t_2), ((t_2, 1)$:Visit $t_1)\}$. The policy for r_1 is a mapping from the local state of r_1 to the corresponding action. If r_1 is at state $(t_1, 0)$, then the action that r_1 would take is to Visit t_2. However, if r_1 is at state $(t_2, 1)$, then she would choose action Visit t_3. Looking at the policy, r_1 starts at t_1 at time step 0, and tries to visit t_2 and then t_3, while defender agent r_2 starts at t_3 at time step 0, and traverses toward t_2 and then t_1. The global state s at time step 0, would be $\{(r_1 : (t_1, 0)), (r_2 : (t_3, 0))\}$, where r_1 is at t_1 and r_2 is at t_3.

Each pure strategy $\boldsymbol{\pi}^j$ induces a distribution over global states visited. Denote by $\Pr(s|\boldsymbol{\pi}^j)$ the probability that global state s is reached given $\boldsymbol{\pi}^j$. The expected effectiveness of target-time pair b from defender pure strategy $\boldsymbol{\pi}^j$, is denoted by P_b^j; formally,

$$P_b^j = \sum_s \Pr(s|\boldsymbol{\pi}^j)\texttt{eff}(s, b) \tag{2}$$

Given a defender pure strategy $\boldsymbol{\pi}^j$, and an attacker pure strategy of target-time pair b, the expected utility of the defender is

$$U_d(b, \boldsymbol{\pi}^j) = P_b^j U_d^c(b) + (1 - P_b^j)U_d^u(b) \tag{3}$$

The attacker's utility is defined similarly as:

$$U_a(b, \boldsymbol{\pi}^j) = P_b^j U_a^c(b) + (1 - P_b^j)U_a^u(b) \tag{4}$$

The defender may also play a mixed strategy $\mathbf{x}$, which is a probability distribution over the set of pure strategies $\boldsymbol{\pi}^J$. Denote by x_j the probability of playing

pure strategy $\boldsymbol{\pi}^j$. Simply choosing a single defender pure strategy, $\boldsymbol{\pi}^j$, or a single joint policy, is typically not the defender's optimal strategy due to the various constraints that limit the coverage over all the target-time pairs. For example, a single defender pure strategy may only allow the defender team to visit half of the possible target-time pairs. In this example, if the defender decides to select a single pure strategy to execute, then the attacker would decide to attack one of the target-time pairs that is not covered by the defender. Therefore, in this situation, a mixed strategy for the defender that covers all possible target-time pairs provides a better strategy for the defender. The players' expected utilities given mixed strategies are then naturally defined as the expectation of their pure-strategy expected utilities. Formally, the defender's expected utility given the defender mixed strategy $\mathbf{x}$ and attacker pure strategy b is $\sum_j x_j U_d(b, \boldsymbol{\pi}^j)$. Let

$$c_b = \sum_j x_j P_b^j \tag{5}$$

be the *marginal* coverage on b by the mixed strategy $\mathbf{x}$ [66], and $\mathbf{c}$ the vector of marginal coverages over target-time pairs. Then this expected utility can be expressed in terms of marginal coverages, as

$$U_d(b, \mathbf{c}) = c_b U_d^c(b) + (1 - c_b) U_d^u(b) \tag{6}$$

The model above assumes no global events, or when the unaffected state s_u never changes. In the following section, we introduce global events and how it impacts the model.

2.5 Global Events

A global event refers to some event whose occurrence becomes known to all agents in the team, and causes one of the agents in the defender team to become unavailable, causing others to fill in the gaps created. In our example domain, global events correspond to scenarios such as bomb threats or crime, where a agent must stop patrolling and deal with the unexpected event. The entire defender team is notified when a global event occurs. Depending on the type of event, a pre-specified defender agent, which we denote as the qualified defender agent, will be removed from patrolling and allocated to deal with the event once it occurs. This is because certain defender agents have capabilities best suited towards addressing the global event, thereby having the pre-specified, qualified defender agent stop patrolling and handle the global event while the other defender agents continue to monitor and patrol.

To handle such global events, we include the global unaffected state in our security game model. The global unaffected state is a vector of binary variables over different types of events that may be updated at each time step τ. This state is labeled as such because it is known by each defender agent but is not affected by the defender team; the defender team has no control over this global unaffected state. For example, a global state could be a vector $< 1, 0, 1 >$ where

each element corresponds to the type of the event such as bomb threat, active shooter, or crime. If the first element corresponds to a bomb threat and is set to 1, that implies that a bomb threat has been received.

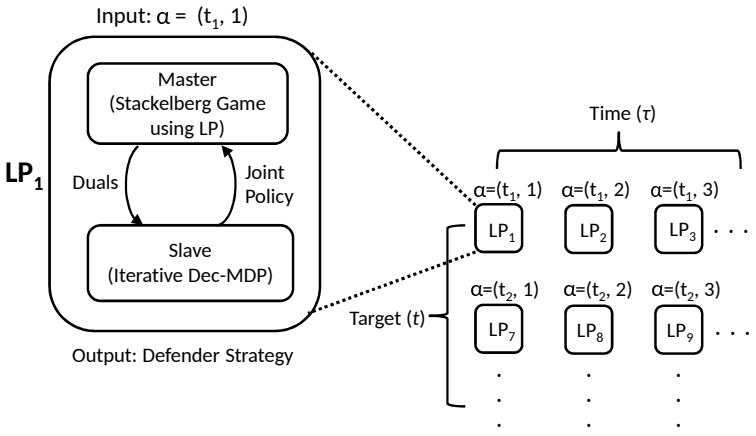
When the global unaffected state is updated (a global event occurs), this results in a change in the state for both the qualified defender agent as well as the other defender agents. The qualified defender agent stops patrolling to address the global event while the remaining defender agents may change their strategy and subsequent actions to account for the qualified defender agent leaving the system. Transitions associated with global unaffected state, i.e., $T_u(s_u, s'_u)$ could potentially be computed based on the threat/risk levels of various events at the different time steps. The transitions associated with individual defender agents, i.e., $T_i(\langle s_u, s_i \rangle, a_i, s'_i)$ are dependent on whether the defender agent is responsible for handling a global event that has become active in that time step. If s_u indicates that a bomb threat is active and i is the qualified defender agent, then the valid joint policy indicates that the qualified defender agent handles the global event and goes out of patrolling duty. If s_u indicates a bomb threat and i is not the qualified defender agent, then agent i would choose an action a_i based on s_u with the knowledge that the qualified defender agent is no longer patrolling.

Problem Statement: Our goal is to compute the *strong Stackelberg equilibrium* of the new game representation that includes joint policies as defined earlier as the pure strategies for the defender. In other words, we want to find the optimal (highest expected value) mixed strategy for the defender to commit to considering that a strategic adversary best responds to her strategy.

3 Approach to Solve Multiple Linear Programs and Iterative Dec-MDP

This section begins with a linear program (LP) to solve for the defender's optimal strategy based on the game model discussed in the previous section (Section 2). Given the exponential number of defender pure strategies (joint policies) that are needed to solve the LP, we introduce a column generation framework [4] to intelligently generate a subset of pure strategies for the defender. The space of joint policies is very large. We look to Dec-MDP algorithms to cleverly search that space [6, 14, 43, 54] as Dec-MDPs are used by researchers to coordinate multiple agents when there is uncertainty in the system. This fits well in helping to find a pure strategy for the defender agents in handling uncertainty. However, optimal Dec-MDP algorithms are difficult to scale-up, and hence we use heuristics that leverage ideas from previous work on Dec-MDPs [59]. We need to solve multiple Dec-MDP instances as each computed joint policy is used as a single pure strategy for the defender. The use of heuristics results in the possibility that our algorithm does not find the optimal defender mixed strategy. However, we show in the experimental results that the heuristic solution is able to scale-up and perform better than algorithms that do not handle uncertainty (which

can scale-up but suffer from solution quality loss) in Section 5.4 or algorithms that attempt to find the optimal solution (which may not suffer from solution quality loss but cannot scale up) in Section 5.5 or algorithms that attempt to find even higher quality solutions heuristically (they still fail to perform better) in Section 5.6.2.



24

Figure 1: Diagram of the System

Figure 1 gives a high level view of the system as a whole. The right half of the diagrams shows that for each possible attacker choice (a target-time pair) we solve a separate LP. For each LP, a column generation approach using a master and slave component (shown on the left side of the diagram) is used to find the defender strategy given the attacker's choice. The master component is solved by finding the optimal defender strategy of the Stackelberg game given the set of defender joint policies generated by the slave component. The slave component computes the joint policy by solving an iterative Dec-MDP. Each part of the system is explored in depth in the rest of this section.

A standard method for solving Stackelberg games is the Multiple-LP algorithm [12]. The Multiple-LP approach involves iterating over all attacker choices. The attacker has $|B|$ choices and hence we iterate over these choices. In each iteration, we assume that the attacker's best response is fixed to a pure strategy α, which is a target-time pair, $\alpha = (t, \tau)$.

$$\max_{\mathbf{c},\mathbf{x}} \quad U_d(\alpha, \mathbf{c}) \tag{7}$$

$$U_a(\alpha, \mathbf{c}) \geq \quad U_a(b, \mathbf{c}) \quad \forall b \neq \alpha \tag{8}$$

$$c_b - \sum\nolimits_{j \in J} P_b^j x_j \leq \quad 0 \quad \forall b \in B \tag{9}$$

$$\sum\nolimits_{j \in J} x_j = \quad 1 \tag{10}$$

$$x_j \geq 0 \quad \forall j \in J, \quad c_b \in [0,1] \quad \forall b \in B \tag{11}$$

The LP for α, shown in Equations (7) to (11), solves the optimal defender mixed strategy $\mathbf{x}$ to commit to, given that the attacker's best response is to attack α. Then among the $|B|$ solutions, the solution that achieves the best objective (i.e., defender expected utility) is chosen. In more detail, Equation (8) enforces that the best response of the attacker is indeed α. In Equation (9), $\mathbf{P}^j$ is a column vector which gives the values of expected effectiveness P_b^j of each target-time pair b given the defender's pure strategy $\boldsymbol{\pi}^j$. An example of a set of column vectors is shown below:

$$\mathbf{P} = \begin{array}{c} \\ b_1 \\ b_2 \\ b_3 \\ b_4 \end{array} \begin{array}{c} \begin{array}{ccc} j_1 & j_2 & j_3 \end{array} \\ \begin{bmatrix} 0.0 & 0.5 & 0.4 \\ 0.2 & 0.7 & 0.0 \\ 0.5 & 0.6 & 0.2 \\ 0.6 & 0.0 & 0.8 \end{bmatrix} \end{array} \tag{12}$$

Column $\mathbf{P}^{j_1} = \langle 0.0, 0.2, 0.5, 0.6 \rangle$ gives the effectiveness $P_{b_i}^{j_1}$ of the defender's pure strategy $\boldsymbol{\pi}^{j_1}$ over each target-time pair b_i. For example, policy $\boldsymbol{\pi}^{j_1}$ has an effectiveness of 0.5 on b_3. Thus, Equation (9) enforces that given the probabilities x_j of executing mixed strategies π^j, c_b is the marginal coverage of b.

Figure 2 gives a diagram of how the Multiple-LP algorithm applies to our solution approach. Focus first on the right side of Figure 2. There the figures show several LPs. In particular, this approach generates a separate LP for each attacker pure strategy denoted as α in Equations (7) to (11). For example, the first LP that is solved, assumes that the attacker's best strategy, α is to attack target t_1 at time $\tau = 1$. The algorithm fixes the attacker's best strategy, $\alpha = (t_1, 1)$, and then solves for the defender team's strategy under the constraint that the attacker's best response is α. The algorithm then iterates to the next LP, which corresponds to a new attacker strategy. Once all the LPs have been solved, we compare the defender's strategy for each attacker strategy/LP and choose the one that gives the defender the highest expected utility.

For each LP that is being solved, the input is the attacker's best strategy, denoted as α, which is composed of a target and time. The output of each LP is the defender's strategy against an attacker whose best strategy is α. To determine the defender's strategy against the attacker, all the defender pure strategies must be enumerated. However, in our game there is an exponential number of possible defender pure strategies, corresponding to joint policies —

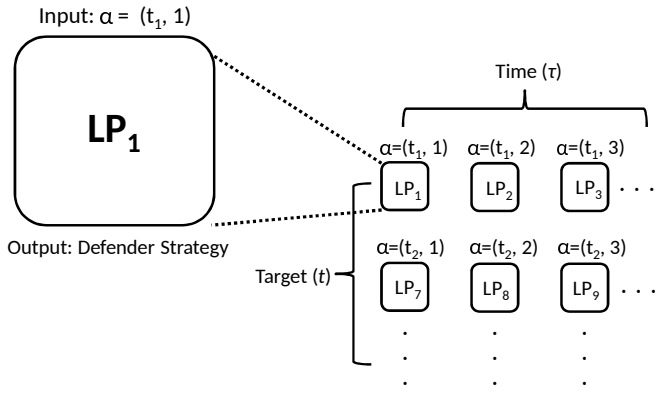


Figure 2: Diagram of the Multiple-LP approach

and thus a massive number of columns that cannot be enumerated in memory — so that the Multiple-LP algorithm cannot be directly applied. For N stations, T time steps, and R defender agents, we will have $(N^T)^R$ policies.

Since this grows exponentially large in proportion to the number of stations, time steps, and defender agents, we turn to column generation to solve the LP and intelligently compute a subset of defender pure strategies along with the optimal defender mixed strategy. We solve an LP using a column generation framework for each possible target-time pair for the attacker strategy and then choose the solution that achieves the highest defender expected utility. The column generation framework is composed of two components, the master and slave. The master component solves the LP given a subset of defender pure strategies (or joint policies). The slave component computes the next best defender pure strategy or joint policy to improve the solution found by the master component. We cast the slave problem as a Dec-MDP to generate the joint policy for the defender team. In the next section, we explore in detail the column generation framework.

3.1 Column Generation

The defender needs to know all possible pure strategies in order to compute the optimal strategy against the attacker. However, as stated in the previous section, the number of possible defender pure strategies grows exponentially

in the number of stations, time steps, and defender agents. To deal with this problem, we apply column generation [4], a method for efficiently solving LPs with large numbers of columns. At a high level, it is an iterative algorithm composed of a master and a slave component; at each iteration the master solves a version of the LP with a subset of columns, and the slave smartly generates a new column (defender pure strategy) to add to the master.

