Not a Lone Ranger: Unleashing Defender Teamwork in Security Games

by

Eric Shieh

A Dissertation Presented to the
FACULTY OF THE USC GRADUATE SCHOOL
UNIVERSITY OF SOUTHERN CALIFORNIA
In Partial Fulfillment of the
Requirements for the Degree
DOCTOR OF PHILOSOPHY
(COMPUTER SCIENCE)

May 2015

Acknowledgments

This Ph.D. is the culmination of critical experiences, conversations, and relationships - in particular, investments (big and small) made by those with whom I've had the privilege of knowing over the past five years.

First, I would like to thank God for guiding me through this whole process. From the beginning, in leading me to USC and to Professor Milind Tambe, to strengthening me in the tougher times of the doctoral program, and finally to its completion.

I would like to thank my advisor, Professor Milind Tambe, for believing in me and accepting me as a PhD student. Thank you for the many opportunities that you provided to me, from working on the PROTECT system with the United States Coast Guard to authorship of papers and collaborations that enabled travel to conferences all over the world. Your care for each advisee is unparalleled, and is evident in the way you foster a tight community of students that truly feels like family. The strength of this network is especially palpable in our annual fall welcoming dinner, the annual spring Teamcore retreat, and even in soccer practices. Thank you for your devotion to research and passion to continually be forward-thinking, and for stressing use-inspired research. It truly has been a blessing to be advised by you.

I would like to thank my committee members for their feedback and guidance in my research: Andrea Armani, Morteza Dehghani, Rahul Jain, and Cyrus Shahabi. Thank you, Christopher

Kiekintveld, for not only serving on my qualifying committee, but also for providing input and suggestions throughout my doctoral career.

I would like to thank the United States Coast Guard personnel that I had the fortune of working with to make the PROTECT application a success. I would like to thank Craig Baldwin, for your tireless energy devoted to not just making the PROTECT application a success at Boston, but continuing it at the port of New York, and then even pushing it to get deployed nationwide. Thank you, Joseph DiRenzo, David Boyd, and Ben Maule, for being a champion of PROTECT and helping to explain the game theoretic aspects to other USCG personnel.

During my time at USC, I had the privilege of collaborating with many excellent researchers: Bo An, Albert Xin Jiang, Francesco Maria Delle Fave, Pradeep Varakantham, Rong Yang, Manish Jain, Amulya Yadav, Matthew Brown, Chao Zhang, Zhengyu Yin, Fernando Ordonez, Heather Rosoff, Meritxell Vinyals, Jesus Cerquides, and Juan Antonio Rodriguez-Aguilar.

I would like to thank the entire Teamcore family: Matthew Brown (for being my fellow lab mate since the beginning, including the time our office was moved to the storage room), Bo An, James Pita, Manish Jain (for being patient with me and guiding me through this PhD process), Jason Tsai, Jun-young Kwak (for having such a kind and generous heart, and your USB-fan), Zhengyu Yin, Rong Yang (for being a great office mate and help), Thanh Nguyen (for your constant presence in the office and many emoticons), Leandro Marcolino (for being a great friend and providing memorable experiences at conferences), Fei Fang, Albert Xin Jiang, Yundi Qian (for putting up with me practicing my limited Chinese), Francesco Delle Fave, Debarun Kar, Benjamin Ford, Haifeng Xu, Amulya Yadav (for your chill and friendly attitude, and also your help in running experiments), William Haskell, Aaron Schlenker, Sara McCarthy, Yasi Abbasi, and Shahrzad Gholami.

I would like to thank the entire Church of Southland community, especially my cell brothers for supporting and praying for me. Seong Hwang, Chris Chew, Jonny Whang, Perry Lew, and Pat Kim: Your encouragement and sporadic check-ins always brought a smile to my face.

I would like to thank my parents, Ching-Chyuan Shieh and Miaw-Meei Shieh, for always pushing me and believing the best in me. Thank you for your support, guidance, words of wisdom, and especially your daily prayers for me. I would like to thank my brothers, Angell Shieh and Luke Shieh, and also my sister-in-law, Judy Shieh, for your support and prayers for me.

I would like to thank my parent-in-laws, Vek Huong Taing and Samoeun Taing, for allowing me to stay at your home (at virtually no cost!) for the duration of my PhD career. Thank you also for your constant prayers for me. I also want to thank my brother and sister-in-law, Brian Taing and Nyka Taing, for your generosity, for providing meals when deadlines approach, and for giving me many Yogurtland gift cards.

Lastly, I would like to give a BIG thanks to my beautiful wife, Linda Taing Shieh, for being my rock throughout this whole PhD process. Without you, I would not have been able to complete this program. Thank you for being a constant source of encouragement and for speaking words of truth into my life. I love you, aizhen.

Table of Contents

List of Figures

List of Tables

Abstract

Game theory has become an important research area in handling complex security resource allocation and patrolling problems. Stackelberg Security Games (SSGs) have been used in modeling these types of problems via a defender and an attacker(s). Despite recent successful real-world deployments of SSGs, scale-up to handle defender teamwork remains a fundamental challenge in this field. The latest techniques do not scale-up to domains where multiple defenders must coordinate time-dependent joint activities. To address this challenge, my thesis presents algorithms for solving defender teamwork in SSGs in two phases. As a first step, I focus on domains without execution uncertainty, in modeling and solving SSGs that incorporate teamwork among defender resources via three novel features: (i) a column-generation approach that uses an ordered network of nodes (determined by solving the traveling salesman problem) to generate individual defender strategies; (ii) exploitation of iterative reward shaping of multiple coordinating defender units to generate coordinated strategies; (iii) generation of tighter upper-bounds for pruning by solving security games that only abide by key scheduling constraints.

In the second stage of my thesis, I address execution uncertainty among defender resources that arises from the real world by integrating the powerful teamwork mechanisms offered by decentralized Markov Decision Problems (Dec-MDPs) into security games. My thesis offers the following novel contributions: (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 resources to handle real-world scenarios; (v) Exploration of the robustness of randomized pure strategies. Different mechanisms, from both solving situations with and without execution uncertainty, may be used depending on the features of the domain. This thesis opens the door to a powerful combination of previous work in multiagent systems on teamwork and security games.

Chapter 1: Introduction

The challenge of providing security applies to many domains across the world. One large focus has been on the protection of critical infrastructure and facilities. Some examples of the impact of terrorist attacks include the Oklahoma City bombing in 1995 which killed 168 people and caused at least an estimated $652 million in damages [Hewitt, 2003], the September 11 attacks on the World Trade Center towers and the Pentagon which killed 2,996 people and resulted in an estimated loss of over $100 billion [Institute for the Analysis of Global Security, 2004]. In the 2004 Madrid train bombings, 191 people were killed and approximately 1,800 people were injured along with disrupting the train system [Wikipedia, 2014]. The 2008 Mumbai terrorist attacks on several buildings and sites killed 164 people while injuring 308 people [Press Information Bureau (Government of India), 2008]. On April 15, 2013, two homemade bombs exploded during the Boston Marathon which killed 3 people and injured approximately 264 people [Kotz, 2013]. Even more recently was the Charlie Hebdo shooting on January 2015 where two gunmen killed 12 people which included Charlie Hebdo employees and 2 national police officers while wounding 11 others [BBC News Europe, 2015]. A common theme across all these settings is the limited amount of resources that are unable to protect all areas or targets. An additional factor that must

be accounted for in these security settings is the presence of an adversarial agent that is able to conduct surveillance of the resources' strategy, while having the ability to exploit this knowledge.

Stackelberg Security Games (SSGs) have been widely applied to real-world security domains with these applications depending on significant advances in fast algorithms for SSGs [Jain et al., 2010b; Tambe, 2011]. These applications include ARMOR (Assistant for Randomized Monitoring Over Routes) software at the Los Angeles International Airport [Pita et al., 2008] to randomize checkpoints and canine units, the IRIS (Intelligent Randomization in Scheduling) system used by the United States Federal Air Marshal Service to allocate air marshals on flights [Tsai et al., 2009], the PROTECT (Port Resilience Operational / Tactical Enforcement to Combat Terrorism) application used by the United States Coast Guard to schedule patrols to protect the ports [Shieh et al., 2012], the TRUSTS (Tactical Randomization for Urban Security in Transit Systems) tool to produce patrol schedules in transit systems [Yin et al., 2012], the ARMOR-FISH system that aids the United States Coast Guard in depending fisheries from illegal fishing in the Gulf of Mexico [Brown et al., 2014a], and the PAWS (Protection Assistant for Wildlife Security) algorithm to help wildlife rangers execute patrols to deter poaching in national parks in Uganda [Yang et al., 2014].

SSGs focus on modeling the strategic interactions between a defender (e.g., security personnel) and attacker (e.g., terrorist) where the defender has a limited amount of resources to protect a set of targets. The challenge addressed in SSGs is optimizing the use of a defender's limited security resources in the presence of an adversary who can conduct surveillance before planning an attack.

(a) US Coast Guard patrolling the port of Boston

(b) Sheriff in a transit system

Figure 1.1: Different domains of Stackelberg security games

1.1 Problem Addressed

In solving SSGs, the challenge is how to optimally allocate limited security resources over a set of potential targets [Basilico et al., 2009; Conitzer and Sandholm, 2006; Paruchuri et al., 2008]. Current algorithms solve this problem by generating mixed defender strategies, which represents a probability distribution over a set of targets or patrol schedules, while also considering that the attacker conducts surveillance over the defender's strategy.

An example of a deployed application based on SSGs is the PROTECT system which has been in use by the United States Coast Guard (USCG) since 2011 to generate patrol schedules to help protect the ports from terrorist attacks [Shieh et al., 2012]. While working on this application with the USCG, one concern that they expressed was that the system did not model coordination across multiple boats and/or resources. Modeling and solving security games that handle coordination among multiple defender units significantly increases the number of possible defender strategies.

This results in an exponentially large defender strategy space that for even small scale games are unable to fit into memory.