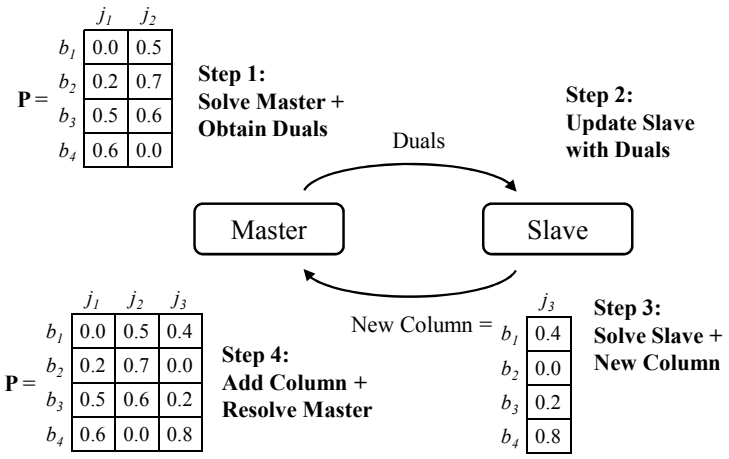


Figure 3: Column generation illustration including the master and slave components. The column generation algorithm contains multiple iterations of the master-slave formulation.

Figure 3 gives an example that shows the master-slave column generation algorithm. Note that there are four steps in this figure to explain the process and interaction between the master and slave component. In the first step, the master component solves an LP to generate a defender mixed strategy while also computing the corresponding dual variables (Step 1). The master starts with a subset of defender pure strategies represented as columns in $\mathbf{P}$. In this example, the master is solving the LP given two columns, j_1 and j_2. The dual values from the master component are then used as input for the slave component (Step 2).

Then the slave component computes a defender pure strategy (joint policy) and returns the column that corresponds to the defender pure strategy back to the master component (Step 3). We show in this example that the column j_3 is generated by the slave component. The master component then adds this new column to the existing set of columns, $\mathbf{P}$, and then resolves the LP which now includes the new column generated from the slave (Step 4). We see here that

now the master resolves the LP but with three columns now, j_1 to j_3. This master-slave cycle is repeated for multiple iterations until the column generated by the slave no longer improves the strategy for the defender. Next, we go in detail about first the master component and then the slave component.

The master is an LP of the same form as Equations (7) to (11), except that instead of having all pure strategies, J is now a subset of pure strategies. Pure strategies not in J are assumed to be played with zero probability, and their corresponding columns do not need to be represented. We solve the LP and obtain its optimal dual solution.

The slave's objective is to generate a defender pure strategy π^j and add the corresponding column $\mathbf{P}^j$, which specifies the marginal coverages, to the master. We show that the problem of generating a good pure strategy can be reduced to a Dec-MDP problem.

To start, consider the question of whether adding a given pure strategy π^j will improve the master LP solution. This can be answered using the concept of the *reduced cost* of a column [4], which intuitively gives the potential change in the master's objective when a candidate pure strategy π^j is added. Formally, the reduced cost $\overline{f}_j$ associated with the column $\mathbf{P}^j$ is defined as:

$$\overline{f}_j = \sum\nolimits_b y_b \cdot P_b^j - z \tag{13}$$

where z is the dual variable of (10) and $\{y_b\}$ are the dual variables of Equation family (9), and are calculated using standard techniques. If $\overline{f}_j > 0$ then adding pure strategy π^j will improve the master LP solution. When $\overline{f}_j \leq 0$ for all j, the current master LP solution is optimal for the full LP.

Thus the slave computes the π^j that maximizes $\overline{f}_j$, and adds the corresponding column to the master if $\overline{f}_j > 0$. If $\overline{f}_j \leq 0$ the algorithm terminates and returns the current master LP solution.

3.2 Dec-MDP Formulation of Slave

We formulate this problem of finding the pure strategy that maximizes reduced cost as a transition independent Dec-MDP [6]. The rewards are defined so that the total expected reward is equal to the reduced cost. The states and actions are defined as before. We can visualize them using *transition graphs*: for each agent r, the transition graph $\mathcal{G}_r = (N'_r, E'_r)$ contains state nodes $s_r = (t, \tau) \in S_r$ for each target and time. In addition, the transition graph also contains action nodes that correspond to the actions that can be performed at each state s_r. There exists a single action edge between a state node s_r and each of the action nodes that correspond to the possible actions that can be executed at s_r. From each action node a_r from s_r, there are multiple outgoing chance edges, to state nodes, with the probability $T_r(s_r, a_r, s'_r)$ labeled on the chance edge to s'_r. In the illustrative example scenario that we have focused on, with there being delays, each action node has two outgoing chance edges with one chance edge going to the intended next state and another chance edge going to a different state which has the same location as the original node but a later time.

Example: Figure 4 shows a sample transition graph showing a subset of the states and actions for agent i. Looking at the state node $(t_1, 0)$, assuming target t_1 is adjacent to t_2 and t_5, there are three actions, Stay at t_1,Visit t_2, or Visit t_5. If action, Visit t_2 is chosen, then the transition probability is: $T_i((t_1, 0), \text{Visit } t_2, (t_2, 1)) = 0.9$ and $T_i((t_1, 0), \text{Visit } t_2, (t_1, 1)) = 0.1$.

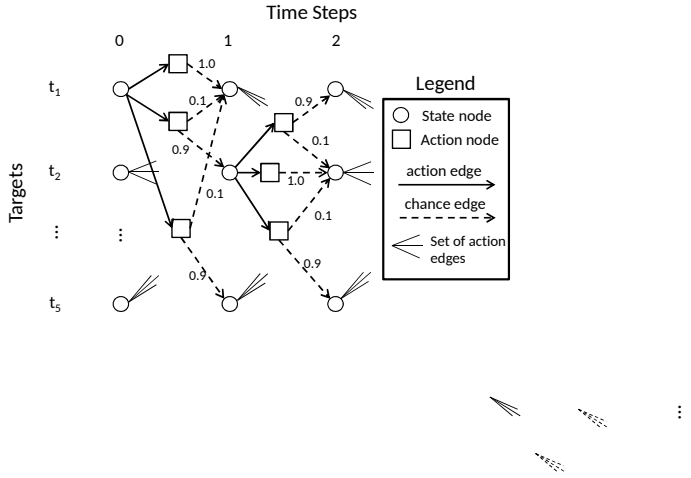


Figure 4: Example Transition Graph for one defender agent

The transition independent Dec-MDP consists of multiple such transition graphs, which we represent as $\mathcal{G}_r$. There is however a joint reward function $R(s)$. This joint reward function, $R(s)$, is dependent on the dual variables, y_b, from the master, and the effectiveness $\texttt{eff}(s, b)$ of agents with global state s on target-time pair b, as defined in Section 2:

$$R(s) = \sum\nolimits_b y_b \cdot \texttt{eff}(s, b). \tag{14}$$

Multiple transition graphs are needed because each defender agent may have a different graph structure and/or action space.

We provide an example for the joint reward function $R(s)$, continuing from the scenario described in Section 2.4. The example global state is $s_i = \{(r_1 : (t_1, 0)), (r_2 : (t_3, 0))\}$, where r_1 is at t_1 and r_2 is at t_3. Since there are only two target-time pairs in this global state, we only need to sum over these two pairs because for all other pairs, the effectiveness, $\texttt{eff}(s, b) = 0$. If we define $\xi = 0.6$, the defender's effectiveness of a single agent visiting a target-time pair, $b_1 = (t_1, 0)$, and $b_2 = (t_3, 0)$ then:

$$R(s) = \sum\nolimits_b y_b \cdot \texttt{eff}(s,b) = y_{b_1} \cdot 0.6 + y_{b_2} \cdot 0.6 \tag{15}$$

Proposition 3.1. *Let π^j be the optimal solution of the slave Dec-MDP with reward function defined as in* (14)*. Then π^j maximizes the reduced cost $\overline{f}_j$ among all pure strategies.*

Proof. The expected reward of the slave Dec-MDP given π^j is

$$\sum\nolimits_s \Pr(s|\pi^j)R(s) = \sum\nolimits_b y_b \sum\nolimits_s Pr(s|\pi^j)\texttt{eff}(s,b) \tag{16}$$

$$= \sum\nolimits_b y_b P_b^j = \overline{f}_j + z. \tag{17}$$

Therefore the optimal policy for the Dec-MDP maximizes $\overline{f}_j$. □

3.3 Solving the Slave Dec-MDP

If the Dec-MDP is solved optimally each time it is called in the master-slave iteration, we would achieve the optimal solution of the LP. Unfortunately, optimally solving Dec-MDPs, particularly given large numbers of states (target-time pairs) is extremely difficult. The optimal algorithms from the MADP toolbox[55] along with the MPS algorithm [14] are unable to scale up past four targets and four agents in this problem scenario. Experimental results illustrating this outcome are shown in Section 5. Hence this section focuses on a heuristic approach. As mentioned earlier, this implies that we do not guarantee achieving the optimal value of each LP we solve; however, we do show in Section 5 that this approach scales better than one attempting to achieve the optimal and one that scales but does not handle uncertainty.

Our approach, outlined in Algorithm 1, borrows some ideas from the TREMOR algorithm [59], which iteratively and greedily updates the reward function for the individual agents and solves the corresponding MDP. We do not use the TREMOR algorithm but reference this algorithm as the closest algorithm in the Dec-MDP literature to the one implemented in this section. In particular, unlike TREMOR, there is no iterative process in our algorithm. More specifically, for each agent r, this algorithm updates the reward function for the MDP corresponding to r and solves the single-agent MDP; the rewards of the MDP are updated so as to reflect the fixed policies of previous agents.

The MDP for each agent consists of: S_r, the set of local states s_r in the form of a tuple (t, τ); A_r, the set of actions that can be performed by the agent; $T(s_r, a_r, s'_r)$, the transition function of the agent at state s_r taking the action a_r and ending up at state s'_r; and $R(s_r)$, the reward function which represents the reward for visiting and covering state s_r. The value of the reward is determined both by the dual variable y_b, from the master and the policies of defender agents that have already been computed from previous iterations.

Algorithm 1 SolveSlave(y_b, $\boldsymbol{\mathcal{G}}$)

1: Initialize π^j
2: **for all** $r \in R$ **do**
3: $\quad \boldsymbol{\mu}_r \leftarrow \text{ComputeUpdatedReward}(\pi^j, y_b, \mathcal{G}_r)$
4: $\quad \pi_r \leftarrow \text{SolveSingleMDP}(\boldsymbol{\mu}_r, \mathcal{G}_r)$
5: $\quad \pi^j \leftarrow \pi^j \cup \pi_r$
6: $\mathbf{P}^j \leftarrow \text{ConvertToColumn}(\pi^j)$
7: return $\pi^j, \mathbf{P}^j$

In more detail, this algorithm takes the dual variables y_b (refer Section 3.1) from the master component and $\boldsymbol{\mathcal{G}}$ as input and builds π^j iteratively in Lines 2–5. Line 3 computes vector $\boldsymbol{\mu}_r$, the *additional* reward of reaching each of agent r's states.

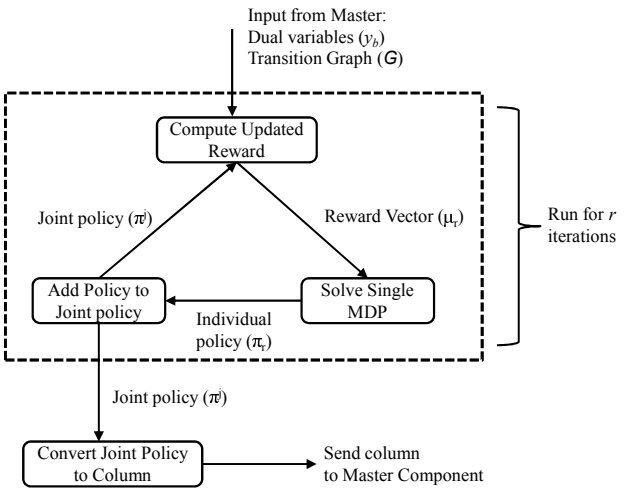


Figure 5: Diagram of the algorithm for the slave component

Figure 5 gives a diagram of how the slave component operates. It receives as input from the master component the dual variables y_b and the transition graph $\boldsymbol{\mathcal{G}}$. It then solves and generates an individual policy, π_r, for each agent, based on the reward vector. This reward vector takes into account the dual variables from the master along with the individual policies of agents that have already been computed. After all individual policies have been generated, the joint policy is converted into a column and then sent to the master.

Consider the slave Dec-MDP defined on agents $1, \ldots, r$ (with joint reward function (14)). The additional reward $\mu_r(s_r)$ for state s_r is the marginal contri-

bution of r visiting s_r to this joint reward, given the policies of the $r-1$ agents computed in previous iterations, $\pi^j = \{\pi_1, \ldots, \pi_{r-1}\}$. Specifically, because of transition independence, given $\{\pi_1, \ldots, \pi_{r-1}\}$ we can compute the probability $p_{s_r}(k)$ that k of the first $r-1$ agents have visited the same target and time as s_r. Then $\mu_r(s_r) = \sum_{k=0}^{r-1} p_{s_r}(k)(\texttt{eff}(k+1) - \texttt{eff}(k))$, where we slightly abuse notation and define $\texttt{eff}(k) = 1-(1-\xi)^k$. $\mu_r(s_r)$ gives the additional effectiveness if agent r visits state s_r by computing the effectiveness of agent r visiting state s_r (incorporating the policies of the agents that have already been computed) and subtracting the effectiveness due to just the previous agents and not agent r. For example, if two previously computed agents already visit a state s_r, then if the third agent visits state s_r, the individual reward for the third agent will not be the joint reward of having three agent visit the state, but will instead be the additional effectiveness of having three agents visit the state versus two agents. This avoids double-counting for states that have been visit by other previously computed agents.

Line 4 computes the best individual policy π_r for agent r's MDP, with rewards $\boldsymbol{\mu}_r$. We compute π_r using value iteration (VI):

$$V(s_r, a_r) = \mu_r(s_r) + \sum\nolimits_{s'_r} T_r(s_r, a_r, s'_r) V(s'_r) \tag{18}$$

where $V(s_r) = \max_{a_r} V(s_r, a_r)$ and $\pi_r(s_r) = \arg\max_{a_r} V(s_r, a_r)$.

The way that the Dec-MDP value function is decomposed into the individual MDP value function is that for each MDP for an agent, the rewards are updated/precomputed based on the policies of prior agents that have already been computed. For the first agent, the value function on each state for the MDP would simply be the reward if there is just one agent. This agent then solves the MDP to generate an individual policy. For the second agent, the value function now gets updated based on the individual policy of the first agent. More specifically, the value function for the second agent gets updated by modifying the rewards ($\mu_r(s_r)$) on the states that the first agent visits, to reflect the additional reward/effectiveness that the defender team would receive if a second agent visits that same state versus having just a single agent visit that state. In particular, the reward vector, $\boldsymbol{\mu}_r$ is being changed in the value function for the different agents (in Line 3).

4 Heuristics for Scaling Up

Without column generation, our model of Dec-MDPs in security games would be faced with enumerating $(N^T)^R$ columns, making enumeration of defender pure strategies impossible, let alone trying to find a solution. While column generation is helpful, each LP still does not scale well and thus in this section, we present three different approaches to further improving the runtime. We first started by examining what component in the algorithm was consuming the majority of the time needed to find the defender's strategy. The slave component within the column generation was found to be taking significantly more time

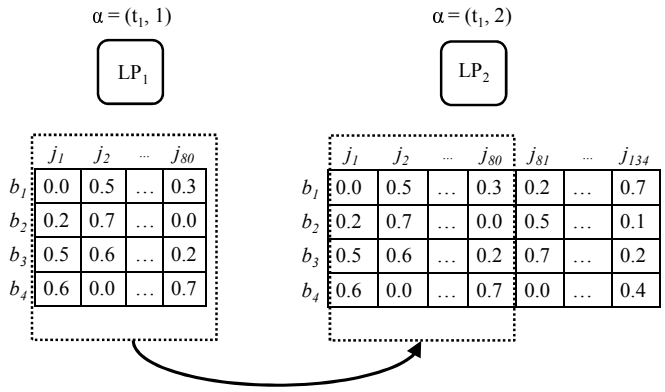
than the master component. When running the algorithm with 8 targets, 8 time steps, and 8 agents, the master component took an average of 7.2 milliseconds while the slave component took an average of 26.3 milliseconds. Increasing the number of agents from 8 to 12 resulted in the master component taking an average of 7.3 milliseconds and the slave component taking an average of 101.3 milliseconds. Further increasing the number of agents from 12 to 16, the master component took on average 7.5 milliseconds while the slave component took on average 1,229.8 milliseconds. Thus, as the number of agents increased, the master component did not increase in runtime while the runtime for the slave component increased exponentially from 26.3 milliseconds to 101.3 milliseconds, and then to 1,229.3 milliseconds. This demonstrates that the slave component is clearly a bottleneck.

As discussed in Section 3.1, the column generation approach requires multiple master-slave iterations, and thus there are three different approaches that could be used to attempt to improve the runtime of the column generation process by focusing on the slave component. First, we focus on reducing the number of iterations that the column generation algorithm needs to execute, thereby reducing the number of times the slave component is called in Section 4.1. Second, we then concentrate on decreasing the runtime of a single slave iteration (which we find to take significantly more time than the master component) to aid in scaling up to more defender agents in Section 4.2. The third approach that was considered to improve the runtime of the algorithm was the idea of computing a higher quality solution for the slave component so that the number of total iterations needed by column generation would be reduced (Section 4.3).

4.1 Reducing the Number of Column Generation Iterations

The initial approach starts with each LP computing its own columns (i.e., cold-start). However, this does not scale well and thus we build on this approach with several heuristics for scale-up that focuses on reducing the amount of times column generation needs to be executed:

Append: First, we explored reusing the generated defender pure strategies and columns across the multiple LPs. The intuition is that the defender strategies/columns generated by the master-slave column generation algorithm for an LP might be useful in solving subsequent LPs, resulting in an overall decrease in the total number of defender pure strategies/columns generated (along with fewer iterations of column generation) over all the multiple LPs. Figure 6 gives an example of how the Append heuristic shares the columns across different LPs. This figure shows two of the multiple LPs that need to be solved (refer to Figure 2 for the diagram of the Multiple-LP approach). In this example, in the first LP, the column generation approach outputs 80 columns or defender pure strategies in determining the defender's strategy, when the attacker's optimal strategy is to attack target-time pair $(t_1, 1)$. Then the second LP, where the attacker's optimal strategy is set to $(t_1, 2)$ is solved. The 80 columns that were generated to solve the first LP are then carried over to be used in the second



25

Figure 6: Example of the Append heuristic

LP (as denoted by the dashed line box). To extend the example shown in this figure, all 134 columns that are used in the second LP will then be carried over to the third LP. This continues for all subsequent LPs.

Cutoff: To further improve the runtime, we explored setting a limit on the number of defender pure strategies generated (i.e., the number of iterations of column generation that is executed) for each LP.

Ordered: With this limit on the columns generated, some of the $|B|$ LPs return low-quality solutions, or are even infeasible, due to not having enough columns. Combined with reusing columns across LPs, the LPs that are solved earlier will have fewer columns. Since we only need a high-quality solution for the LP with the best objective, we would like to solve the most promising LPs last, so that these LPs will have a larger set of defender pure strategies to use. While we do not know apriori which LP has the highest value, one heuristic that turns out to work well in practice is to sort the LPs in increasing order of $U_a^u(b)$, the uncovered payoff of the attacker strategies (target-time pairs) chosen; i.e., to solve the LPs that correspond to attack strategies that are less attractive to the attacker first, and LPs (attack strategies) that are more attractive to the attacker later.

4.2 Reducing Runtime for a Single Slave Iteration

The heuristics in Section 4.1 target reducing the total number of iterations, but not the run-time within a single slave iteration. Here, we focus on reducing the runtime of a single iteration which helps to scale up as the number of agents increases. The importance of scaling up to handle defender teams that are comprised of multiple agents is demonstrated in a large scale real-world experiment of security games that had to plan for 23 defender security teams [18].

To deal with the inability of the previous heuristics in Section 4.1 to handle many defender agents, we explored the following desiderata to guide our selection of an idea to allow us to scale up: (1) The idea has to focus on the part of the entire algorithm that actually causes a slowdown. (2) If we introduce a heuristic, the slave should report the column truthfully to the master. If the slave does not report the column truthfully, then the master will compute a solution that is inaccurate for the LP (in the Multiple-LP approach). If the solution/value for the LP is incorrect, then we may end up selecting the best LP incorrectly and choose a low valued strategy. (3) The heuristic itself should be very simple. The master calls the slave multiple times within any given problem instance, and it is important that the slave generate a column in a timely fashion. (4) The heuristic should preferably lead the slave to be conservative, i.e., it is preferred if the heuristic does not place fewer agents on important targets.

The rationale for why the slave component was taking a long time to run, was the exponential increase due to two factors: (1) the size of the state space, when the number of agents increases, and (2) the computation of the updated rewards that is needed to determine the effectiveness at each state based on the defender's joint policy (Algorithm 1, Line 3). For example, if there are 16 defender agents and each agent has a non-zero probability of visiting state s, then the computation of the updated reward would require iterating through all subsets of the 16 defender agents, or $\binom{16}{1} + \binom{16}{2} + \cdots + \binom{16}{16} = 65,535$ possible combinations of defender agents.

Algorithm 2 ComputeEffectiveness(π, b)

```
1: Initialize w
2: R_s ← FindResourcesAtState(π, b)
3: for n = 1 ... |R_s| do
4:    C ← CombinationGenerator(R, n)
5:    for all c ∈ C do
6:       p ← ComputeEffectInstance(c, π, b)
7:       w ← w + p
8: return w
```

To improve the runtime to handle a larger number of agents, we used the desiderata as a guideline. We explored setting a limit on the number of agents in the computation of the effectiveness of a given state, $\texttt{eff}(s, b)$, but do not actually place a limit in the game and in the column that is computed by the slave component and used by the master component. The reasoning to place a

limit on the number of agents is that the effectiveness for the defender does not significantly increase when there are already a few defender agents at a state. For example, if a state is already covered by ten defender agents, adding an additional defender agent will not provide a significant increase in effectiveness, compared to the additional benefit if there was just one defender agent and another agent was added. Algorithm 2 gives the algorithm of computing the effectiveness of joint policy π on state b. Algorithm 2 is used in Algorithm 1, for the computation of the updated rewards (Algorithm 1, Line 3) and in transforming the policy that encompasses all agents into a column for the master (Algorithm 1, Line 6). In both cases, we need to enumerate all combinations of agents for each state to compute the effectiveness of the defender agents at each state. The computation of the updated rewards (Algorithm 1, Line 3) is used more expansively in the slave component compared to the conversion of the policy to a column (Algorithm 1, Line 3) and thus we focus on improving the runtime and computation of the updated rewards. Since this updated computation of the effectiveness can potentially generate a lower effectiveness value (as described in detail below), by not modifying the computation of the policy to a column, the algorithm still provides an accurate column for the master component. By placing a limit on the maximum number of agents at any given state, the solution quality may decrease because the resulting joint policy computed by the slave does not consider the increased effectiveness of additional agents above the imposed limit, but at the end of the slave calculation (Algorithm 1, Line 6) the column return to the master accurately describes the effectiveness of the joint policy.

Algorithm 2 starts by computing R_s, which is the set of agents that have a non-zero probability of visiting state b (Line 2), by scanning through the policy of each agent to see if there is a possibility of reaching state b. It then iterates from 1 to the total number of agents that have a non-zero probability, or $|R_s|$, of visiting state b. This value of n, represents the number of agents that visit state b, where the algorithm computes the probability and corresponding effectiveness. In Line 4, the algorithm generates all possible combinations of agents of size n. For example, if $R = 5$ and $n = 2$, then $C = \{(1,2),(1,3),(1,4),(1,5),(2,3),(2,4),(2,5),(3,4),(3,5),(4,5)\}$, where the numbers in each set correspond to different agents. For each combination, the effect of each particular combination is computed and added together (Lines 6-7). For example, if $c = (1,4)$, then ComputeEffectInstance(c, π, b) (Line 6) would compute the effectiveness of two agents at state b, multiplied by the probability of agent 1 and 4 at state b, along with the probability of all other agents not being at state b.

During this computation of the effectiveness of joint policy π on state b, instead of computing the effectiveness by allowing up to $|R_s|$ agents, we place a limit on the maximum number of agents (set to z) that can be at state b (just in our calculation of the updated rewards but not while converting the policy to a column). To accomplish this, Algorithm 2 is modified at Line 3 so instead of n iterating from 1 to $|R_s|$, it will instead iterate from 1 to z.

This simplifies the computation of the effectiveness, $\texttt{eff}(s, b)$, for all states

and in turn improves the runtime of the slave. This is because the algorithm does not need to compute all combinations of agents from lines 3 to 7, which grows exponentially large as the number of agents increases. By placing a limit of at most z agents to consider while calculating the effectiveness, we are able to improve the runtime and scale up to a larger number of agents. Despite this limit in calculating the effectiveness, in reality more than z agents may visit this state.

However, when converting the defender's joint policy to a column (Algorithm 1, Line 6), we can compute the exact effectiveness, $\texttt{eff}(s, b)$, by calling Algorithm 2 without placing a limit on the maximum number of agent. In Algorithm 2, Line 3, instead of just iterating from 1 to z, the algorithm iterates from 1 to $|R_s|$ to compute an exact effectiveness of the policy for the column that is returned to the master component. In other words, we speed up policy computation but ensure that the value of the policy is correctly returned to the master.

Referring to the diagram of the slave component in Figure 5, the changes that are made are within the Compute Updated Reward step. This is where the limit is placed on the maximum number of agents that can visit a state. In the step where the joint policy is converted to a column (once the slave is done computing individual policies for each agent), this computation does not place a limit on the maximum number of agents to ensure that the column returned to the master is a correct representation of the joint policy (fulfilling the second desiderata criteria).

The idea we present above fulfills all four points of the desiderata in scaling up to handle many defender agents. It focuses on modifying the slave component, which has been shown to consume the majority of the runtime. The heuristic, while modifying the computation of the effectiveness value in the updated rewards, still reports an accurate column for the master component. If the column generated underestimated the effectiveness, this would result in an incorrect value for the LP as computed by the master. This may cause the Multiple-LP algorithm to choose the best LP incorrectly and therefore result in low valued strategy for the defender. This heuristic, as shown in Section 5.8, is extremely beneficial in speeding up while still providing a high level of solution quality.

4.3 Improving the Solution Quality of the Slave

Another approach that we considered in improving the runtime of the algorithm was generating a higher quality solution for the slave component (even at the expense of the slave component running slightly slower) with the notion that if the slave component produces a better column for the master, the column generation algorithm will converge more quickly to a solution, thereby speeding up the overall algorithm.

In the slave component, in Algorithm 1, we generate a policy for each agent by iterating over each agent in a single iteration (Line 2). Therefore, the policy of the first agent does not take into account the policies of all other agents.

The slave computes the optimal policy for the first agent assuming there are no other agents. The slave component then computes the optimal policy for the second agent given the policy for the first agent (which is now fixed and does not change). The policy of the third agent is computed with the knowledge of the policies of the first two agents. This continues until policies are generated for all agents.

Algorithm 3 SolveRepeatedSlave(y_b, $\boldsymbol{\mathcal{G}}$)

1: Initialize π^j, ψ_p, ψ_c
2: **while** $\psi_p \neq \psi_c$ **do**
3: **for all** $r \in R$ **do**
4: $\pi^j \leftarrow \pi^j - \pi_r$
5: $\boldsymbol{\mu}_r \leftarrow$ ComputeUpdatedReward($\pi^j, y_b, \mathcal{G}_r$)
6: $\pi_r \leftarrow$ SolveSingleMDP($\boldsymbol{\mu}_r, \mathcal{G}_r$)
7: $\pi^j \leftarrow \pi^j \cup \pi_r$
8: $\psi_p \leftarrow \psi_c$
9: $\psi_c \leftarrow$ ComputeObjective(π^j)
10: $\mathbf{P}^j \leftarrow$ ConvertToColumn(π^j)
11: return $\pi^j, \mathbf{P}^j$

As mentioned, the policy of the first agent does not consider the policies of any other agent as we use this heuristic to be able to scale up. We proposed modifying the slave component to include a *repeated* iterative process where instead of a single `for` loop (Algorithm 1, Line 2), we repeatedly iterate Lines 2 - 5, until we reach a local optimum where the policies of the defender agents do not change across iterations.