Such coordination is an important aspect of real-world security systems as there are benefits that may accrue from coordination across multiple defender resources, e.g., if a target is visited by a single defender resource, it may only be 50% effective in detecting (and hence stopping) a potential attack. The arrival of a second defender resource may increase the effectiveness to 80% (as the attacker may be forced to react to a second defender). Previous algorithms would typically consider a target fully covered (or 100% effective) if a single defender resource visits a target and thus do not handle varying effectiveness nor additional benefit if a second resource also visits a target. No prior SSG algorithms can scale-up to handle coordinated patrols for real-world domains; neither general purpose Stackelberg algorithms such as [Conitzer and Sandholm, 2006; Paruchuri et al., 2008], nor special purpose SSG algorithms [Jain et al., 2010b].

In addition to coordination among defender resources, the challenge of deploying game-theoretic schedules in the field has not been addressed by research in SSGs. Despite some initial evaluation of the PROTECT system [Shieh et al., 2012], a head-to-head comparison between game-theoretic schedules and human generated schedules, the way in which most security agencies allocate their resources, is still missing from literature. However, this type of study would be extremely useful to advance the state-of-the-art of game-theoretic scheduling, because it would allow the user to measure, for the first time, the actual performance of such schedules when deployed in the real world.

1.2 Contributions

My thesis will focus on modeling and efficiently solving teamwork among defender resources in security games for real-world domains.

1.2.1 Defender Teamwork in Security Games with No Uncertainty

The first part of my thesis will first address teamwork in SSGs for defender resources in domains with no execution uncertainty while also providing valuable real-world feedback that compares human generated schedules versus game-theoretic generated schedules. To handle teamwork among defender resources in SSGs without uncertainty, my thesis presents SMART, *Security games with Multiple coordinated Activities and Resources that are Time-dependent*, a model extending the framework of security games to *explicitly represent* jointly coordinated activities, and two algorithms: an optimal algorithm, SMART$_O$, that computes optimal defender strategies for SMART problems, and a heuristic algorithm, SMART$_H$, that achieves further speed-up over SMART$_O$ [Shieh et al., 2013].

The algorithms build upon work that has leveraged the branch-and-price framework [Jain et al., 2010b]. These algorithms are able to exploit the structure of the joint activity coordination problem to gain speed up based on the following key ideas: (i) use of insights from the Traveling Salesman Problem to order the search space, especially in SMART$_H$, while maintaining coordination, (ii) efficient greedy computation of patrols per resource via iterative reward shaping to generate a joint patrol, and (iii) generation of tight upper-bounds exploiting scheduling constraints to allow pruning of the search space based on the submodular property of joint activities.

This part of the thesis also presents an important head-to-head comparison of the game-theoretic schedules generated by SMART$_H$ versus schedules generated by humans in a large scale real-world experiments of a one-day patrol exercise of the Los Angeles Metro System. The results show that the game-theoretic schedules from SMART$_H$ were evaluated to outperform the schedules generated by humans.

1.2.2 Defender Teamwork in Security Games with Execution Uncertainty

The second part of my thesis presents scalable solution approaches for handling the presence of execution uncertainty of coordinating defender resources (which significantly increases the complexity of the problem) [Shieh et al., 2014]. Teamwork can be complicated by three factors—(i) requiring defender resources to coordinate under uncertainty; (ii) handling the dynamic inability of a resource to continue teamwork; and (iii) lack of communication. The next part of my thesis addresses these additional factors of uncertainty that frequently arise in the real-world in security games. I leverage previous work in teamwork in multiagent systems such as the TREMOR algorithm which solves distributed Partially Observable Markov Decision Problems (DEC-POMDPs) for a team of agents that work together in environments with uncertainty [Varakantham et al., 2009]. First, I provide a new model of a security game where a joint policy is used as the defender's pure strategy to handle coordination under uncertainty. Second, I present a new algorithm that uses column generation to efficiently generate Dec-MDP policies as pure strategies used in determining the optimal mixed strategy for the defender team. Third, global events among defender resources are modeled and leveraged in handling teamwork. Fourth, I show multiple heuristics that help scale-up to real-world scenarios.

1.3 Overview of Thesis

This thesis is organized in the following manner. Chapter 2 discusses the necessary background materials for the research presented here. Chapter 3 provides an overview of the relevant research. Chapter 4 presents the SMART model and corresponding algorithms to solve these types of games. Chapter 5 describes the algorithm and heuristics used to solve coordinating defender resources in SSGs in the presence of execution uncertainty via the use of Dec-MDPs in security games. Chapter 6 provides additional evaluation and analysis into the performance of the algorithms presented in this thesis. Chapter 7 concludes this thesis and presents ideas for future work.

Chapter 2: Background

In this chapter, I present two areas of research that the thesis draws upon. I begin by introducing the Stackelberg Security Game model in Section 2.1 as the work in this thesis extends the security game model. The second research area is Decentralized Markov Decision Problems that explore multiple distributed cooperating agents, which is leveraged in providing teamwork under uncertainty and is described in more detail in Section 2.2.

2.1 Stackelberg Security Games

Stackelberg Security Games (SSGs) [Conitzer and Sandholm, 2006; Kiekintveld et al., 2009] are composed of two players, a leader and a follower, where the leader (denoted as the defender) must protect a set of targets from the follower (denoted as the attacker or adversary). The defender has a limited amount of resources with which to protect the set of targets against the adversary whose strategy is to determine which target to attack. The adversary conducts surveillance and thus learns the defender's strategy before choosing the target to attack.

Both the attacker and defender have pure strategies where the defender's pure strategy would be an allocation of resources on patrols or targets, while the adversary's pure strategy would be the target that is to be attacked. The optimal strategy of resource allocation for the defender will

	U_d^u	U_d^c	U_a^u	U_a^c
t_1	-5	4	6	-3
t_2	-3	1	7	-2

Table 2.1: Sample payoffs for the defender and attacker.

typically be a mixed strategy, which is a probability distribution over the set of defender pure strategies.

For domains where the defender resources must conduct patrols over a set of targets while adhering to a maximum patrol time, the defender's pure strategy is now defined on a graph $G_r = (T, E_r)$, where the vertices T are the targets and the edges E_r represent connectivity between the targets for resource r. This allows for heterogeneous resources, e.g., boats or helicopters, which have the same targets but the connectivity between nodes can be different. For each $e \in E_r$, $\tau(e)$ represents the time it takes one defender resource to traverse the edge e. As usual with SSGs [Yin et al., 2010], for each target $t \in T$, there is an associated reward $U_d^c(t)$ and penalty $U_d^u(t)$ to the defender if t was protected with an *effectiveness* of 100% and 0% respectively. Similarly, payoffs $U_a^c(t)$ and $U_a^u(t)$ are defined for the attacker, with $U_d^u(t) < U_d^c(t)$ and $U_a^c(t) < U_a^u(t)$. Table 2.1 gives an example of the payoffs for two targets, t_1 and t_2, for the defender and the attacker. If the defender protects target t_1, then the defender's payoff would be $U_d^c(t_1) = 4$ while the attacker's payoff would be $U_a^c(t_1) = -3$. However, if the defender leaves target t_2 unprotected, then the defender's payoff would be $U_d^u(t_2) = -3$ while the attacker's payoff would be $U_a^u(t_2) = 7$. The defender has a set of R resources, and each resource can choose an activity from the set $\mathcal{A} = \{\alpha_1, \alpha_2, \ldots \alpha_K\}$. Each pure strategy of the defender is a route for each resource.

There have been many algorithms and models developed to solve SSGs, including DOBSS [Paruchuri et al., 2008] which solves SSGs using a mixed-integer linear program, ASPEN [Jain et al., 2010b] which solves SSGs that contain a greater number of defender resources

and larger strategy space, ORIGAMI [Kiekintveld et al., 2009] which provides a polynomial time algorithm for SSGs that contain no scheduling constraints, along with HUNTER [Yin and Tambe, 2012] and RECON [Yin et al., 2011] which compute robust strategies for security games. However, these algorithms are unable to scale-up in solving security games that require teamwork (in the form of joint coordinated activities) among defender resources.

2.2 Dec-MDP

Teamwork among multiple agents has been a well studied area. Some well known frameworks for teamwork include the SharedPlans [Grosz and Kraus, 1996] framework where agents communicate their beliefs, desires, and intentions, the STEAM [Tambe, 1997] framework that builds a partial hierarchy of joint intentions/actions, and the Generalized Partial Global Planning (GPGP) [Decker and Lesser, 1995] framework that is based on a group of coordinating mechanisms to handle different task environments. However, these frameworks of teamwork focus on creating a standard methodology for coordination and communication while failing to address uncertainty and rewards.

Decentralized Markov Decision Problems (Dec-MDP) models provide a framework to handle distributed decision-making problems to generate coordinated multiagent policies where uncertainty exists in the domain while also accounting for rewards and costs associated with different states [Bernstein et al., 2002]. The Dec-MDP model that is used by this thesis (for teamwork under uncertainty) is defined by the tuple: $\langle Ag, S, A, T, R \rangle$. $Ag = \{1, \ldots, n\}$ represents the set of n defender resources. $S = S_u \times S_1 \times \cdots \times S_n$ is a finite set of world states of the form $s = \langle s_u, s_1, \cdots, s_n \rangle$. Each resource 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 resource i reaches target t_i. Time is discretized and there are m decision epochs

$\{1, \ldots, m\}$. s_u is the *unaffected state*, meaning that it is not affected by the resources' actions. It is employed to represent occurrence of global events (bomb threats, increased risk at a location etc.)

$A = A_1 \times \cdots \times A_n$ 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 resource i. $T : S \times A \times S \rightarrow \mathbb{R}$ 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 joint action is a. Since transitions between resource i's local states are independent of actions of other resources, we have transition independence. Due to the presence of unaffected states, this notion of transition independence is equivalent to the one employed in Network Distributed POMDPs [Nair et al., 2005]. Formally, $T(s, a, s') = T_u(s_u, s'_u) \cdot \prod_i T_i(\langle s_u, s_i \rangle, a_i, s'_i)$.