Algorithm 3 outlines the updated repeated iterative slave. ψ_p and ψ_c represent the computed objective value of the joint policy for the previous iteration and current iteration respectively. This is used to determine whether the joint policy has changed across iterations. The main difference between Algorithm 3 and Algorithm 1 is the outer while loop (Line 2) that compares the objective across iterations to see if it has improved or reached a local maximum. In Line 4, the joint policy, π^j is modified by removing the current individual policy of agent r. The updated individual policy for the agent r is then recomputed and re-added to the joint policy. After the individual policies of each agent is computed, the objective of the joint policy is computed in Line 9. While further improvements could be made, the question we focused on is whether this style of improvement in solution quality of individual joint policies would help us reduce the total run-time.

The rationale for this repeated iterative process in the slave is to improve the joint policy (and equivalent column) that is computed by the slave component and to provide a higher defender expected utility. First, we tested the solution quality of a single instance of running the slave, comparing the output of the single iteration slave versus the repeated iterative slave. This is to verify that the solution quality of the joint policy from the repeated iterative slave is higher

	1	2	3	4	5	6	7	8
Single iteration	6.939	5.152	3.009	3.160	5.094	4.374	6.676	7.083
Repeated iterative	7.305	5.416	3.053	3.246	5.432	4.422	6.821	7.203

Table 2: Comparison of solution quality for only one instance of the slave when using a single iteration versus repeated iterative slave

than the joint policy computed by the single iteration slave. We show this comparison in Table 2 where each column represents the solution quality after running a single instance of the slave component. Therefore, each of the values in this table measure the solution quality of a single defender pure strategy or joint policy.

In a follow-up test, we compared the performance of the repeated iterative slave versus a single iteration slave run over the whole game instance to find the defender's mixed strategy over the set of pure strategies generated via the column generation framework. This is different from the results in Table 2, where in this test we run the Multiple-LP algorithm including column generation to determine the defender's expected utility and mixed strategy. In a preliminary test, with 5 targets, 8 time steps, and 4 agents and averaged over 15 game instances, in comparing the repeated iterative slave versus a single iteration slave, the solution quality (defender expected utility) when using a repeated iterative slave was 0.861 while the solution quality for the single iteration slave was 0.849. The maximum improvement of the repeated iterative slave over the single iteration slave was 0.057. This shows that the overall solution quality of the repeated iterative slave is higher than the single iteration slave. This is what we expect for the repeated iterative slave as it computes a locally optimal joint policy compared to the single iteration slave.

5 Evaluation

This section begins by providing a motivating domain of security in the metro rail in Section 5.1. Section 5.2 introduces, motivates and provides background on security games. Section 5.3 provides the details of the parameters and scenarios used in the experiments. Section 5.4 explores the importance of modeling teamwork and uncertainty. Section 5.5 follows with a comparison of the various Dec-MDP solvers. Section 5.6 evaluates the various runtime improvements explained in Section 4. Section 5.7 examines the robustness of the algorithms. Finally Section 5.8 provides a summary of all the heuristics presented in this paper.

5.1 Motivating Domain: Security of Metro Rail

In recent news, there have been terrorism related events pertaining to metro rail systems across the world. In April 2013, two men were arrested for plotting to carry out an attack against a passenger train traveling between Canada and the

United States [11]. In August 2013 an article reported planned attacks by Al Qaeda on high-speed trains in Europe which prompted authorities in Germany to step up security on the country's metro rail system [47]. A presentation by Arnold Barnett suggested that the success of aviation security may be shifting criminal/terrorist activity towards other venues like commuter metro rail systems, and he also argues that "the prevention of rail terrorism warrants high priority" [25].

In the metro rail domain, the defender agents (i.e., canine, motorized) patrol the stations while the adversary conducts surveillance and may take advantage of the defender's predictability to plan an attack. With limited agents to devote to patrols, it is impossible for the defender to cover all stations all the time. The defender must decide how to intelligently patrol the metro rail system. Additional constraints include the defender agents having to travel on the train lines, thus being limited in path and sequences of stations and having to adhere to the daily timetables of the trains. Recent research on security games focused on the metro rail domain include the computation of randomized patrol schedules for the Singapore metro rail network [60] and security patrolling for fare inspection in the Los Angeles Metro Rail system [30].

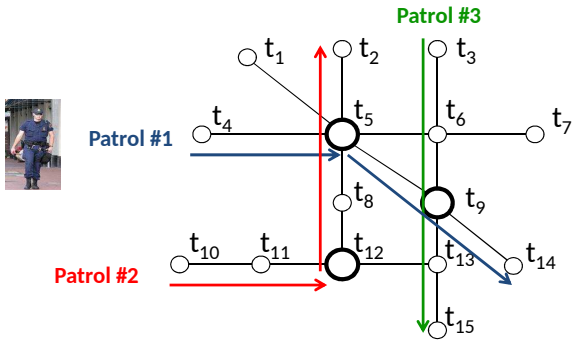


Figure 7: Example of the metro rail domain

In Figure 7, we give an example of the metro rail domain. Each of the circles represent a station, with the various lines corresponding to a separate metro rail line. For example, one line would be composed of the stations/targets: $\{t_4, t_5, t_6, t_7\}$. Another metro rail line is composed of stations $\{t_1, t_5, t_9, t_{14}\}$.

Not all stations have the same payoff, for example some stations may have transfers between multiple train lines and are more attractive for the adversary to attack (as shown in the figure with stations t_5, t_9, and t_{12} being represented with a larger circle). In this figure, we give three possible patrols that a single defender agent can execute with a single patrol being unable to visit all of the stations given the time constraints. The path of patrol 1 starts at station t_4, travels to station t_5, then visits t_9, and finally ends with station t_{14}.

Defender agents may engage in *teamwork* to patrol certain key areas that may be advantageous in thwarting the adversary compared to individual patrolling. What we mean is that defender agents may execute multiple patrols, e.g., Patrol 1 and 2 in Figure 7, and coordinate to visit a single station simultaneously (like station 5). Thus, if the adversary observes a coordinated set of defender agents patrolling a station, he will have to overcome multiple defenders if he decides to attack. To address teamwork in the metro rail domain along with the constraint that the defender agents must travel on the train lines that adhere to a fixed daily schedule (e.g., to allow the defenders to arrive simultaneously at a train station), we model the time as discrete, based on the train arrivals and departures. Indeed such discrete time is important to represent since even individual defender agent actions are based on train arrival and departure times.

Within this metro rail domain, we can see three factors that complicate teamwork and are not addressed by previous work in security games. First, uncertainty in execution may cause miscoordination. In particular, while defender agents are on patrol, one or more of them may be forced to deviate from the given patrol due to unforeseen events (we denote as execution uncertainty), such as questioning of suspicious individuals which results in delays and uncertainty in the patrol – while still needing to coordinate with other agents. This type of uncertainty occurs on a local level to the individual defender agent and is not known by the other defender agents (due to limited communication as described below). Using the example discussed above of teamwork, e.g., patrol 1 and 2, this type of uncertainty would cause the patrols to arrive at station 5 at different times, instead of having the agents visit station 5 together.

Second, a global event may cause a agent to leave the system and stop patrolling. This type of global event affects the entire team and impacts the coordination among patrol agents which is different from the local level events that may occur to an individual defender agent. One of the defender agents may get interrupted to deal with a serious bomb threat – the entire team may be alerted to this threat via an emergency channel and the responsible agents may take over the response, resulting in the agent stopping the patrol and requiring others to fill in any gaps as a team. The remaining agents will continue to patrol the metro rail system to guard against subsequent future attacks that may arise. Third, in this metro rail domain there is often limited communication among the defender agents. Reasons for this limited communication include the trains and stations being underground or the use of cell phones being jammed to avoid triggering of explosions or radio giving away the defender's coordinates or information (with the emergency channel reserved for emergencies). This

prevents defender agents from constantly communicating with other agents to determine their location.

5.2 Security Games with an Example Application in the Metro Domain

Security games were first formalized by Kiekintveld et al. [31] which is based on a two-player Stackelberg game between a defender (leader) and an attacker (follower). In a security game the leader (defender) plays a strategy first while the follower (attacker) observes the defender's strategy before choosing his response [20, 29, 57]. Thus, using the Stackelberg (i.e., leader-follower) model as a basis for security games allows us to capture the attacker's conducting of surveillance of the defender strategy before launching any major attack [31, 45, 64]. The security game is a two stage game: the defender plays a mixed strategy and the attacker then responds with an attack on a time and target; the game then terminates. The defender does not get to observe the attacker or form beliefs about the attacker. The focus of this section is how the security game model applies to the metro domain as considered in previous work such as in [30], but without coordination of multiple agents under uncertainty.

In this model, both the attacker and defender have a set of possible pure strategies. The attacker's pure strategies correspond to the set of target-time pairs, B, where each target-time pair $b = (t, \tau)$ is defined as t being the target to attack and τ being the time point to carry out the attack. In the train domain, targets correspond to stations in the metro system. The attacker chooses a single target-time pair to attack based on the observation of the defender's marginal coverage (defined in detail later in Section 2, but based on the concepts of marginals introduced in [31]). The defender's pure strategies correspond to visiting a set of target-time pairs given a set of agents. By convention, in the rest of the paper, we refer to the defender as *she* and attacker as *he*.

The payoffs for both the attacker and defender are dependent on whether the target-time pair is covered by the defender or left uncovered (based on the strategy of the defender). The defender's actions and capabilities influence the *effectiveness* of coverage on target-time pairs, allowing for partial effectiveness. Each target-time pair b has a payoff associated with it for both the attacker and defender, with $U_d^c(b)$ denoting the payoff for the defender if b is covered (100% effectiveness), and $U_d^u(b)$ denoting the payoff for the defender if b is uncovered (0% effectiveness) — we define defender expected utility under partial effectiveness later[27, 50, 66]. We choose to have payoffs on both the location and time, due to the payoff being dependent on time, e.g., in the train domain, at rush hour the payoffs are larger than in the middle of the night with very few passengers. The payoffs are influenced by the number of people at the station/trains because if an attack is carried out when there are a lot of people and the station is more crowded, then the attack will result in a greater number of deaths and injuries compared to an attack when the station and train lines are not as busy.

The payoffs for the attacker are in the same format, $U_a^c(b)$ and $U_a^u(b)$. A

common assumption for security games is that $U_d^c(b) > U_d^u(b)$ and $U_a^c(b) < U_a^u(b)$, i.e., when a defender covers b, she receives a higher reward while the attacker receives a lower reward compared to when the defender does not cover b [5, 28, 56]. The model in our paper allows a non-zero-sum game, where the sum of the defender's and attacker's payoff values may be non-zero.

The objective in the security game is to compute the defender mixed strategy that maximizes the defender's utility given the attacker's strategy where the attacker has full knowledge of the defender's strategy. In other words, the goal in security games is one of optimizing the use of the defender's limited security agents while taking into account the attacker's ability to observe the defender's mixed strategy and to respond optimally to such a strategy. Note that we compute the mixed strategy for the defender but only need to consider the pure strategies of the attacker [42]. This is because given a fixed mixed strategy of the defender, the attacker faces the problem that contains linear rewards and thus if a mixed strategy is optimal for the attacker, then so are each of the pure strategies in the support set of the mixed strategy (pure strategies that have a non-zero probability). Therefore, we do not need to consider the mixed strategies of the attacker.

This optimization goal is equivalent to finding the Strong Stackelberg equilibrium (SSE), which was first proposed by Lietmann [33]. Significant research on the strong Stackelberg equilibrium versus other types of Stackelberg equilibrium has already been done in previous work and led to SSE being commonly used in security game research [3, 12, 19, 26, 27, 28, 30, 31, 44, 45, 49, 50, 66].

5.3 Experimental Setup

The experiments detailed in the rest of this section were performed on a quad core Linux machine with 12 GB of RAM and 2.3 GHz processor speed. The test results were averaged over 30 game instances, with each game having random payoffs in the range [-10,10]. Unless otherwise stated, the scenarios are run over 8 targets, 4 agents, a patrol time of 80 minutes discretized into 10 minute intervals, 5% probability of delay, and 5% probability of a global events, using VI with append + cutoff + ordering. The graphs of the scenarios are formed by connecting targets together in lines of length 5, and then randomly adding $\frac{|T|}{2}$ edges between targets, to resemble train systems in the real world with complex loops. *All key comparisons where we assert superiority of particular techniques, e.g., as in Figure 20, are statistically significant with* $p < 0.01$.

5.4 Importance of Teamwork and Uncertainty

In this section, we focus on showing that the problem we are solving and the way we model it, provides significant improvement over previous models. The purpose is to show that modeling and solving for defender teamwork and uncertainty is an important research topic and that generating a high quality solution is not trivial. Given the complexity of the problem and model that we are solving, a reasonable question would be whether solving a more simple model, such

as one that does not take into account the effectiveness of multiple agents or uncertainty, would provide a solution quality that is "good enough" or close to the solution that we receive from our algorithm that takes into account teamwork and uncertainty. This section shows that this is not the case, where solving a more simple model provides a significantly lower solution quality compared to the model that we present in this paper. In addition, these experiments are benchmarks for our algorithm in providing lower bounds. We first demonstrate that the algorithm we use to solve this model provides a significant improvement over a naive approach of a uniform random strategy. Next, we present three properties that we investigated: (i) defender teamwork in the form of additional effectiveness for the defender team as multiple agents visit the same state; (ii) global events that are handled in our model versus one that ignores these types of events; (iii) execution uncertainty in the form of delays in the defender's patrol. The following figures show that each property that we model provides a considerable improvement in the defender's strategy versus a model that ignores the property.

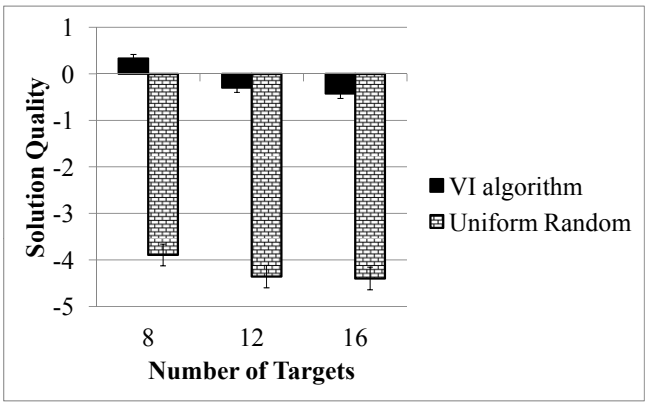


Figure 8: Comparison of our VI algorithm versus a uniform random strategy

First, we compare the solution quality of our VI algorithm versus a uniform random strategy in Figure 8. The x-axis denotes the number of targets while the y-axis shows the solution quality (defender expected utility). The purpose of this comparison is to use the uniform random strategy as an initial benchmark to measure the performance and increase in solution quality that our VI algorithm provides to the defender team. This figure shows that for varying numbers of targets, our algorithm significantly outperforms a naive uniform random approach.

Figure 9 shows the benefit received in our model's ability to handle teamwork. This demonstrates the improvement in the solution quality, or defender expected utility, that comes from the increased effectiveness of having multiple defender agents visit the same state. More specifically, it shows the difference in solution quality between our algorithm that generates policies that takes into

Figure 9: Benefit of considering the effectiveness of multiple agents

account benefit to having multiple agents covering the same target-time pair, $\texttt{eff}(s, b) = 1 - (1 - \xi)^{\sum_i I_{s_i=b}}$, and an algorithm that generates policies that ignores additional effectiveness, $\texttt{eff}(s, b) = \xi \cdot I_{b \in s}$ (i.e., it is ξ as long as at least one agent covers b). This algorithm that ignores additional effectiveness is still solving individual MDPs for each defender agent (in the slave component) and providing joint policies for the master component.

For both algorithms, when evaluating and computing the defender expected utility, if multiple agents visit the same state, the defender receives an additional effectiveness. In other words even for the algorithm that generates policies that ignore multiple agents, when evaluating and computing the defender expected utility, if there is more than one agent at a state, the defender will get an additional effectiveness. As the number of defender agents increases, the solution quality for when there is a benefit to having multiple agents increases at a faster rate than when there is no benefit of multiple agents visiting the same state (no teamwork).

Figure 10 further illustrates the expressiveness of our teamwork model. It compares the solution quality when we consider global events versus solving under the assumption of no global events. The x-axis denotes the number of targets while the y-axis shows the solution quality. In the latter case, the system solves the model under the assumption that there is no global event, and we compute the defender expected utility if there is a 5% probability of global events at each time step. This shows the need and improvement to incorporating and handling global events.

Figure 11 shows the importance of taking account execution uncertainty in the form of delays. The x-axis is the number of targets and the y-axis is the solution quality. It compares the solution quality of our algorithm that considers and plans for uncertainty (in the form of delays), to an algorithm that does not take into account execution uncertainty (i.e., assumes that the

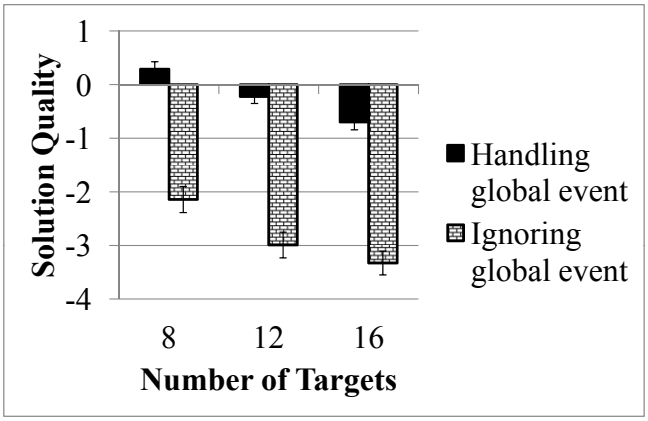


Figure 10: Solution quality of handling global events versus ignoring global events

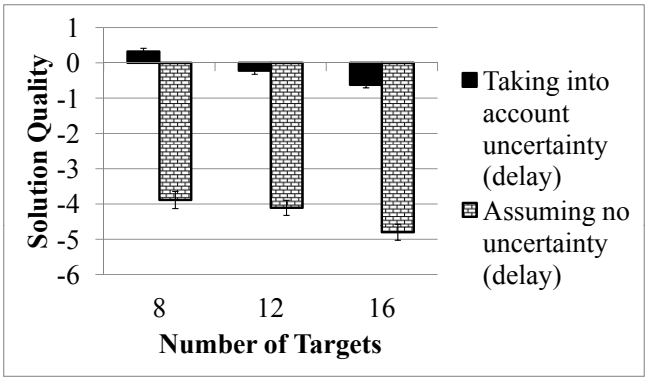


Figure 11: Comparison of solution quality taking into account the probability of delay

policy/patrol schedule will be performed exactly as indicated and that there will be no unforeseen events or delays). When executing the policy that does not take into account delays, when a delay is encountered, the policy terminates with no action. The solution quality of the algorithm that assumes no uncertainty generates a mixed strategy for the defender, that is then analyzed with the assumption of a 5% probability of delay. This figure reinforces the usefulness and value to handling execution uncertainty with multiple coordinated defender agents.

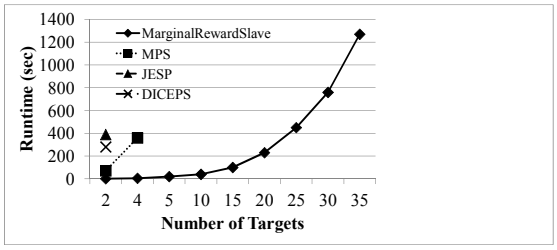


Figure 12: Comparison of various Dec-MDP solvers

5.5 Comparison with other Dec-MDP solvers

In Figure 12 we compare the runtime of the VI-based slave for one iteration (no column generation) with other algorithms for Dec-MDPs such as MPS [14], JESP [36] and DICEPS [39]—this is the only figure in this section that focuses only on the slave and not on the master-slave algorithm in full.[2] The x-axis shows the number of targets and the y-axis is the execution time (seconds). We are thankful for the advances in Dec-MDP algorithms and are in debt to the multi-agent planning under uncertainty community for important research that we utilize in our algorithm. There is new fertile ground for new research that exploits deeper insights from MPS along with demonstrating the importance towards fast heuristic algorithms to solve Dec-MDPs.

This figure shows that JESP and DICEPS run out of memory for more than two targets, while MPS runs out of memory for more than four targets. For a single iteration of the slave, MPS takes over six minutes with four targets, whereas our algorithm takes less than 10 seconds. This suggests that security games can benefit from a new family of fast approximate Dec-MDP algorithms, such as our VI-based slave, that provides a new direction for further Dec-MDP research.

5.6 Evaluating Runtime Improvements

This section begins with a comparison among all the runtime heuristics presented in Section 4. In Section 5.6.1, we show the increased performance when the slave component is modified to place a limit on the number of agents at each state. Then in Section 5.6.2 we further explore the impact of the repeated iterative slave.

The first two figures, Figure 13 and 14 compare the runtime and solution quality across the multiple heuristics as described in Section 4. In both figures, the x-axis is the number of targets while the y-axis is the runtime in minutes for Figure 13 and the solution quality in Figure 14. Focusing on Figure 13, the heuristics that provide the fastest runtime are: Append + Cutoff, Append

[2]We would like to thank Jilles Dibangoye for providing the MPS algorithm and Matthijs Spaan for providing the MADP toolbox (for JESP and DICEPS).

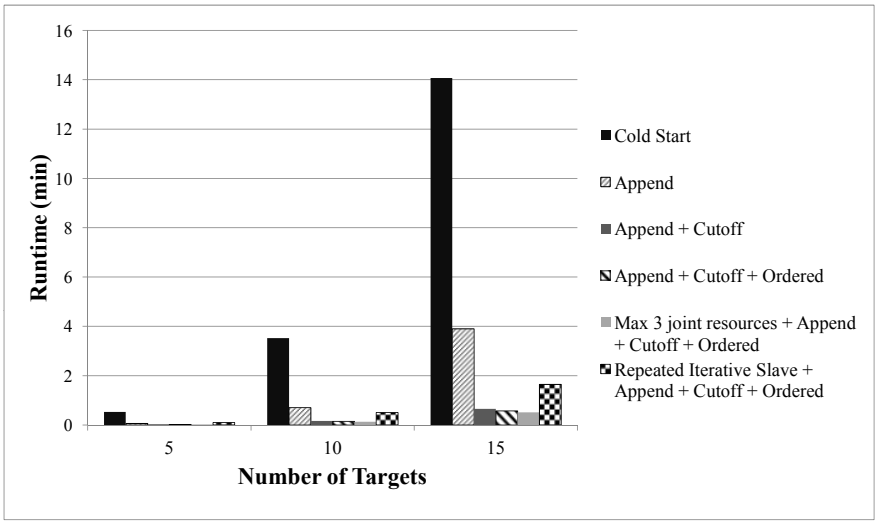


Figure 13: Runtime comparison of heuristics

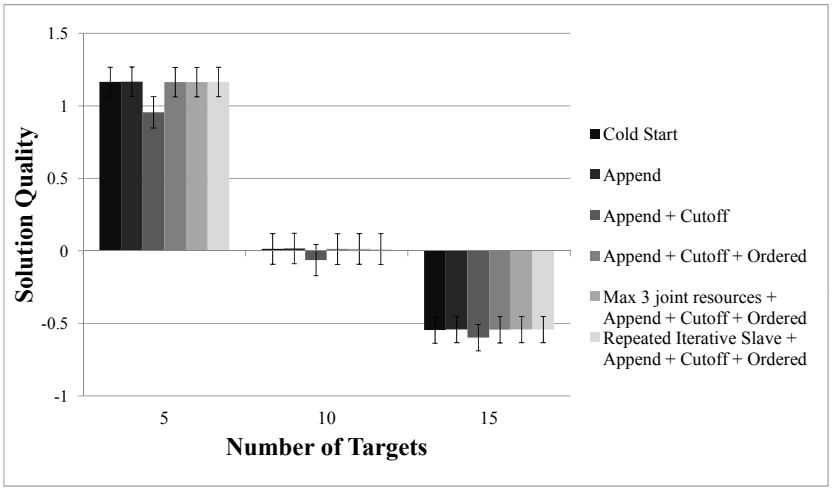


Figure 14: Solution quality comparison of heuristics

+ Cutoff + Ordered, and Max 3 joint agents + Append + Cutoff + Ordered. Both Cold Start and Append take the longest time to run.

When comparing the solution quality, in Figure 14, all the heuristics compute approximately the same solution quality or defender expected utility except for the Append + Cutoff heuristic. The rationale as to why there is a decrease in solution quality for the Append + Cutoff heuristic is that due to the cutoff

function where there is a limit placed on the total number of columns generated within a single LP, this may result in a lower defender expected utility. By adding the Ordered heuristic, we are solving the LPs that are less attractive to the attacker first, thereby allowing additional columns to be generated and used by the LPs that are solved later in the algorithm to give the defender a larger number of columns or defender pure strategies to use. These two figures show that the Append + Cutoff + Ordered heuristic runs at least as fast as the Cold Start, Append, and Append + Cutoff algorithms, while also computing approximately the same solution quality (and outperforming Append + Cutoff). Therefore, we focus on improving the Append + Cutoff + Ordered heuristic in the following section to handle situations where there is a larger number of defender agents.

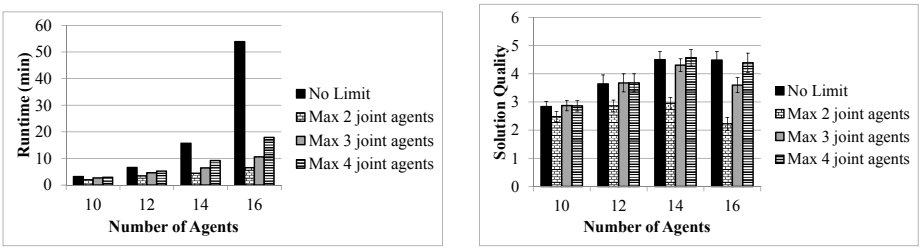
5.6.1 Maximum agents per state in the slave

We show in this section that the Append + Cutoff + Ordered heuristic, cannot scale up well when further increasing the number of agents. Figure 15 shows the runtime improvements and solution quality when limiting the maximum number of agents at a state. The "No Limit" column represents the algorithm that uses Append + Cutoff + Ordered, but does not place a limit as to the maximum number of agents that can visit a state (equivalent to the number of agents at the same target-time pair). We show the solution quality and runtime when we place a limit of 2, 3, and 4 maximum agents at a state.

Figure 15(a) shows the runtime in minutes (y-axis) as the number of agents increases from 10 to 16 (x-axis). Even using the append, cutoff, and ordering improvements, when there are 16 agents, the program takes over 50 minutes to run, compared to under 20 minutes to run when we limit the maximum number of agents at a state to 4. In Figure 15(b), the x-axis is the number of agents and the y-axis is the solution quality. This figure shows the amount of loss in solution quality when placing limits on the number of agents that can visit the same state with the error bars denoting a 95% confidence interval. Notice that when we place a limit of 3 defender agents that can visit a state, when there are a total of 10 or 12 agents, the solution quality is approximately the same as when we do not place a limit on the number of agents that can visit the same state. However, when the number of agents increases to 14 and 16, placing a limit of only 3 agents results in a loss in solution quality. Overall, these figures show the improvement in runtime for placing limits on the maximum number of agents that can visit a state, while not significantly decreasing in solution quality (e.g., when there is a limit of 4 agents).