My thesis focuses on modeling game-theoretic interactions, in which the rewards depend on the strategies of both the defender and the attacker. Therefore standard Dec-MDP reward functions cannot be directly applied. Nevertheless, as part of our algorithm, I will reduce a subproblem to a Dec-MDP problem with a standard Dec-MDP joint reward function of the form $R : S \rightarrow \mathbb{R}$, where $R(s)$ represents the reward for reaching joint state s. Unlike in the ND-POMDP framework, our reward function is not decomposable.

Dec-MDPs are a popular framework for multiagent planning and coordination under uncertainty, with work ranging from a simplified model for transition independent Dec-MDPs [Becker et al., 2004], a toolbox for multiagent planning solvers [Spaan and Oliehoek, 2008], the use of heuristic search and constraint optimization [Dibangoye et al., 2012], to multi-robot exploration [Matignon et al., 2012]. A major difference in this thesis is the addition of an adversarial agent that is able to respond to the joint policy of the Dec-MDP.

Chapter 3: Related Work

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

3.1 Stackelberg Security Games

Stackelberg security games (SSGs) have gathered significant attention in literature [Basilico et al., 2009; Dickerson et al., 2010; Korzhyk et al., 2011a,b; Letchford and Conitzer, 2013; Letchford et al., 2012; Letchford and Vorobeychik, 2013]. Early work focused on solving Stackelberg games but did not consider their application to the security domain. Stackelberg games were first solved via the MultipleLP approach [Conitzer and Sandholm, 2006] where the leader's strategies were computed for a Stackelberg game. A faster algorithm (compared to the MultipleLP approach) known as DOBSS [Paruchuri et al., 2008] used a mixed-integer linear program to solve for the leader's optimal strategy for Stackelberg games.

Following DOBSS, ORIGAMI and ERASER algorithms [Kiekintveld et al., 2009] 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 [Kiekintveld et al., 2009] provided a more compact representation of the defender strategies for multiple resources (compared to DOBSS) while also handling scheduling constraints that arise from the defender's strategies. ASPEN [Jain et al., 2010b] 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, thereby significantly improving the efficiency of solving Stackelberg security games. However, these algorithms do not consider teamwork and joint activities among the defender resources.

As discussed earlier, SSG models and algorithms have been used to build decision aids including ARMOR [Pita et al., 2008], IRIS [Tsai et al., 2009], GUARDS [Pita et al., 2011], PROTECT [Shieh et al., 2012], TRUSTS [Yin et al., 2012] and RaPtoR [Varakantham et al., 2013b]. All these decision aids have been developed to assist the security of transportation infrastructure including ports, airports and train lines. However, most of these decision aids do not model joint coordinated activities. The IRIS system models some of this coordination by assigning a negative infinite weight to the joint action of two Federal Air Marshals (FAMS) taking the same flight, explicitly restricting the maximum number of FAMS on any flight to one (see the work of Jain et al. [2010a] for more details). This type of solution, however, does not model more complex forms of joint effectiveness as we will do in this thesis. In fact, most decision aids, including ARMOR, GUARDS, PROTECT, TRUSTS and RaPtoR, do not account for jointly coordinated activities. They allocate security resources without considering the benefits that could be accrued by different resources combining their effort. As a consequence, they are generating schedules that are not as effective as they could be.

Jiang et al. [Jiang et al., 2013c] 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 resources and instead focuses on a single defender, or multiple independent defender resources that do not consider the additional effectiveness that arises from having multiple resources at the same target.

Other than execution uncertainty, there has been recent work investigating coordination in security games. In, Jiang et al. [2013b], the impact and loss from miscoordination between defender resources is analyzed and quantified. Additional work has explored coordination mechanisms for defenders that desire to share only limited amounts of sensitive information in the context of security games [Procaccia et al.]. However, they do not consider and address the optimization challenges and modeling of coordination in addition to execution uncertainty.

In addition to decision aids and security allocation, research in SSGs has also addressed problems of multi-robot patrolling. More specifically, research has developed the multi-robot adversarial patrolling games (MAPG) framework, a restricted type of SSG, which considers the problem of coordinated patrols of multiple robots around a closed area with the existence of an adversary attempting to penetrate into the area [Agmon et al., 2008a, 2011]. The penetration requires time and the defender should identify the attacker during his attempt. Similarly, the work from Sless et al. [2014] requires the robots to physically inspect penetration attempts for a given time period. More specifically, Sless et al. [2014] investigate the problem of coordinated attacks, in which the adversary initiates two attacks in order to maximize its chances of successful penetration, assuming a robot from the team will be sent to examine a penetration attempt. Such MAPGs patrols are frequency-based patrols in adversarial environments, but do not consider targets of

varying importance and the impact of *joint activities* [Agmon et al., 2008b; Machado et al., 2003; Vanek et al., 2010].

3.2 Decentralized Markov Decision Process

There has been significant amount of work done on decentralized Markov Decision Problems. I first present models and extensions of Dec-MDPs. I then explore two subareas including communication in Dec-MDPs and interactive sequential decision making under uncertainty. Then the section describes some of the algorithms developed to efficiently solve Dec-MDPs including the Expectation-Maximization framework.

One of the earliest models of Dec-MDPs addressed the issue of scalability when there exists transition independence among the agents [Becker et al., 2004]. 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 [Nair et al., 2005]. Another extension of Dec-MDPs include a decision theoretic model known as the decentralized sparse-interaction Markov decision process (Dec-SIMDP) [Melo and Veloso, 2011], a subclass of Dec-MDPs, where local interactions and communications are abstracted to interaction areas and observable interactions. Unfortunately, the complexity of Dec-MDPs have been shown to be NEXP-complete [Bernstein et al., 2002]. These models provide a framework for coordination among agents, however are unable to model an adversarial agent that has a different reward function.

Another subarea of Dec-MDPs include communication among agents in a decision theoretic, decentralized environment by Goldman et al. [Goldman et al., 2007]. 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 [Goldman and Zilberstein, 2008] 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. Dec-MDPs have also been used to address the issue of multi-robot exploration under communication breakdowns [Matignon et al., 2012]. Another area of research related to sequential decision making under uncertainty is interactive partially observable Markov decision processes (I-POMDP) [Gmytrasiewicz and Doshi, 2005], where the beliefs of the agents are not constrained just by the state space, but include the physical environment and models of other agents. An initial technique used to solve I-POMDPs included the use of particle filters to obtain approximate solutions [Doshi and Gmytrasiewicz, 2005]. Additional research has focused on graphical models to represent and solve I-POMDPs [Doshi et al., 2007, 2009] along with a policy iteration algorithm to solve I-POMDPs [Sonu and Doshi, 2012]. These models and algorithms explore the coordination aspect under uncertainty but again fail to account for strategic agents with a different reward structure.

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 resources are a local phenomenon, and provides a fast approximate solution that exploits the structure [Spaan and Melo, 2008]. Roth et al. [Roth et al., 2007] explored solving multi-agent domains with collective observability where factored policy

representations are used. Dibangoye et al. [Dibangoye et al., 2012], 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. A general framework that has been used to solve multiagent planning problems is the Expectation-Maximization (EM) framework [Dempster et al., 1977], 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 [Kumar and Zilberstein, 2010]. The EM algorithm has also been used in solving infinite horizon POMDP and Dec-POMDP problems that are represented as finite state controllers [Pajarinen and Peltonen, 2011]. Kumar et al. [Kumar et al., 2011] use value factorization within the EM framework where the inference process is able to be separated into smaller components providing greater scalability. While these techniques are useful in solving larger problems of Dec-MDPs, they are unable to handle a game theoretic model that incorporates an adversarial agent.

Chapter 4: Teamwork for Defender Resources without Execution Uncertainty

Despite significant advances in SSGs as mentioned earlier in the thesis, scaling up remains a significant issue in advancing the scope of SSGs. A major drawback of the current algorithms is their failure to scale up to SSGs where multiple defender resources explicitly perform joint activities, i.e., games where coordination in space and time will provide the defender with additional benefits [Jain et al., 2010a; Paruchuri et al., 2008; Vanek et al., 2011]. To date, neither general purpose SSG algorithms [Conitzer and Sandholm, 2006; Paruchuri et al., 2008], nor special purpose SSG algorithms [Jain et al., 2010a] can scale up to handle these joint activities. Yet, joint activities are an important aspect of real-world security. For example, the algorithm used in PROTECT [Shieh et al., 2012] only focuses on one boat patrols. Yet, if a single boat is, perhaps, 50% effective in detecting (and hence stopping) a potential adversary attack, a second boat may increase the effectiveness significantly to, perhaps, 80% as the adversary may be forced to react to this second boat. PROTECT is unable to handle such coordination over multiple boats. Similarly, when patrolling a train line, security resources such as explosive detective canine (EK9) teams often patrol train lines in cooperation with other resources. By doing so, their effectiveness is increased. In essence, the key problem for most of the algorithms, discussed above, is that representing this

type of joint activity space significantly accelerates the growth in the number of pure strategies, which severely impacts their ability to scale up in several security domains (e.g., port security).

Furthermore, a key question raised for deployed applications of SSGs is the evaluation of their performance in the field. Despite earlier attempts, the actual evaluation of the deployed SSGs-applications in the field is still a major open challenge [Shieh et al., 2012]. A significant number of practical constraints (e.g., time to train the officers, availability of personnel to organize, run and evaluate the experiment) limits the ability of researchers to conduct head-to-head comparisons between SSGs-applications and human schedulers. Hence, a systematic study that evaluates the benefits using a head-to-head comparison is still missing from the literature.