5.6.2 Repeated iterative slave

Figures 16, 17, and 18 compare the performance of the single iteration versus the repeated iterative slave algorithm as explained in Section 4.3. In all three figures, the x-axis denotes the number of targets. In Figure 16, the y-axis is solution quality while in Figure 17 the y-axis is the runtime in minutes. For Figure 18



(a) Runtime

(b) Solution Quality Comparison

Figure 15: Improvements in limiting maximum number of agents

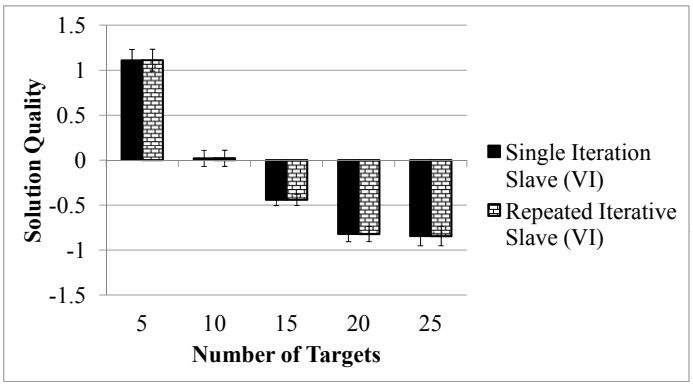


Figure 16: Solution quality comparison of the single versus repeated slave

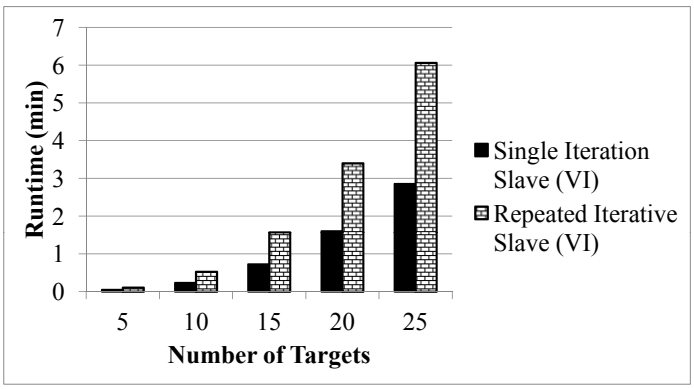


Figure 17: Runtime comparison of the single versus repeated slave

the y-axis is the total number of column generation iterations. As mentioned in Section 4.3, for initial test cases, the repeated iterative slave computed a higher

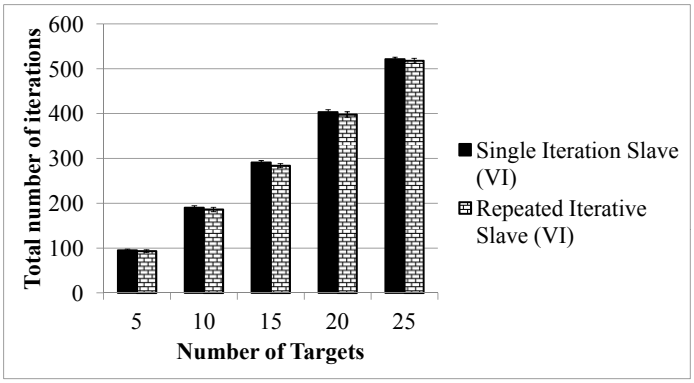


Figure 18: Comparison of the number of iterations of the single versus repeated slave

solution quality than the single iteration slave.

However, as we ran additional tests as shown in these figures, the repeated iterative slave approach did not provide a significant increase in the solution quality over the single iteration approach as we initially thought, nor did it improve the overall runtime. In addition, both algorithms executed for approximately the same number of total iterations. The intuition for both the repeated iterative slave and the single iteration slave resulting in a similar solution quality is that although the repeated iterative slave produces a better joint policy for a single iteration than the single iteration slave, recall that the master component computes a mixed strategy over all joint policies generated by the slave component. Therefore, the initial joint policies computed by the repeated iterative slave may be better, but over multiple iterations of column generation, the single iteration slave will generate effective joint policies so that the resulting defender expected utility is similar to the resulting defender expected utility for the repeated iterative slave.

These results indicate that improving the solution quality of the joint policy returned by the slave is by itself not sufficient to guarantee a faster run-time or higher solution quality. In fact, the effect may be counter-productive. Thus, whereas we have settled on a particular heuristic approach, we have shown now in various ways that going towards a slave that computes an optimal policy fails to scale up (Figure 12), going towards a slave that computes a potentially higher quality policy fails to degrade runtime while not improving solution quality (Figures 16 and 17), and going towards a slave that ignores the uncertainty provides a very low solution quality (Figure 11). This does not preclude further improvements to the heuristic slave presented in this paper, but suggests that such an improvement will require a deeper exploration.

5.7 Robustness

There are two types of robustness issues that we explore. The first type of robustness that we study examines the impact of uncertainty in different network structures. An example of this includes the performance of the algorithm when the probability of delay increases. In the real world, this value will be determined based on the frequency that the actual patrols get delayed or interrupted. In addition to determining the performance of the algorithm as the probability of delay changes, we evaluate the impact of network structure. The rationale for studying the robustness of the algorithm across different types of network structure, is to see if having different types of connectivity among the targets/stations would affect the defender's strategy and expected utility. Would having a more sparse graph or densely connected graph influence the defender's expected utility and how would that expected utility change as we change the transition uncertainty? We conduct experiments across different types of network structures and varying levels of transition probability in Section 5.7.1 to show that a network structure that has more edges (e.g., complete graph) provides greater resilience even as the probability of delay increases.

The second type of robustness that we explore addresses the impact of uncertainty over uncertainty. In our algorithm, we assume a probability of delay, where the defender agent may get delayed during a patrol. However, there may be uncertainty in what this probability may actually be in the real world. We present a different approach in generating the defender's policy within the slave, to provide a more robust solution to this type of uncertainty.

As described in Section 3.3, the slave generates a policy for each defender agent using value iteration. We present a different way to solve the MDP by using soft-max value iteration (SMVI) [58] to provide a more robust solution to uncertainty. SMVI is similar to VI except that the soft-max function is used instead of max while computing the value function of a state s. SMVI generates randomized policies – i.e., randomized pure strategies – associating probability $\pi_r(s_r, a_r)$ to each action a_r at each state s_r. Formally,

$$V(s_r) = \text{softmax}_{a_r} V(s_r, a_r) \equiv \log \sum_{a_r} e^{V(s_r, a_r)} \tag{19}$$

$$\pi_r(s_r, a_r) = \frac{e^{V(s_r, a_r)}}{e^{V(s_r)}} \tag{20}$$

SMVI was first explored for its ability to speed up convergence, as in [58]. In our experiments SMVI did not provide significant runtime improvement, however we discovered that the randomized policy obtained from SMVI provides robustness to uncertainty in our estimates of transition probabilities, which is a highly useful feature since this uncertainty often arises in practice. The intuition behind SMVI providing robustness to uncertainty stems from the fact that the SMVI algorithm computes a policy that spreads out the probability of choosing an action at each state, instead of choosing only one action at each state (VI). In the presence of uncertainty, if the action that is chosen by VI

is no longer the best action, it will still be chosen with a probability of 1 and therefore the updated optimal action due to uncertainty will now be chosen with a probability of 0. With soft-max, the probability over the action to take at each state is distributed over the set of possible actions based on their values. Therefore when noise or uncertainty is added, the randomized policy will have a non-zero probability of choosing the updated best action (or a close-to-best action). For example, if there are two possible actions, a_1 and a_2, that give the values 5 and 4 respectively, then the VI algorithm will generate in a policy that chooses a_1, 100% of the time. However, the SMVI algorithm will compute a policy that chooses a_1, 73.1% of the time and a_2, 26.9% of the time. If noise is added to the system that now results in a_2 giving a *higher* value than a_1, the VI-based policy will be choosing a suboptimal action, or a_1, 100% of the time or the optimal action, a_2, 0% of the time. However, the SMVI-based policy will choose the optimal action, a_2, 26.9% of the time, thereby giving the defender a higher value. Such probability will be significant especially when there are many close-to-optimal pure policies.

To test these level of robustness of our algorithms, we first evaluate the algorithm under different graph structures. Next, we show the robustness of the SMVI slave with uncertainty in the transition probability. Then, the performance of both VI and SMVI slaves are studied with variations in the payoff structure.

5.7.1 Varying graph structure

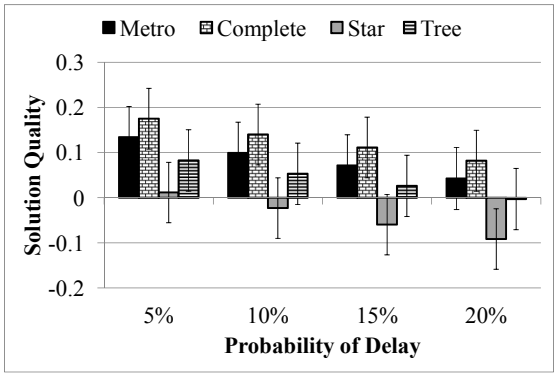


Figure 19: Comparison of different graph structures under varying probabilities of delay

In all the prior figures, the graph structure of the targets resemble a metro/train system composed of connected lines. In Figure 19, we compare the impact of different graph structures while also varying the probability of delay. The x-axis is the probability of delay and the y-axis is the solution quality. The payoffs

for the targets are the same across all four graph structures. The four graph structures that were examined are:

- Metro - a metro-based graph as described in the first paragraph of Section 5
- Complete - a complete graph where all targets are connected to each other
- Star - a star graph where only one internal target is connected to all other leaf targets
- Tree - a binary tree graph where each target has at most two "children" targets

Across varying levels of probability of delay, the complete graph always gives the highest solution quality, followed by metro graph, tree graph, and finally star graph. The complete graph gives the highest solution quality because each target is connected to the other, thereby having less constraints for the paths of the patrols/policies. The star graph gives the lowest solution quality because for the defender agent to visit two leaf targets, the agent must traverse past the internal target, thereby not being able to visit as many targets within the maximum patrol time. As the probability of delay increases, from 5% to 20%, the solution quality for all graph structures decreases, however the complete graph continues to enjoy the highest solution quality. This is because there is increased amounts of uncertainty as to the location of the other defender agents.

From this figure and set of experiments, we show that adding more connectivity in the graphs that are used for patrolling is valuable and improves the overall solution quality. Even when there is a high amount of uncertainty, having a highly connected graph still results in a higher performance of the algorithm.

5.7.2 Evaluating SMVI and VI

Figure 20 shows the difference in solution quality of soft-max value iteration (SMVI) versus value iteration (VI) in the presence of uncertainty in transition probability. The x-axis is the number of targets and the y-axis is the solution quality. The uncertainty that is added corresponds to the probability of the transition uncertainty being different than the initial assumed value. In this scenario, SMVI and VI obtain Dec-MDP based pure strategies with the assumption that the probability of delay of 5% (the value that is assumed is the probability of delay). We evaluate how the solution quality is impacted with a probability of delay of 10%, while the algorithms assume a delay of 5%. In other words, we measure the change in solution quality (defender expected utility) when the algorithms generate a defender strategy that assumes that the probability of delay is 5% but evaluate the defender strategy if there is actually a probability of delay of 10%. This shows that without any uncertainty SMVI performs worse than VI, but with uncertainty in the transition probability, SMVI gives a higher solution quality than VI. Thus, SMVI is a more favorable option given uncertainty in transition probability.

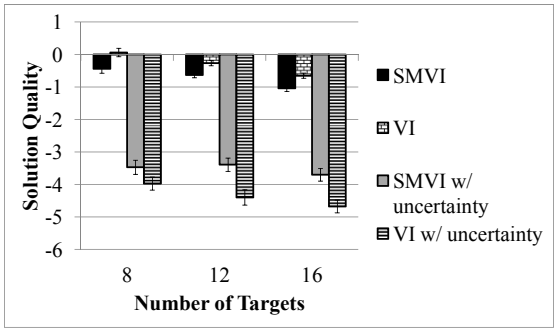


Figure 20: Comparison of SMVI and VI under uncertainty in transition probability

5.7.3 Varying payoff structure

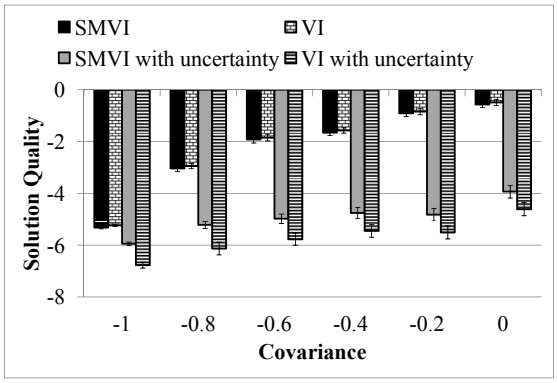


Figure 21: Comparison of SMVI and VI under uncertainty in transition probability with varying payoff structures

In Figure 20, both SMVI and VI have a significant decrease in solution quality when there is uncertainty in the transition probability. To further explore if this phenomenon happens across different scenarios, we generated different payoff structures by varying the covariance values of the payoffs using the covariance game generator of the GAMUT package [38]. We find that regardless of the payoff structure, the SMVI-based algorithm provides a greater robustness to uncertainty in the transition probability compared to the VI-based algorithm.

The covariance value is chosen from the range $[-1.0, 0.0]$, which provides the correlation between the defender's payoff and the adversary's payoff. The rewards for both the defender and adversary are positive integers in the range $[1, 10]$ while the penalties for both the defender and adversary are negative

integers in the range $[-10, -1]$. A covariance value of -1.0 is equivalent to a zero-sum game where if the defender's reward is 8, then the corresponding adversary's penalty would be -8. A covariance value of 0 is equivalent to random payoffs where there is no correlation between the defender's and adversary's payoffs. Figure 21 shows the change in solution quality for both SMVI and VI under uncertainty in the transition probability with different types of payoff structures. When the covariance value is -1.0, or under a zero-sum game, note that the solution quality of the SMVI-based algorithm drops by only 0.6 when there is uncertainty in the transition probability, while the VI-based algorithm drops by 1.5. In other words, the drop in solution quality of the VI-based algorithm is 50% larger than the drop in solution quality for the SMVI-based algorithm when there is uncertainty in the transition probability.

As the covariance value increases from -1.0 to 0, the solution quality when there is no uncertainty in the transition probability increases at a faster rate than the solution quality with uncertainty in the transition probability. Under the Strong Stackelberg Equilibrium [10, 33, 31], the follower (adversary) will choose the optimal strategy (state to attack) for the leader (defender). This leads to a higher solution quality for the defender when there is no transition uncertainty. When there is uncertainty added to the system, the adversary may instead choose to attack a state that gives the defender a significantly worse expected utility, thereby resulting in a larger drop in solution quality, as seen in Figure 21, as the covariance value deviates from -1 (a zero-sum game). In the more realistic cases, where the payoff structure is closer to real-world scenarios (with the covariance being closer to -1 or the left portion of the figure), there is less degradation in the solution quality for both SMVI- and VI-based algorithms with SMVI continuing to provide greater robustness compared to VI.

5.8 Summary of Heuristics

This section compares all the heuristics and extensions presented in this article. The Cold Start, Append, and Append + Cutoff heuristics are not included in the table as they are dominated by the Append + Cutoff + Ordered heuristic in both runtime and solution quality. Table 3 compares the four primary heuristics/extensions proposed in this paper. The first heuristic, Append + Cutoff + Ordered, works well for scenarios where the user may have a lot of targets but less than ten defender agents. When there are a significant number of defender agents, the user should choose to use the heuristic that places a maximum number of agents at a state (within the slave) in addition to the Append + Cutoff + Ordered heuristic. For scenarios where there may be a lot of uncertainty in the parameters, the Soft-Max Value Iteration provides a more robust defender strategy.

Heuristic	Positives	Negatives
Append + Cutoff + Ordered	Scales up as number of targets increases	Fails to scale as number of agents increases
Maximum number of agents at a state with Append + Cutoff + Ordered	Scales up as number of agents increases	Have to determine suitable limit for agents at a state
Repeated iterative slave	Finds a higher quality joint policy for the defender for each slave iteration	Overall solution quality does not significantly improve and takes longer time to run
Soft-Max Value Iteration (SMVI)	Computes robust solution when uncertainty exists in the parameters	Generates a solution quality worse than value iteration (VI) when there is no uncertainty in the parameters

Table 3: Comparison of Heuristics

6 Related Work

There are two main areas of related work. The first area is on security games. The second area is the work done on Decentralized Markov Decision Processes. We next explore these two areas in more detail.

6.1 Security Games

As security game research is discussed earlier (see Section 5.2), we focus on research most relevant to the problem discussed in this paper. Whereas early work on commitment to mixed strategies in Stackelberg games is explored in [61], it was the DOBSS algorithm [42] that used a mixed integer program and opened the door to applications of Stackelberg games to security.

Following DOBSS, ORIGAMI and ERASER algorithms [31] were developed with ORIGAMI providing a polynomial time solution for security games with no scheduling constraints (an example of scheduling constraints is patrolling a metro system where the defender is restricted in the stations that can be visited based on patrol time and metro lines). ERASER-C [31] provided a more compact representation of the defender strategies for multiple agents (compared to DOBSS) while also handling scheduling constraints that arise from the defender's strategies. ASPEN [26] was later developed which utilized a branch-and-price approach to generate a subset of the defender's pure strategies while still computing the optimal solution for the defender allowing arbitrary scheduling constraints. This significantly improved the efficiency of solving security games with scheduling constraints. However, these algorithms do not consider teamwork and joint activities among the defender agents.

Shieh et al. [50] introduced coordination and teamwork among defender agents in security games via the use of joint activities for the defender. However they do not consider execution uncertainty, such as delays, that may arise in the defender agents' strategy. We show in Section 5.4, Figure 11 the impor-

tance of accounting for execution uncertainty in the defender strategy. Jiang et al. [30] explored handling execution uncertainty of defender patrols in security games via the use of Markov Decision Processes, but do not consider coordination and teamwork among the multiple defender agents and instead focuses on a single defender, or multiple independent defender agents that do not consider the additional effectiveness that arises from having multiple agents at the same target.

Research in security games also includes defending mobile targets [9], patrolling in extensive-form infinite-horizon games [5], simulations and tools for maritime security [29], and multiple self-interested defenders for games with a network environment [52]. Additional techniques in solving security games have included the use of a sequence-form double-oracle algorithm [8]

6.2 Decentralized Markov Decision Process

In this paper we turned to Dec-MDPs in generating joint policies for the defender's pure strategies. The complexity of Dec-MDPs have been shown to be NEXP-complete [7]. One of the earliest models of Dec-MDPs addressed the issue of scalability when there exists transition independence among the agents and is shown to be NP-complete [6]. Transition independent Dec-MDPs have been extended in a variety of ways. For example, one extension allows partial observability in network structures known as network-distributed POMDP or ND-POMDP [37]. Another extension of Dec-MDPs includes a decision theoretic model known as the decentralized sparse-interaction Markov decision process (Dec-SIMDP) [34], a subclass of Dec-MDPs, where local interactions and communications are abstracted to interaction areas and observable interactions.

Given the complexity of Dec-MDPs there has been a lot of work exploring ways to increase the scalability of solving these types of problems. Spaan and Melo proposed an interaction-driven Markov game, which is a model that extends Dec-MDPs and takes advantage of the situations where the interaction between the agents are a local phenomenon, and provides a fast approximate solution that exploits the structure [54]. Roth et al. [48] explored solving multi-agent domains with collective observability where factored policy representations are used. Dec-MDPs have also been reformulated as a bilinear program which allows for efficient algorithms to solve this kind of problem [43]. Dibangoye et al. [14], present an algorithm to solve transition independent Dec-MDPs while also providing error-bounds and fast convergence rates via the use of continuous state MDPs and piecewise linear convex functions. Additional research has focused on solving infinite-horizon transition independent Dec-MDPs [15].

Communication among agents in a decision theoretic, decentralized environment extending Dec-MDPs has been studied by Goldman et al. [22]. The Dec-MDP model is extended to Dec-MDP-Com model that includes the language of communication and cost to transmit a message. The use of communication can potentially help overcome the issues that arise from miscordination and provide a more robust solution. Another framework known as Dec-SMDP-Com [23] for a decentralized semi-Markov decision process with direct communication has

been used to represent communication within multi-agent planning in stochastic domains, where the agents operate independently between communication.

A general framework that has been used to solve multi-agent planning problems is the Expectation-Maximization (EM) framework [13], which has helped in scaling up. In solving infinite-horizon multi-agent sequential decision making problems, Kumar and Zilberstein, reformulated the problem to a set of dynamic Bayes nets and use the EM algorithm to find the optimal policy of the dynamic Bayes nets [32]. Another area of research related to sequential decision making under uncertainty is interactive partially observable Markov decision processes (I-POMDP) [21], where the beliefs of the agents are not constrained just by the state space, but include the physical environment and models of other agents. Research has focused on graphical models to represent and solve I-POMDPs [16, 17] along with a policy iteration algorithm to solve I-POMDPs [53].

Recent multi-agent literature for decision making under uncertainty has started exploring the notion and impact of time [1, 35]. Amato et al. [1] extend the Dec-POMDP model to address macro-actions (actions that require different amounts of time) in determining what macro-actions to execute and for which agent. A case study by Messias et al. [35] applies the Generalized Semi-Markov Decision Processes to a multi-robot scenario while demonstrating that the modeling of asynchronous events over continuous time gives a greater solution quality compared to discrete time models.

In this paper, we exploit the advances already provided by the Dec-MDP community in providing a rich set of algorithms and ideas, and the algorithm devised in this paper builds on these ideas. There are two major differences between the work in this paper and previous work. The first major difference in this paper is the addition of an adversarial agent that is able to respond to the joint policy of the Dec-MDP. The adversary is able to strategically determine the effectiveness of the defender's strategy (which is based on the joint policies from the Dec-MDP) that the defender must strategize against. The slave component of our algorithm can use any Dec-MDP solver as the purpose of the slave is to generate a joint policy for the defender agents. The contribution of this paper stems from the column generation framework that allows for the decomposition of the security game in the master component and the joint policy for the defender in the slave component allowing the integration of security games and Dec-MDPs. This decomposition brings forth a second key difference between the work presented in this paper and previous work: our emphasis is not on generating just one single optimal joint policy via the Dec-MDP algorithms employed in our slave component, but a number of joint policies. This shift in emphasis has led us to develop heuristics that allow for fast generation of good enough policies rather than focusing on a single optimal or near-optimal policy.

The contribution this paper provides to the Dec-MDP community is a new domain where Dec-MDP research can be used. Dec-MDPs can be used in game theoretic problems to compute defender strategies under uncertainty. This differs from prior Dec-MDP research areas as this opens up a need for Dec-MDP algorithms that focus on rapid computation of policies rather than a single op-

timal or near-optimal policy.

7 Conclusion

The key contribution of this paper is opening up a fruitful new area of research at the intersection of security games and multi-agent teamwork. We present a novel game theoretic model that for the first time addresses teamwork under uncertainty for security games. To solve this model, we present an algorithm that leverages column generation and iterative Dec-MDPs to generate defender strategies under uncertainty. Additionally, we present heuristics to improve the runtime and demonstrate the robustness of using randomized pure strategies.

A future direction we would like to inquire into is to look at various team types that allow heterogeneous team members (e.g., three boats, two boats and a helicopter, one boat and two helicopters) in security games, where there are costs associated with each defender agent (e.g., one boat has the same cost as two helicopters). These heterogeneous agents will also have different levels of effectiveness based on which agents are conducting a joint activity/working together. For example, having a helicopter and a boat visit the same target will provide increased effectiveness compared to having two helicopters or two boats visiting that target. The challenge is to find the optimal team composition given a fixed cost with the additional complexity of varying levels of effectiveness among heterogeneous defender agents. Another direction of future work could be to allow for a richer set of defender policies allowing for observation uncertainty (and hence we would need to bring in Dec-POMDPs for coordination), requiring us to handle the significant additional complexity of Dec-POMDPs.

8 Acknowledgments

We are grateful to the reviewers of this article for their valuable comments and suggestions. We thank Samantha Flores and Albert Venegas for their contribution and help in running experiments and generating figures. This research was supported by the United States Department of Homeland Security through the National Center for Risk and Economic Analysis of Terrorism Events (CREATE) under award number 2010-ST-061-RE0001 and MURI grant W911NF-11-1-0332.

9 Author Biographies

Eric Shieh is a research scientist at the Lockheed Martin Advanced Technology Laboratories. His research focuses on multiagent systems and game theory in real world applications. He received his Ph.D. at the University of Southern California in computer science. He obtained his M.S. in computer science from the University of Southern California and B.S. in computer science from Carnegie Mellon University.

Albert Xin Jiang is an assistant professor in the Department of Computer Science at Trinity University. He received his PhD from the Department of Computer Science at the University of British Columbia, and was a postdoctoral research associate in the TEAMCORE research group at the Department of Computer Science at the University of Southern California. Much of his research is addressing computational problems arising in game theory, including the efficient computation of solution concepts such as Nash equilibrium, Stackelberg equilibrium and correlated equilibrium, as well as applications of game-theoretic computation to real-world domains such as large-scale infrastructure security and electronic commerce. He has published over 50 articles in refereed international conferences and journals, and won the best student paper award at ACM-EC 2011 and was finalist for the best paper award at AAMAS 2013. He received the Canadian Artificial Intelligence Association (CAIAC) Doctoral Dissertation Award in 2012 and was runner-up for the IFAAMAS Victor Lesser Distinguished Dissertation Award. He has led research projects on applying game theory to security that have been successfully deployed at the Los Angeles Metro and the Staten Island Ferry.

Amulya Yadav is a third year Ph.D. student in the Computer Science Department at the University of Southern California (USC), where he works in the Teamcore Research Group, while being advised by Prof. Milind Tambe. Before coming to USC, he worked as a Software Development Engineer at Amazon.com after graduating with a CS B. Tech from Indian Institute of Technology Patna.

Pradeep Varakantham received his Ph.D. degree in Computer Science from the University of Southern California and he was a post doctoral fellow at Carnegie Mellon University. Currently, he serves as assistant professor at Singapore Management University. His research is focussed on developing agent and multi-agent systems for urban environments. He is an author or co-author of more than 50 international publications at top tier journals (JAAMAS, JAIR) and conferences (AAAI, IJCAI, NIPS, UAI, AAMAS, ICAPS) in AI. He was nominated for best senior program committee member at AAMAS'13 and one of his papers was nominated for best student paper at AAMAS'09. He has organised and co-chaired multiple workshops on planning under uncertainty, multi-agent coordination and game theory for security. He serves on the program committee of most top tier conferences (AAMAS, AAAI, ICAPS, IJCAI) and reviews for most top tier journals (JAIR, AIJ, JAAMAS) in Artficial Intelligence.

Milind Tambe is Helen N. and Emmett H. Jones Professor in Engineering at the University of Southern California(USC). He is a fellow of AAAI (Association for Advancement of Artificial Intelligence) and ACM (Association for Computing Machinery), as well as recipient of the ACM/SIGART Autonomous Agents Research Award, Christopher Columbus Fellowship Foundation Homeland security award, the INFORMS Wagner prize for excellence in Operations Research practice, the Rist Prize of the Military Operations Research Society, IBM Faculty Award, Okawa foundation faculty research award, RoboCup scientific challenge award, Orange County Engineering Council Outstanding Project Achievement Award, USC Associates award for creativity in research

and USC Viterbi use-inspired research award. Prof. Tambe has contributed several foundational papers in agents and multiagent systems; this includes areas of multiagent teamwork, distributed constraint optimization (DCOP) and security games. For this research, he has received the "influential paper award" from the International Foundation for Agents and Multiagent Systems(IFAAMAS), as well as with his research group, multiple best paper awards at conferences such as the International Conference on Autonomous Agents and Multiagent Systems and International Conference on Intelligent Virtual Agents. In addition, the real-world deployments of the "security games" framework and algorithms pioneered by Prof. Tambe and his research group has led them to receive the US Coast Guard Meritorious Team Commendation from the Commandant, US Coast Guard First District's Operational Excellence Award, Certificate of Appreciation from the US Federal Air Marshals Service and special commendation given by the Los Angeles World Airports police from the city of Los Angeles. For his teaching and service, Prof. Tambe has received the USC Steven B. Sample Teaching and Mentoring award and the ACM recognition of service award. Recently, he co-founded a company based on his research, ARMORWAY, where he serves as the chief of research. Prof. Tambe received his Ph.D. from the School of Computer Science at Carnegie Mellon University.