To address these shortcomings, this chapter presents four contributions. The first contribution is SMART, *Security games with Multiple coordinated Activities and Resources that are Time-dependent*, a model extending the framework of security games to *explicitly represent* jointly coordinated activities. The second contribution consists of two algorithms. I present SMART$_O$, an optimal algorithm to compute optimal defender strategies for SMART problems and SMART$_H$, a heuristic iterative procedure to achieve further speed-up over SMART$_O$. Both SMART$_O$ and SMART$_H$ use a branch-and-price algorithm – an algorithm composed of branch-and-bound and column generation – to deal with the large strategy space of the domain [Barnhart et al., 1994]. These algorithms exploit the structure of the joint coordinated activities to gain speed up, based on the following key ideas: (i) use of insights from the Traveling Salesman Problem (TSP) to order the search space during column generation, especially in SMART$_H$, while maintaining coordination; (ii) efficient greedy computation of patrols per resource via iterative modification of rewards to generate a joint patrol, during column generation, and (iii) generation of tight upper-bounds within the branch-and-bound component by exploiting scheduling constraints to allow pruning of the

search space based on the sub-modular property of joint activities. The third contribution is the analysis of the performance of both SMART$_O$ and SMART$_H$ in solving instances of SMART. I analyze the quality of the solutions generated by SMART$_H$ and evaluate both algorithms in simulation comparing their runtime and solution quality.

Finally, the fourth contribution is the real-world evaluation of the game-theoretic schedules generated using SMART$_H$. This evaluation constitutes the largest scale experiment evaluating the performance of SSGs in the field. I present results from a massive transit full-scale exercise (FSE), a real-world experiment whereby 80 security officers coordinated their activities to patrol 10 stations of a metro line for 12 hours. The purpose of the exercise was a head-to-head comparison between SSG-based schedules, generated using SMART$_H$, against human-generated schedules. We were able to evaluate the schedule generation process, as well as provide a thorough evaluation of the performance of both schedules as conducted by a number of security experts located at each of the ten stations during the entire length of the exercise. The results show that the game-theoretic approach, based on SMART$_H$, was able to significantly cut the schedule generation effort for humans compared to manual scheduling. Yet, game-theoretic scheduling was able to generate schedules similar to the ones generated by humans in terms of number of joint activities. In addition, game-theoretic schedules were able to address the *comprehensive* security of the train line by having the different teams patrol *all* the different levels of the stations (e.g., the platform level, the street level and the mezzaning level). Finally, the game-theoretic schedules allocated the more effective teams to the more important stations. These last two factors, which were missing from the human-generated schedules, led security experts to concur that the game-theoretic schedules were more effective in providing security than the human-generated schedules.

The overall conclusion from this real-world exercise is that the game-theoretic schedules, generated by SMART$_H$ were able to perform at least equivalently to (and in fact better than those) generated by human schedulers. This indicates that we could save precious time so security experts could focus on maintaining security rather than on generating schedules. Overall, the data that was collected constitutes a source of information which can be used for evaluating the current status of research in SSGs, and to understand new directions where to take such research.

4.1 The SMART Problem

A SMART problem is an instance of a SSG. A SSG, as discussed in detail in the work of Kiekintveld et al. [2009], is a two-player game involving a defender d and an attacker a competing over a set of targets T. The defender has a limited number of resources R and needs to select which targets to protect considering that the attacker is going to conduct a thorough surveillance to exploit any predictable pattern in the defender resource allocation. In a SSG, each target $t \in T$ is assigned a reward $U_d^c(t)$ and a penalty $U_d^u(t)$ to the defender if t is covered and uncovered, respectively, by a defender's resource. Similarly, each target is assigned a reward $U_a^c(t)$ and a penalty $U_a^u(t)$ to the attacker. As discussed by Kiekintveld et al. [2009], the payoffs are defined such that $U_d^u(t) < U_d^c(t)$ and $U_a^c(t) < U_a^u(t)\ \forall\ t \in T$. The purpose of each player then is to maximize their expected payoffs defined in equations 4.4 and 4.5. In an optimal solution of a SSG, the defender plays a mixed strategy, i.e., a probability distribution over the different targets, which intelligently allocate the defender resources given the importance of each target and considering the behavior of the attacker [Conitzer and Sandholm, 2006].

In a SMART problem instance, each resource chooses an activity from the set $\mathcal{A} = \{\alpha_1, \alpha_2, \ldots \alpha_K\}$ for each target $t \in T$. Each resource $r \in R$ is assigned a graph $G_r = (T, E_r)$,

R	Number of defender resources, subscripted by r
$G_r = (T, E_r)$	Graph of the input problem instance
T	Set of targets
t_b	Home base
$E_r : \{e(t_i, t_j)\}$	Set of edges
$\tau(e(t_i, t_j))$	Time required to traverse the edge e
$\tau(\alpha)$	Time required to conduct activity α
$\texttt{eff}(\alpha)$	Effectiveness of activity α
$\texttt{eff}(\alpha_i, \alpha_j)$	Effectiveness of joint activity $\langle \alpha_i, \alpha_j \rangle$
$\mathbf{P}$	Set of pure strategies of the defender
$\omega_t(\mathbf{P}_i)$	Effective coverage of t in $\mathbf{P}_i$
Γ_r	Maximum time allowed for an individual patrol for resource r
W	Time window for a joint activity
$\mathcal{X}^r$	The set of pure strategies for resource r

Table 4.1: Notation Table

where the set of vertices T represents the set of targets to patrol and the set of edges E_r represents the connectivity between such targets. Each edge $e \in E_r$ is assigned a time value $\tau(e)$ representing the time that it takes for one defender resource r to traverse e. Each graph encodes the motion of different resources. For example, security assistants patrol the stations of a train line by taking the trains. In contrast, sheriffs and EK9 units use a car. They can go from one end to the other of a train line without having to cross each station. Similarly, aerial patrols can move freely and reach any area of a port, whereas boat patrols might be constrained to take certain routes. The notation used in Smart is described in Table 4.1.

The attacker's pure strategy space is the set of all targets, T. A pure strategy for the defender is a set of routes, one route X_i for each resource. Formally, each patrol route is defined as an ordered list of 3-tuples $X_i = [X_i^1, \ldots, X_i^j, \ldots]$. The j^{th} 3-tuple $X_i^j = (t, \alpha, \gamma)$ represents a time-action segment for defender resource i: she conducts and completes activity α at target t at time γ. Each time-action segment is different since different activities might require different amounts of time and have a different effect on the target to protect (as discussed below).

Each patrol route starts and ends at the same, pre-defined, home base $t_b \in T$, i.e., $X_i^1.t = t_b$ and $X_i^{|X_i|}.t = t_b$. The total route length of each resource's patrol is upper bounded by a specific value Γ_r as follows:

$$\overbrace{\sum_{j=1}^{|X_i|-1} \tau(e(X_i^j.t, X_i^{j+1}.t))}^{\text{traversal time}} + \overbrace{\sum_{j=1}^{|X_i|} \tau(X_i^j.\alpha)}^{\text{time for activities}} \leq \Gamma_r \; \forall \; X_i \tag{4.1}$$

$\mathcal{X}^r$ is defined as the set of pure strategies for resource r and the set of joint pure strategies $\mathbf{P}$ is given by the cross-product of pure strategies for each resource, i.e., $\mathbf{P} = \prod_{r=1}^{R} \{\mathcal{X}^r\}$.

SMART is unique since it explicitly models *joint activities*, or activities coordinated in space and time between multiple defender resources. The defender is said to conduct a joint activity $\langle \alpha_i, \alpha_j \rangle$ in her pure strategy if there exists at least two tuples $(t_i, \alpha_i, \gamma_i) \in X_i$ and $(t_j, \alpha_j, \gamma_j) \in X_j$ in the defender's pure strategy such that $t_i = t_j$ and $|\gamma_i - \gamma_j| \leq W$. In other words, i.e., the two activities are on the same target and are within a time window of width W. Here, the time width W represents the minimum interval of time within which two different activities have a joint effect. For instance, if one aerial and one boat patrol explore the same area one after the other within a time frame of 10 minutes, their effectiveness will be much larger then if they were patrolling one after the other but within a time frame of 30 minutes. In the former case, it can be assumed that they were conducting a joint patrol action. In contrast, in the second case, given the large temporal distance, the two actions can be considered individually.

For each activity α_i, $\texttt{eff}(\alpha_i)$ represents the individual effectiveness[1] of the activity α_i. This effectiveness ranges from 0% to 100%, and measures the probability that the defender will be able to successfully prevent an attack on target t if such an attack overlaps with the activity α_i at t that the defender is conducting . This is similar to what was done in PROTECT [Shieh et al., 2012]. We define the effectiveness of the joint activity $\langle \alpha_i, \alpha_j \rangle$ as $\texttt{eff}(\alpha_i, \alpha_j)$. While a joint activity may be composed of two *or more* resources – and our experimental results show the benefits of n-ary joint activities in Section 4.4.1 – in this section we focus on joint activities composed of two resources for simplicity of explanation. In this case, a joint activity composed of two resources receives the maximum effectiveness and any additional resource visiting target t in the time window will have no additional benefit.Thus, it is possible to define a total order relation $\geq$ on $\mathcal{A}$ such that $\alpha_i \geq \alpha_j$ if and only if (1) $\texttt{eff}(\alpha_i) \geq \texttt{eff}(\alpha_j)$ and (2) $\texttt{eff}(\alpha_i, \alpha_k) \geq \texttt{eff}(\alpha_j, \alpha_k), \forall \alpha_k$. In other words α_i provides a greater effectiveness than α_j.