References

[1] Amato, C., Konidaris, G.D., Kaelbling, L.P.: Planning with macro-actions in decentralized pomdps. In: Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems, pp. 1273–1280. International Foundation for Autonomous Agents and Multiagent Systems (2014)

[2] Amato, C., Oliehoek, F.A.: Scalable planning and learning for multiagent pomdps. Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI-15) (to appear) (2015)

[3] An, B., Brown, M., Vorobeychik, Y., Tambe, M.: Security games with surveillance cost and optimal timing of attack execution. In: Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems, pp. 223–230. International Foundation for Autonomous Agents and Multiagent Systems (2013)

[4] Barnhart, C., Johnson, E., Nemhauser, G., Savelsbergh, M., Vance, P.: Branch and price: Column generation for solving huge integer programs. In: Operations Research (1994)

[5] Basilico, N., Gatti, N., Amigoni, F.: Patrolling security games: Definition and algorithms for solving large instances with single patroller and single intruder. Artificial Intelligence **184-185**, 78–123 (2012)

[6] Becker, R., Zilberstein, S., Lesser, V., Goldman, C.V.: Solving transition independent decentralized markov decision processes. In: JAIR, vol. 22, pp. 423–455 (2004)

[7] Bernstein, D.S., Givan, R., Immerman, N., Zilberstein, S.: The complexity of decentralized control of markov decision processes. Mathematics of operations research **27**(4), 819–840 (2002)

[8] Bosansky, B., Kiekintveld, C., Lisy, V., Cermak, J., Pechoucek, M.: Double-oracle algorithm for computing an exact nash equilibrium in zero-sum extensive-form games. In: Proceedings of the 2013 International Conference on Autonomous Agents and Multi-agent Systems, AAMAS '13, pp. 335–342. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC (2013). URL `http://dl.acm.org/citation.cfm?id=2484920.2484975`

[9] Bošanskỳ, B., Lisỳ, V., Jakob, M., Pěchouček, M.: Computing time-dependent policies for patrolling games with mobile targets. In: AAMAS (2011)

[10] Breton, M., Alj, A., Haurie, A.: Sequential stackelberg equilibria in two-person games. Journal of Optimization Theory and Applications **59**(1), 71–97 (1988)

[11] Carter, C.J.: Congressman: Thwarted terror plot targeted train from Canada to U.S. (2013). Retrieved Oct 3, 2013 from http://www.cnn.com/2013/04/22/world/americas/canada-terror-plot-thwarted/index.html

[12] Conitzer, V., Sandholm, T.: Computing the optimal strategy to commit to. In: ACM EC-06, pp. 82–90 (2006)

[13] Dempster, A.P., Laird, N.M., Rubin, D.B.: Maximum likelihood from incomplete data via the em algorithm. Journal of the Royal Statistical Society. Series B (Methodological) pp. 1–38 (1977)

[14] Dibangoye, J.S., Amato, C., Doniec, A.: Scaling up decentralized MDPs through heuristic search. In: UAI, pp. 217–226 (2012)

[15] Dibangoye, J.S., Amato, C., Doniec, A., Charpillet, F.: Producing efficient error-bounded solutions for transition independent decentralized mdps. In: Proceedings of the 2013 International Conference on Autonomous Agents and Multi-agent Systems, pp. 539–546. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC (2013)

[16] Doshi, P., Zeng, Y., Chen, Q.: Graphical models for online solutions to interactive pomdps. In: Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems, p. 217. ACM (2007)

[17] Doshi, P., Zeng, Y., Chen, Q.: Graphical models for interactive pomdps: representations and solutions. Autonomous Agents and Multi-Agent Systems **18**(3), 376–416 (2009)

[18] Fave, F.M.D., Shieh, E., Jain, M., Jiang, A.X., Rosoff, H., Tambe, M., Sullivan, J.P.: Efficient solutions for joint activity based security games: Fast algorithms, results and a field experiment on a transit system. Journal of Autonomous Agents and Multi-Agent Systems (JAAMAS)(to appear) (2014)

[19] Gan, J., An, B., Vorobeychik, Y.: Security games with protection externalities. In: Twenty-Ninth AAAI Conference on Artificial Intelligence (2015)

[20] Gatti, N.: Game theoretical insights in strategic patrolling: Model and algorithm in normal-form. In: ECAI-08, pp. 403–407 (2008)

[21] Gmytrasiewicz, P.J., Doshi, P.: A framework for sequential planning in multi-agent settings. Journal of Artificial Intelligence Research (JAIR) **24**, 49–79 (2005)

[22] Goldman, C.V., Allen, M., Zilberstein, S.: Learning to communicate in a decentralized environment. Autonomous Agents and Multi-Agent Systems **15**(1), 47–90 (2007)

[23] Goldman, C.V., Zilberstein, S.: Communication-based decomposition mechanisms for decentralized mdps. Journal of Artificial Intelligence Research (JAIR) **32**, 169–202 (2008)

[24] Haskell, W.B., Kar, D., Fang, F., Tambe, M., Cheung, S., Denicola, L.E.: Robust protection of fisheries with compass. In: Conference on Innovative Applications of Artificial Intelligence (IAAI) (2014)

[25] INFORMS: Terrorism risk greatest for subway/rail commuters, says MIT paper at INFORMS conference (2012). Retrieved Oct 3, 2013 from https://www.informs.org/About-INFORMS/News-Room/Press-Releases/Terrorism-Rail-Risk

[26] Jain, M., Kardes, E., Kiekintveld, C., Ordóñez, F., Tambe, M.: Security Games with Arbitrary Schedules: A Branch and Price Approach. In: AAAI, pp. 792–797 (2010)

[27] Jain, M., Korzhyk, D., Vanek, O., Conitzer, V., Pechoucek, M., Tambe, M.: A double oracle algorithm for zero-sum security games on graphs. In: AAMAS, vol. 1, pp. 327–334 (2011)

[28] Jain, M., Tsai, J., Pita, J., Kiekintveld, C., Rathi, S., Tambe, M., Ordóñez, F.: Software assistants for randomized patrol planning for the lax airport police and the federal air marshal service. Interfaces **40**(4), 267–290 (2010)

[29] Jakob, M., Vaněk, O., Hrstka, O., Pěchouček, M.: Agents vs. pirates: multi-agent simulation and optimization to fight maritime piracy. In: AAMAS, vol. 1, pp. 37–44 (2012)

[30] Jiang, A.X., Yin, Z., Zhang, C., Tambe, M., Kraus, S.: Game-theoretic randomization for security patrolling with dynamic execution uncertainty. In: AAMAS, vol. 1, pp. 207–214 (2013)

[31] Kiekintveld, C., Jain, M., Tsai, J., Pita, J., Tambe, M., Ordóñez, F.: Computing optimal randomized resource allocations for massive security games. In: AAMAS, vol. 1, pp. 689–696 (2009)

[32] Kumar, A., Zilberstein, S.: Anytime planning for decentralized pomdps using expectation maximization. Proceedings of the Twenty-Sixth Conference on Uncertainty in Artificial Intelligence pp. 294–301 (2010)

[33] Leitmann, G.: On generalized stackelberg strategies. Journal of Optimization Theory and Applications **26**(4), 637–643 (1978)

[34] Melo, F.S., Veloso, M.: Decentralized MDPs with sparse interactions. Artificial Intelligence **175**(11), 1757–1789 (2011)

[35] Messias, J.V., Spaan, M.T., Lima, P.U.: Gsmdps for multi-robot sequential decision-making. In: AAAI (2013)

[36] Nair, R., Tambe, M., Yokoo, M., Pynadath, D., Marsella, S.: Taming decentralized POMDPs: Towards efficient policy computation for multiagent settings. In: IJCAI, vol. 1, pp. 705–711 (2003)

[37] Nair, R., Varakantham, P., Tambe, M., Yokoo, M.: Networked distributed POMDPs: A synthesis of distributed constraint optimization and POMDPs. In: AAAI, vol. 1, pp. 133–139 (2005)

[38] Nudelman, E., Wortman, J., Shoham, Y., Leyton-Brown, K.: Run the gamut: A comprehensive approach to evaluating game-theoretic algorithms. In: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 2, pp. 880–887. IEEE Computer Society (2004)

[39] Oliehoek, F.A., Kooi, J.F., Vlassis, N.: The cross-entropy method for policy search in decentralized POMDPs. Informatica **32**, 341–357 (2008)

[40] Oliehoek, F.A., Spaan, M.T., Amato, C., Whiteson, S.: Incremental clustering and expansion for faster optimal planning in dec-pomdps. Journal of Artificial Intelligence Research pp. 449–509 (2013)

[41] Oliehoek, F.A., Spaan, M.T., Witwicki, S.J.: Influence-optimistic local values for multiagent planning. Proceedings of the Fourteenth International Conference on Autonomous Agents and Multiagent Systems. Extended Abstract. (To appear) (2015)

[42] Paruchuri, P., Pearce, J.P., Marecki, J., Tambe, M., Ordóñez, F., Kraus, S.: Playing games with security: An efficient exact algorithm for Bayesian Stackelberg games. In: AAMAS-08, pp. 895–902 (2008)

[43] Petrik, M., Zilberstein, S.: A bilinear programming approach for multiagent planning. JAIR **35**(1), 235–274 (2009)

[44] Pita, J., Jain, M., Ordonez, F., Tambe, M., Kraus, S.: Robust solutions to stackelberg games: Addressing bounded rationality and limited observations in human cognition. Artificial Intelligence Journal, 174(15):1142-1171 (2010)

[45] Pita, J., Jain, M., Western, C., Portway, C., Tambe, M., Ordonez, F., Kraus, S., Parachuri, P.: Deployed ARMOR protection: The application of a game-theoretic model for security at the Los Angeles International Airport. In: AAMAS, vol. 1, pp. 125–132 (2008)

[46] Qian, Y., Haskell, W.B., Jiang, A.X., Tambe, M.: Online planning for optimal protector strategies in resource conservation games. In: International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2014), vol. 1, pp. 733–740 (2014)

[47] Reuters: Al Qaeda planning attacks on high-speed trains in Europe: newspaper (2013). Retrieved Oct 3, 2013 from http://www.reuters.com/article/2013/08/19/us-germany-security-qaeda-idUSBRE97I0IN20130819

[48] Roth, M., Simmons, R., Veloso, M.: Exploiting factored representations for decentralized execution in multiagent teams. In: Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems, p. 72. ACM (2007)

[49] Shieh, E., An, B., Yang, R., Tambe, M., Baldwin, C., DiRenzo, J., Maule, B., Meyer, G.: PROTECT: A deployed game theoretic system to protect the ports of the united states. In: AAMAS, vol. 1, pp. 13–20 (2012)

[50] Shieh, E., Jain, M., Jiang, A.X., Tambe, M.: Efficiently solving joint activity based security games. In: IJCAI, vol. 1, pp. 346–352 (2013)

[51] Shieh, E., Jiang, A.X., Yadav, A., Varakantham, P., Tambe, M.: Unleashing dec-mdps in security games: Enabling effective defender teamwork. In: European Conference on Artificial Intelligence (ECAI), vol. 1, pp. 819–824 (2014)

[52] Smith, A., Vorobeychik, Y., Letchford, J., Livermore, C.: Multi-defender security games on networks. In: Workshop on Pricing and Incentives in Networks and Systems (2013)

[53] Sonu, E., Doshi, P.: Generalized and bounded policy iteration for finitely-nested interactive pomdps: scaling up. In: AAMAS, vol. 2, pp. 1039–1048 (2012)

[54] Spaan, M.T., Melo, F.S.: Interaction-driven markov games for decentralized multiagent planning under uncertainty. In: AAMAS, vol. 1, pp. 525–532 (2008)

[55] Spaan, M.T., Oliehoek, F.A.: The multiagent decision process toolbox: Software for decision-theoretic planning in multiagent-systems. In: Proceedings of the Joint Conference on Autonomous Agents and Multiagent Systems (2008)

[56] Tambe, M.: Security and Game Theory: Algorithms, Deployed Systems, Lessons Learned. Cambridge University Press (2011)

[57] Vaněk, O., Jakob, M., Lisý, V., Bošanský, B., Pěchouček, M.: Iterative game-theoretic route selection for hostile area transit and patrolling. In: The 10th International Conference on Autonomous Agents and Multiagent Systems, vol. 3, pp. 1273–1274 (2011)

[58] Varakantham, P., Ahmed, A., Cheng, S.F.: Decision support for assorted populations in uncertain and congested environments. In submission to JAIR (2013)

[59] Varakantham, P., Kwak, J., Taylor, M., Marecki, J., Scerri, P., Tambe, M.: Exploiting coordination locales in distributed POMDPs via social model shaping. In: ICAPS, pp. 313–320 (2009)

[60] Varakantham, P., Lau, H.C., Yuan, Z.: Scalable randomized patrolling for securing rapid transit networks. In: IAAI, pp. 1563–1568 (2013)

[61] Von Stengel, B., Zamir, S.: Leadership with commitment to mixed strategies. CDAM Research Report LSE-CDAM-2004-01 (2004)

[62] Wu, F., Zilberstein, S., Jennings, N.R.: Monte-carlo expectation maximization for decentralized pomdps. In: Proceedings of the Twenty-Third international joint conference on Artificial Intelligence, pp. 397–403. AAAI Press (2013)

[63] Yang, R., Ford, B., Tambe, M., Lemieux, A.: Adaptive resource allocation for wildlife protection against illegal poachers. In: International Conference on Autonomous Agents and Multiagent Systems (AAMAS), vol. 1, pp. 453–460 (2014)

[64] Yang, R., Tambe, M., Ordonez, F.: Computing optimal strategy against quantal response in security games. In: AAMAS, vol. 2, pp. 847–854 (2012)

[65] Yin, Z., Jiang, A., Johnson, M., Tambe, M., Kiekintveld, C., Leyton-Brown, K., Sandholm, T., Sullivan, J.: Trusts: Scheduling randomized patrols for fare inspection in transit systems. In: Conference on Innovative Applications of Artificial Intelligence (IAAI) (2012)

[66] Yin, Z., Korzhyk, D., Kiekintveld, C., Conitzer, V., Tambe, M.: Stackelberg vs. Nash in Security Games: Interchangeability, Equivalence, and Uniqueness. In: AAMAS, vol. 1, pp. 1139–1146 (2010)