Given a set of activities $S = \{\alpha_i\}_{i=1\ldots k}$ on a target within the same time window, labeled so that $\alpha_i \geq \alpha_j$ for all $i > j$, we extend the notation of $\texttt{eff}$ such that $\texttt{eff}(\{\emptyset\}) = 0$, $\texttt{eff}(S) = \texttt{eff}(\alpha_1)$ if $S = \{\alpha_1\}$, i.e., $|S| = 1$, and $\texttt{eff}(S) = \texttt{eff}(\alpha_1, \alpha_2)$ if $S = \{\alpha_i\}_{i=1\ldots k}$, i.e., $|S| > 1$. $\texttt{eff}(S)$ represents the maximum effectiveness of an individual or a joint activity over a set S of activities performed at a target within the same time window. One interesting aspect to understand about the operator $\texttt{eff}()$ is whether it is submodular or not. We define $\texttt{eff}()$ as *submodular* if for all $S_1 \subseteq S_2$ and all α_i the following condition holds:

$$\texttt{eff}(S_1 \cup \{\alpha_i\}) - \texttt{eff}(S_1) \geq \texttt{eff}(S_2 \cup \{\alpha_i\}) - \texttt{eff}(S_2) \tag{4.2}$$

[1] We associate effectiveness with activities and not with targets, assuming that each activity is equally effective at all targets.

This means that each additional activity performed has diminishing gains in effectiveness. The reason why we are interested in submodularity is that whenever it holds it becomes possible to define an approximate greedy algorithm, SMART$_H$, which provides performance guarantees on the quality of the solution that it calculates. More specifically, in Section 4.3, we formally demonstrate that whenever `eff`() is submodular, the solutions generated by SMART$_H$ are upper bounds to the optimal solutions of the problem (see Equation 4.26).

Therefore, the submodularity property is crucial from a practical perspective. Whenever it holds in a real-world domain, we can provide theoretical guarantees on the performance of the deployed officers. One example of such domain is port security, if one boat from the US Coast Guard is exploring a specific area of a port, any additional boat is unlikely to provide additional benefit in terms of deterrence effect or ability to capture criminals. In contrast, submodularity will not hold in domains where the defender has two different resources that only provide benefit when they are acting together. As we will see in Section 4.4.2, this is the case of the train domain, where some security resources (e.g., the EK 9) will be characterized by a null individual effectiveness but a non-zero joint effectiveness. If the submodularity property does not hold, the SMART$_H$ algorithm is still able to solve and generate an approximate solution of the problem, however nothing can be said about such solution's quality.

The expected utilities $U_d(\mathbf{P}_i, t)$ and $U_a(\mathbf{P}_i, t)$ for both players, when the defender is conducting pure strategy $\mathbf{P}_i$ (defined as a joint pure strategy for multiple defender resources), and when the attacker chooses to attack target t is given as follows:

$$\omega_t(\mathbf{P}_i) = \max_{\substack{(t,\alpha,\gamma)\in\mathbf{P}_i \\ \{(t,\alpha_l,\gamma_l),(t,\alpha_m,\gamma_m)\}\subseteq\mathbf{P}_i, |\gamma_l-\gamma_m|\leq W}} \{\texttt{eff}(\alpha), \texttt{eff}(\alpha_l, \alpha_m)\} \tag{4.3}$$

$$U_d(\mathbf{P}_i, t) = \omega_t(\mathbf{P}_i)U_d^c(t) + (1 - \omega_t(\mathbf{P}_i))U_d^u(t) \tag{4.4}$$

$$U_a(\mathbf{P}_i, t) = \omega_t(\mathbf{P}_i)U_a^c(t) + (1 - \omega_t(\mathbf{P}_i))U_a^u(t) \tag{4.5}$$

Here $\omega_t(\mathbf{P}_i)$ defined in Equation (4.3) represents the *effective coverage* of the defender on target t when executing pure strategy $\mathbf{P}_i$. This is computed by taking the maximum effectiveness of either a single or joint activity performed at target t at any time during the defender's patrols. The justification here is that in many domains the time that it takes to prepare and carry out a complex attack on a target, is often longer than the time required to patrol. Hence, we can safely assume that the attacker only cares about the maximum effective activity or joint activity (nonetheless, the formulation could be extended to situations involving shorter attack durations by dividing a patrol based multiple attack periods, a topic we leave for future work). Once the effectiveness $\omega_t(\mathbf{P}_i)$ is computed from the pure strategy $\mathbf{P}_i$, the defender and attacker expected utilities $U_d(\mathbf{P}_i, t)$ and $U_a(\mathbf{P}_i, t)$ are calculated as defined in Equation (4.4) and (4.5). The following example illustrates how the effectiveness of each pure strategy is calculated.

Example 1. *Consider the problem composed of* 5 *targets as given by the graph in Figure 4.1. The travel time on the edges between targets t_i and t_j is denoted as τ_{ij}. Assume that the home base, $t_b = t_1$. Furthermore, consider that the defender has* 2 *resources, each of which could conduct activities $\{\alpha_1, \alpha_2, \alpha_3\}$, such that α_1 is a thorough inspection of the target premises, α_2 is waiting at*

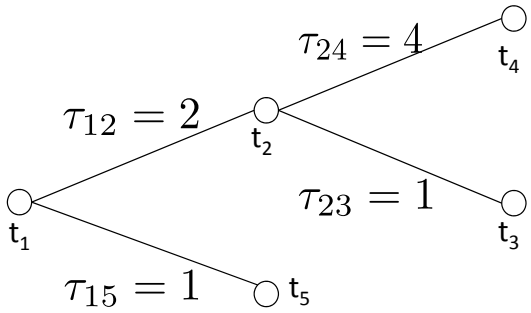


Figure 4.1: Example graph of targets and edges

Activity	α_1	α_2	α_3
Effectiveness, $\texttt{eff}(\alpha_i)$	0.5	0.4	0.1
Time	2	1	0

Table 4.2: Effectiveness of a single activity at any target.

	α_1	α_2	α_3
α_1	0.8	0.7	0.58
α_2	0.7	0.55	0.45
α_3	0.58	0.45	0.11

Table 4.3: Joint Effectiveness of each joint activity at any target.

the target looking for suspicious behavior and α_3 just transiting through the target area. Tables 4.2 and 4.3 give the time required for activity α and the effectiveness `eff` *for each individual as well as joint activity. The time at each tuple is computed by summing the distance between the targets and the activity time in Table 4.2.*

An example of a pure strategy for the first defender resource is: $X_1 = [(t_1, \alpha_3, 0), (t_5, \alpha_1, 3), (t_1, \alpha_1, 6)]$. *In words, the strategy describes a patrol whereby the first defender resource start at the home base target, t_1, performs activity α_3, go to t_5, performs activity α_1 and then returns back to the base, t_1 to perform activity α_1.*

An example of a pure strategy of the second defender resource is: $X_2 = [(t_1, \alpha_3, 0), (t_2, \alpha_3, 2), (t_3, \alpha_3, 3), (t_2, \alpha_3, 4), (t_1, \alpha_2, 7)]$. *This strategy describes a patrol where the second defender resource starts at t_1 and travels to t_2, t_3, t_2, and then back to t_1 performing activity α_3 at all targets except at the second visit target t_1, performing activity α_2.*

The pure defender strategy considering all defender resources is defined as $\mathbf{P}_1 = (X_1, X_2)$. *Assuming that the time window* $W = 2$, *then the effectiveness of the defender's pure strategy is computed by first determining the most effective single activity for this target, which is* α_1 *with* $\texttt{eff}(\alpha_1) = 0.5$ *for target* t_1, *as shown in Table 4.2. Looking at the schedule of the two defender resources, we then find the maximum coordinated joint activity that are within the time window* $W = 2$. *For* t_1, *this is determined to be* $(t_1, \alpha_1, 6)$ *for resource 1 and* $(t_1, \alpha_2, 7)$ *for resource 2, which gives* $\texttt{eff}(\alpha_1, \alpha_2) = 0.7$, *from Table 4.3. Thus,* $\omega_{t_1}(\mathbf{P}_1) = \max(0.7, 0.5) = 0.7$, *as defined in Equation* (4.3). *Similarly,* $\omega_{t_2}(\mathbf{P}_1) = 0.1$, $\omega_{t_3}(\mathbf{P}_1) = 0.1$, $\omega_{t_4}(\mathbf{P}_1) = 0.0$, $\omega_{t_5}(\mathbf{P}_1) = 0.5$.

Given pure strategy $\mathbf{P}_1$, *the defender's expected utility for target* t_1 *is computed using Equation* (4.4), *and is equal to* $0.7 \cdot U_d^c(t_1) + 0.3 \cdot U_d^u(t_1)$. *The attacker's expected utility for target* t_1 *given pure strategy* $\mathbf{P}_1$ *is computed in a similar fashion.*

Problem Statement: The objective of the defender is to maximize her expected utility in the SMART problem by computing the optimal mixed strategy given that the attacker will best respond to the defender's strategy.

4.2 SMART$_O$: Optimal Branch-and-Price Solver

SMART$_O$ is an optimal algorithm to compute solutions for SMART problem instances. It builds upon the ASPEN algorithm [Jain et al., 2010a], an optimal algorithm to solve SSGs based on the *branch-and-price* framework [Barnhart et al., 1994]. The two major novelties of SMART$_O$ over ASPEN are the formulation of a slave component capable of handling joint activities (in Section 4.2.1) and the improved upper bounds on the quality of the solutions recovered by the algorithm (in Section 4.2.2). The price SMART$_O$ pays for its optimality is its lack of scalability (as discussed later in Section 4.4.1). Nonetheless, it is useful to understand SMART$_O$'s functioning

because it lays the foundation for SMART$_H$, which is a more scalable algorithm. SMART$_O$ also allows us to measure SMART$_H$'s performance on smaller-sized problems.

4.2.1 Pricing component

The branch-and-price framework constructs a branch-and-bound tree, where for each leaf of the tree, the attacker's target is fixed to a different t'. The objective of the pricing component is to find the best defender mixed strategy $\mathbf{x}$ at that leaf, such that *the best response of the attacker* to $\mathbf{x}$ is to attack target t'. Due to the exponential number of defender pure strategies, the best defender mixed strategy is determined using *column generation*, which is composed of a *master* and *slave* procedure, where the slave iteratively adds a new column (defender strategy) to the master. Each component is defined as follows:

$$\min_{\mathbf{c},\mathbf{x}} -U_d(t', \mathbf{c}) \tag{4.6}$$

$$U_a(t', \mathbf{c}) \geq U_a(t, \mathbf{c}) \; \forall t \neq t' \tag{4.7}$$

$$c_t - \sum_{j \in J} \omega_t(\mathbf{P}_j) x_j \leq 0 \; \forall t \in T \tag{4.8}$$

$$\sum_{j \in J} x_j = 1 \tag{4.9}$$

$$x_j \in [0, 1] \; \forall j \in J, c_t \in [0, 1] \; \forall t \in T \tag{4.10}$$

Master: The master LP, given in Equations (4.6) to (4.10), finds the optimal defender mixed strategy $\mathbf{x}$ given a set of pure strategies J and assuming that the pure strategy of the attacker is

set to t' (determined by the leaf node).[2] This is similar in formulation to the ERASER algorithm [Kiekintveld et al., 2009]. $U_d(t', \mathbf{c})$ and $U_a(t', \mathbf{c})$ are the utilities of the defender and the attacker respectively when the defender's effective marginal coverage is $\mathbf{c}$ and the attacker attacks t'. Both $U_d(t', \mathbf{c})$ and $U_a(t', \mathbf{c})$ are defined following Equations 4.4 and 4.5, e.g., $U_d(t', \mathbf{c}) = c_{t'} U_d^c(t') + (1 - c_{t'}) U_d^u(t')$. For each pure strategy $\mathbf{P}_j$, $\omega_t(\mathbf{P}_j)$ is the effectiveness on t.

Slave: Once the master LP is solved to optimality, the slave problem receives the values of the *duals* of the master LP. The *reduced cost* $\overline{c}_j$ associated with column $\mathbf{P}_j$ is defined as follows:

$$\overline{c}_j = \sum_t y_t \cdot \omega_t(\mathbf{P}_j) - z, \tag{4.11}$$

where z is the dual variable of Equation (4.9) and $\{y_t\}$ are the duals of Equation family (4.8). The reduced cost of a column gives the potential change in the master's objective function when a candidate pure strategy is added to J. The candidate pure strategy with the minimum reduced cost is likely to improve our objective the most [Bertsimas and Tsitsiklis, 1994], since we are minimizing the master in Equation (4.6).

The objective for the slave problem is to find the column $\mathbf{P}_j$ with the least reduced cost, to add to the current set of columns. In addition, if the least reduced cost is greater or equal to 0, the current master solution is optimal for the full LP. The best column is identified using a mixed-integer linear program (MILP) formulation over the transition graph defined below, *which captures all the spatio-temporal constraints of the problem for handling joint activities and avoids having to enumerate all pure strategies.*

[2] For the sake of computation, we formulate the LP as a minimization problem (Equation 4.6); this will be explained in detail when we describe the slave procedure.

The transition graph $\mathcal{G}_r = (N'_r, E'_r)$ contains nodes $\mathtt{u} = (t, \gamma)$ for each target t and time instant $\gamma \in [0, \Gamma_r]$ if it is possible for the defender to be at target t at time instant γ (the time interval is discretized). Each edge in E'_r is associated with an activity α. An edge $e(\mathtt{u},\mathtt{v},\alpha)$ from node $\mathtt{u}$ to node $\mathtt{v}$ maps to a defender patrol that starts from target $t_\mathtt{u}$ at time $\gamma_\mathtt{u}$, goes to target $t_\mathtt{v}$ and conducts activity α at target $t_\mathtt{v}$. Therefore, we can calculate $\gamma_\mathtt{v}$ as follows:

$$\gamma_\mathtt{v} = \gamma_\mathtt{u} + \tau(t_\mathtt{u}, t_\mathtt{v}) + \tau(\alpha) \tag{4.12}$$

where $\tau(t_\mathtt{u}, t_\mathtt{v})$ is the time required to traverse from target $t_\mathtt{u}$ to $t_\mathtt{v}$ and $\tau(\alpha)$ is the time required to conduct activity α. The graph contains virtual `source` and `sink` nodes that contain edges to/from the base target t_b to ensure that patrols start and end at t_b.

Example 2. *Figure 4.2 shows a sample transition graph related to the problem presented in Example 1. Here, $t_b = t_1$ and the source has three edges, one for each activity $\alpha_1 - \alpha_3$. Looking at node $\mathtt{u} = (t_1, 0)$, target t_1 is adjacent to t_2 and t_5, so for each of these targets, three edges are added to represent the travel and corresponding activity at that target. For example, if activity α_2 is then performed at target t_2, then the new vertex would be at time $\gamma = 0 + \tau(\alpha_2) + \tau_{12} = 0 + 1 + 2 = 3$, where $\tau_{12} = 2$, and node $\mathtt{v} = (t_2, 3)$ as shown in Figure 4.2.*

Slave Problem MILP: We now describe our mixed integer linear programming formulation that identifies the pure strategy $\mathbf{P}_j$. The MILP for the slave problem is given in Equation (4.13) to (4.17). This novel MILP component of SMART$_O$ solves for joint activities and generates the optimal defender pure strategy.

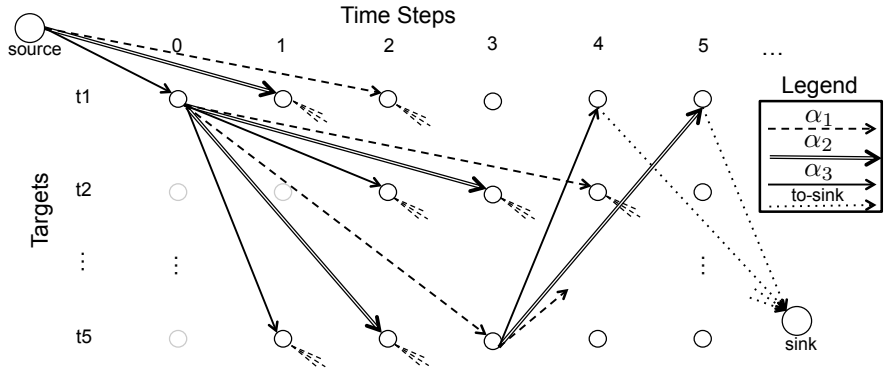


Figure 4.2: An Example for the Transition Graph

$$\min \quad \sum_{t \in T} y_t \cdot \max\{g_t, h_t\} \tag{4.13}$$

$$\sum_{e \in \texttt{out(u)}} f(e_r) = \sum_{e \in \texttt{in(u)}} f(e_r) \; \forall \; \texttt{u}, r \tag{4.14}$$

$$g_t = \max_{\substack{e_r \in \texttt{IA}(t), \\ \forall r}} \{f(e_r) \cdot \texttt{eff}(e.\alpha)\} \tag{4.15}$$

$$h_t = \max_{\substack{e_i, e_j \in \texttt{JA}(r_i, r_j, t), \\ \forall i, j \in R}} \{(f(e_i) + f(e_j) - 1) \cdot \texttt{eff}(e_i.\alpha, e_j.\alpha)\} \tag{4.16}$$

$$f(e_r) \in \{0, 1\} \; \forall e_r, g_t, h_t \; \in [0, 1] \; \forall t \in T \tag{4.17}$$

This MILP uses one copy of the transition graph for *each* defender resource, where $f(e_r)$ represents the flow on edge e for resource r, and g_t and h_t represent the effectiveness of the defender's individual and joint activities on target t. It only considers the maximum effective activity at target t (Equations (4.13), (4.15), and (4.16)) in accordance with our assumption of the attacker's decision making. In all three equations, the maximum effectiveness is computed using integer variables along with a large constant M. The resulting constraints are similar to the ones used in the DOBSS algorithms [Paruchuri et al., 2008].

Here, the set $\texttt{IA}(d)$ represents the set of edges in the transition graph such that they represent one resource performing an activity α on target d, and can be represented as:

$$\texttt{IA}(d) = \{\texttt{in}(\mathtt{u}_r)|\mathtt{u}_r.t = d, \forall \mathtt{u}_r \in N'_r, \forall r \in R\}$$

where $\texttt{in}(\mathtt{u}_r)$ represents all edges with the target node $\mathtt{u}_r$. Similarly, the set $\texttt{JA}(r_i, r_j, d)$ contains pairs of edges $\langle e_i, e_j \rangle$ such that both edges lead to the same target d and are separated by a time window no larger than W, corresponding to when resources i and j perform a joint activity on target d. Formally, $\texttt{JA}(r_i, r_j, d) =$

$$\{\langle e_i = (\mathtt{u}, \mathtt{v}), e_j = (\mathtt{u}', \mathtt{v}')\rangle|\mathtt{v}.t = \mathtt{v}'.t = d, |\mathtt{v}.\gamma - \mathtt{v}'.\gamma| \leq W\}.$$

Both sets IA and JA are defined over all transition graphs.

The result from the slave MILP is a set of 0-1 integer flows for each defender resource r. Given these flows, the defender pure strategy $\mathbf{P}_j$ and the effective coverage $\boldsymbol{\omega}(\mathbf{P}_j)$ are generated, and then both are sent back to the master.

4.2.2 Branch-and-bound component

In our branch-and-price framework, we define a separate branch for each attacker pure strategy, i.e. for each target t. Thus, the leaf node for target $\hat{t}$ has $q_t = 1$ for $t = \hat{t}$ and 0 otherwise. The pricing procedure described earlier is then used to compute the solution for this leaf node. This procedure is repeated for each leaf, after which the best solution obtained thus far is returned as the optimal

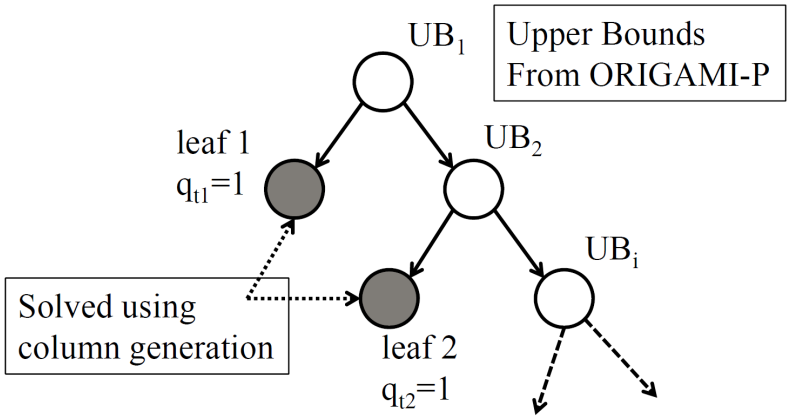


Figure 4.3: The structure of branch-and-price

solution.[3] An example branch-and-bound tree is given in Figure 4.3. The leaf (*gray*) nodes in the figure represent the pure strategies of the attacker, i.e., where the pricing computation is performed. The non-leaf (*white*) nodes represent the nodes for which upper bounds are obtained using a branch-and-bound heuristic (the branch-and-bound heuristic also determines the ordering of leaves, or attacker's pure strategies, in this tree). The objective of the branch and bound component is (i) to compute upper bounds for each internal node of the tree such that leaf nodes can be pruned thereby requiring less computation, and (ii) to determine an efficient ordering of leaves.

We generate these upper bounds using OrigamiP, a modification of Origami [Kiekintveld et al., 2009] specifically designed to generate tighter upper bounds for Smart problem instances by exploiting the structure of the domain.

[3]Only considering pure-strategies for the attacker is not a limitation; Stackelberg games always exhibit at least one Strong Stackelberg equilibrium where the attacker's best response is a pure strategy [Paruchuri et al., 2008].

$$\min_{\mathbf{c},\mathbf{f}(\mathbf{e})} \quad k \tag{4.18}$$

$$0 \leq k - U_a(t, \mathbf{c}) \leq (1 - q_t) \cdot M \; \forall t \in T \tag{4.19}$$

$$\sum_{e \in \texttt{out(source)}} f(e) = R \tag{4.20}$$

$$\sum_{e \in \texttt{in(sink)}} f(e) = R \tag{4.21}$$

$$\sum_{e \in \texttt{out(u)}} f(e) = \sum_{e \in \texttt{in(u)}} f(e) \; \forall \texttt{u} \tag{4.22}$$

$$c_t \leq \sum_{e=(\texttt{u},\texttt{v})|\texttt{v}.t=t} f(e) \cdot \texttt{eff}(\alpha_k) \; \forall t \in T \tag{4.23}$$

$$c_t \in [0, 1], \; q_t \in \{0, 1\} \; \forall t \in T, \; f(e) \in [0, R] \; \forall e \in E \tag{4.24}$$

OrigamiP uses the transition graph defined in the slave formulation (Section 4.2.1). Equations (4.18)–(4.19) minimize the attacker's maximum expected utility, $U_a(t, \mathbf{c})$ defined by Equation 4.5. This utility represents the attacker's utility given the defender's effective marginal coverage is $\mathbf{c}$ and the attacker attacks t. Since, the algorithm is used in the internal nodes of the branch-and-price tree, the attacker's target is fixed to a target t'. Thus, the integer variables q_t representing the attacker's pure strategy are fixed (for target $\hat{t}$, $q_t = 1$ for $t = \hat{t}$ and 0 otherwise) and the original MILP is simplified into an LP. Equations (4.20)–(4.22) define the flows of the edges and enforce the flow conservation property. Equation (4.23) limits the coverage of the defender based on the amount of flow and the respective activity. We can show that OrigamiP satisfies the following proposition:

Proposition 1. ORIGAMIP *computes upper bounds of the defender expected utility* $U_d()$ *if* `eff()` *is submodular.*

Proof. ORIGAMIP estimates the effectiveness of a defender patrol on a target as being the sum of the effectiveness of all individual activities on a target. This is an over-estimate of the effectiveness (thereby providing an upper bound on defender utility) if the effectiveness function `eff` is sub-additive, i.e., $\texttt{eff}(\alpha_i) + \texttt{eff}(\alpha_j) \geq \texttt{eff}(\alpha_i, \alpha_j)$, which is the case when `eff` satisfies the submodularity property in (4.2). □

ORIGAMIP is an LP and therefore solvable in polynomial time. Once the ORIGAMIP solution has been obtained, the defender's expected utility $U_d(t, \mathbf{c})$ is computed for each target t. The targets are then ordered in *decreasing* order of $U_d(t, \mathbf{c})$. This ordering and computation of upper bounds is then exploited to prune the nodes in the branch-and-price tree.

4.3 SMART$_H$: Further scaling up SMART

We now present the SMART$_H$ heuristic to further scale up the computation for SMART problem instances. As we will see in the following section, SMART$_O$ fails to scale beyond 4 targets in our computational experiments. Hence, we introduce SMART$_H$ which follows the same branch-and-price procedure discussed before but uses a novel heuristic slave formulation.

In essence, SMART$_H$ is built on two intuitions related to coordination. The first intuition is that joint patrols can be computed by considering individual patrols iteratively, by using *greedy policy optimization* between iterations to reflect the additive benefit of joint activities. The second intuition is that each defender resource would like to visit as many targets as possible, and visiting targets in accordance with an *ordering* based on a solution of the Traveling Salesman Problem is

likely to extract maximal benefit out of the resource while still accounting for the spatio-temporal constraints needed for coordination. As a result, the SMART$_H$ slave only needs to solve a set of *linear programs* (as opposed to solving a MILP in SMART$_O$'s slave).

4.3.1 Iterative Reward Modification

The slave in SMART$_H$ computes the joint patrol $\mathbf{P}_j$ of the defender by iteratively and greedily building up individual patrols X_r for each defender resource r. The additional benefit of joint activities is considered by appropriately shaping the rewards for each resource based on the patrols of other resources. Greedy policy optimization has been used in other reinforcement learning contexts [Sutton and Barto, 1998]; here we leverage this idea for coordination among multiple resources. This greedy approach allows SMART$_H$ to handle heterogeneous defender resources with each iteration solving for a different resource r.

Algorithm 1 SMART$_H$ Greedy Slave

```
1: Input: y, G
2: Initialize P_j, μ
3: for all r_i ∈ R do
4:     X_i ← SolveSinglePatrol(y, μ, G_r)
5:     P_j ← P_j ∪ X_i
6:     μ ← ComputeCostCoef(P_j, G_r)
7: ω(P_j) ← ConvertToColumn(P_j)
8: return P_j, ω(P_j)
```

SMART$_H$ uses a greedy algorithm, as outlined in Algorithm 1. This algorithm takes the coefficients y_t (refer Equation (4.11)) as input and builds $\mathbf{P}_j$ iteratively in Lines 3–6. Line 4 computes the best individual patrol X_r for the defender resource r (described in Section 4.3.2). X_r is then merged with the rest of the defender's pure strategy $\mathbf{P}_j$ (in Line 5). Line 6 computes $\boldsymbol{\mu}$, the potential effectiveness contribution from one resource to another given the current pure strategy $\mathbf{P}_j$.

This is computed over each edge $e(\mathtt{u},\mathtt{v},\alpha)$ in the transition graph, and measures the added benefit to the defender *if the defender resource was to travel from* $\mathtt{u}.t$ *to* $\mathtt{v}.t$ *at time* $\mathtt{u}.\gamma$ *performing activity* $e.\alpha$ *at target* $\mathtt{v}.t$. These values of $\boldsymbol{\mu}$ are used in the next iteration when computing an individual patrol for the next defender resource.

To understand how close the solution of the greedy algorithm is to the optimal solution, we use some insights from [Nemhauser et al., 1978], which states that greedy maximization of a non-negative submodular function achieves a constant-factor approximation. Recall that the objective of the slave problem is to find a pure strategy $\mathbf{P}_j$ that minimizes the reduced cost $\bar{c}_j$ (see Equation 4.11). This is equivalent to maximizing (since z in Equation 4.11 is a constant):

$$F(\mathbf{P}_j) = -\sum_{t \in T} \omega_t(\mathbf{P}_j) \cdot y_t \tag{4.25}$$

The duals $\mathbf{y}$ from the master are always negative in this formulation making $F(\mathbf{P}_j)$ non-negative. $\omega_t(\mathbf{P}_j)$ is the effectiveness of pure strategy $\mathbf{P}_j$ at target t as defined in (4.3).

If $F(\mathbf{P}_j)$ is submodular, and if $\mathbf{P}_*$ is the optimal defender pure strategy, then, as shown by Nemhauser et al. [1978] the solution $\mathbf{P}_j$ of the greedy algorithm satisfies

$$F(\mathbf{P}_j) \geq \frac{1}{2} F(\mathbf{P}_*) \tag{4.26}$$

For the special case where the time window, W, is greater than or equal to the maximum patrol time[4], Γ, we show that $F(\mathbf{P}_j)$ is submodular. $F(\mathbf{P}_j)$ is submodular if P_1 and P_2 are two sets of routes where $P_1 \subseteq P_2$ and $F(P_1 \cup \{X\}) - F(P_1) \geq F(P_2 \cup \{X\}) - F(P_2)$.

[4] $W \geq \Gamma$ implies that two resources present at the same target at anytime during the patrol are considered to conduct a joint activity.

Theorem 1. *$F(\mathbf{P}_j)$ is submodular in $\mathbf{P}_j$ if $W \geq \Gamma$ and* `eff`() *is submodular.*

Proof. Since $W \geq \Gamma$ and $\omega_t(\mathbf{P}_j) = \texttt{eff}(S_{\mathbf{P}_j})$, where $S_{\mathbf{P}_j}$ is the set of activities of $\mathbf{P}_j$ on target t. To prove that $F(\mathbf{P}_j)$ is submodular, it suffices to show that $\omega_t(\mathbf{P}_j)$ is submodular because $F(\mathbf{P}_j)$ is defined as a non-negative linear combination of $\omega_t(\mathbf{P}_j)$. Considering Equation (4.2):

$$\texttt{eff}(S_{P_1} \cup \alpha_X) - \texttt{eff}(S_{P_1}) \geq \texttt{eff}(S_{P_2} \cup \alpha_X) - \texttt{eff}(S_{P_2})$$

we can write $\omega_t(P_1 \cup X) - \omega_t(P_1) \geq \omega_t(P_2 \cup X) - \omega_t(P_2), P_1 \subseteq P_2$. Thus, $\omega_t(\mathbf{P}_j)$ is submodular when the time window is greater than or equal to the maximum patrol time. □

In real life situations, W may be less than Γ. We show that even in this situation, $F(\mathbf{P}_j)$ is submodular for 2 resources.

Theorem 2. *$F(\mathbf{P}_j)$ is submodular in $\mathbf{P}_j$ for two resources if* `eff`() *is submodular.*

Proof. We prove that $F(P_1 \cup \{X\}) - F(P_1) \geq F(P_2 \cup \{X\}) - F(P_2)$ where $P_1 = \{\emptyset\}$ and P_2 contains a single patrol $\{X_2\}$. To do this, we show that $\omega_t(\{X\}) \geq \omega_t(\{X_2, X\}) - \omega_t(\{X_2\})$, for each target t, based on the submodularity property of `eff`() in (4.2). We proceed in two steps. First, we show that:

$$w_t(\{X\}) \geq w_t(\{X_2, X\}) - w_t(\{X_2\}) \tag{4.27}$$

We use case-reasoning. If X_2 and X are in the same window then the same argument as Theorem 1 applies. Hence, we only need to demonstrate the equation for the case where X_2 and X are not in the same window. We have $w_t(\{X_2, X\}) = \max(\texttt{eff}(X_2), \texttt{eff}(X))$, then:

$$w_t(\{X_2, X\}) - w_t(\{X_2\}) = \max(0, \texttt{eff}(X) - \texttt{eff}(X_2)) \tag{4.28}$$

$$\Leftrightarrow w_t(\{X_2, X\}) - w_t(\{X_2\}) \leq \texttt{eff}(X) = w_t(X) \tag{4.29}$$

Second, we show that Equation 4.27 is equivalent to $w_t(P_1 \cup \{X\}) - w_t(P_1) \geq w_t(P_2 \cup \{X\}) - w_t(P_2)$:

$$w_t(\{X\}) \geq w_t(\{X_2, X\}) - w_t(\{X_2\}) \tag{4.30}$$

$$\Leftrightarrow w_t(\{\emptyset\} \cup \{X\}) - w_t(\{\emptyset\}) \geq w_t(\{X_2, X\}) - w_t(\{X_2\}) \tag{4.31}$$

$$\Leftrightarrow w_t(P_1 \cup \{X\}) - w_t(P_1) \geq w_t(P_2 \cup \{X\}) - w_t(P_2) \tag{4.32}$$

This shows that $\omega_t(\mathbf{P}_j)$ is submodular. As a consequence, $F(\mathbf{P}_j)$ is also submodular because it is a non-negative linear combination of $\omega_t(\mathbf{P}_j)$. □

Qualifying this result for $W < \Gamma$ for 2 resources *is important since this setup is used most frequently in the real world, e.g., the US Coast Guard*. For three or more resources, we can artificially construct counter-examples that break submodularity. However, given actual domain geometries, time windows, and operational rules, submodularity may hold even for larger number of resources – e.g., Theorem 1 shows that relaxing the time window may lead to such submodularity. Characterizing these spaces is a topic left for future work.

4.3.2 TSP Ordering with Transition Graph

To achieve the approximation bound in Equation (4.26), we need to optimally compute an individual patrol X_r for the defender resource r in line 5 of Algorithm 1. This can be solved by an MILP of similar form to the slave MILP (Equations (4.13)-(4.17)), but for a single patrol. The resulting MILP for a single patrol has less variables than the MILP for all patrols, however this still fails to scale up beyond 6 targets (Section 4.4).

Instead, we present a heuristic approach that achieves better scale-up by exploiting the spatial structure of the domain, and is provably optimal in some specific cases. Our approach is based on the following restricted version of the problem: we define an *ordering* of the targets and restrict the sequence of target visits to be increasing in this order. We construct the *ordered transition graph* in the same way as described in Section Ḣowever, now, an edge from node $\mathtt{u}$ to $\mathtt{v}$ is added *only if* target $\mathtt{u}.t$ appears before target $\mathtt{v}.t$ in the ordering. If there does not exist a direct edge from $\mathtt{u}$ to $\mathtt{v}$, an edge is added between these nodes such that $\tau_{u.t,v.t}$ is equal to the shortest path from $u.t$ to $v.t$. Traversing along this edge does not impact the effectiveness of the intermediate targets. Instead of computing the maximum effectiveness of the multiple edges per target, each target is only visited once per patrol in the ordered transition graph. Since each target in the patrol is counted only once, the max expressions in (4.13), (4.15), and (4.16) can be replaced with linear expressions. The resulting problem is equivalent to a min-cost flow, which has integer extreme points that allow us to drop the integrality constraint (4.17), since a feasible solution of the resulting LP is guaranteed to be an integer flow. Hence, these LPs are easier to solve than the above MILPs, both in theory as well as in our experiments.

Fixing an ordering will exclude certain patrols. Therefore, we would like an ordering such that the resulting patrol, which corresponds to a subsequence of the ordering, will still be a sensible way to visit targets compared to patrols with alternative orderings. To that end, SMART$_H$ uses an ordering based on the solution of the traveling salesman problem (TSP). Given an input graph of all targets, $G = (T, E)$, the orderings are generated using a nearest neighbour algorithm (as discussed by Gutin et al. [2002]) which determines the order to which the targets are visited in the patrol. Despite using an approximate algorithm, we are still able show that under certain conditions, the TSP ordering can yield an optimal solution of the single-patrol problem. When such conditions do not hold, it is likely that different orderings, based on more sophisticated TSP algorithms could result in a better solution. This could be a very interesting empirical challenge. However, we decided to focus our analysis on different aspects of the problem because in Section 4.4 we compare the performance of both SMART$_O$ and SMART$_H$ and show that the TSP ordering, despite being generated by an approximate algorithm, generates solutions that are very close to the optimal ones.

We look at a tree structure because various domains in the real world can be represented as a graph similar to a tree. For example, train lines can be represented as a line where edges connect each station and each station is connected to some other nodes representing the different levels of the station (e.g., platform, mezzanine or parking level). Similarly, ports can be imagined as a minimum spanning tree connecting all the locations within a port.

Theorem 3. *Suppose the input graph G is a tree, and the time window for joint effectiveness is greater than or equal to the maximum patrol time. Then* SMART$_H$ *computes a patrol for a single unit that optimizes the objective for the single unit problem in Algorithm 1.*

Proof. We first observe that the optimal TSP tour of G visits each edge of the tree exactly twice. The TSP tour corresponds to a complete preorder traversal of the tree.

SMART$_H$ outputs a patrol P on a subset of targets T_P, corresponding to a subsequence of the TSP ordering. We show that this patrol is a TSP tour of the corresponding subgraph on T_p, denoted G_p. There are two cases:

1. If G_P is a connected subtree of G then P is a preorder traversal of that subtree, and therefore is a TSP tour of the subtree.

2. If G_P is not connected, for each two targets in different connected components of G_P there is a unique path in the original tree graph G that connects the two. By adding nodes on these paths to G_P, we recover a connected subtree G'_P, an optimal TSP tour on which is also optimal for G_P. Then since P is a preorder traversal of the subtree G'_P, it is a TSP tour of G_P.

Consider a patrol P' on T_P that does not follow the TSP ordering. Let P be the patrol we get by reordering targets of P' so that they are increasing in the TSP ordering. Since P is a TSP tour of G_P, if P' finishes within the time limit then P also does. Furthermore, since the time window for joint effectiveness is large, joint activities in P' will also be joint activities in P, and thus P achieves the same slave objective as P'. Therefore, we never lose optimality by considering only patrols that follow the TSP order. □

When the graph is a tree but the time window is smaller than the patrol time limit, the algorithm is not guaranteed to be optimal. However, as we show in our experiments, SMART$_H$ generates optimal or near-optimal solutions for SMART problem instances.

4.4 Experimental Results

The section presents our simulations and our real-world experimental results. In Section 4.4.1, we extensively evaluate the performance of SMART$_H$ and SMART$_O$ in solving instances of SMART. We show the impact of the different components of our approach in terms of runtime and with respect to the difference in solution quality of the optimal versus heuristic algorithm. In Section 4.4.2, we describe our field experiment. This experiment constitutes the first real world head-to-head comparison between game-theoretic and human generated schedules. This comparison covers the effort to generate the schedules, the evaluation of security within a train line by security experts and the coordination between different deployed resources. This evaluation constitutes a contribution to evaluation not only of joint coordinated activities in the real world but also to the general evaluation of SSG-applications in the real world.

4.4.1 Simulations

In our simulations, we are not able to compare with previous algorithms [Conitzer and Sandholm, 2006; Jain et al., 2010a; Paruchuri et al., 2008; Vanek et al., 2011] due to the inability of these algorithms to scale up to the combinatorics unleashed by joint activities. Hence, we compare different versions of SMART$_H$ and SMART$_O$.

In the results below, in each experiment, each data point shown is averaged over 100 game instances, generated with random payoffs in the range [-10,10] and two defender resources unless otherwise noted. All the figures contains error bounds indicating the standard error of the mean. Given, the large number of instances, in some cases the bars are not shown, since they are too small to be seen. All the results are tested for statistical significance using a Student t-test (p

= 0.05). All experiments were run on graphs constructed beginning with a tree that spans the targets and then adding 10 random edges between nodes to create some random cycles in the graph. The root of the tree corresponded to the home base, t_b. The idea is to simulate the typical topology of a transportation hub or network, e.g., the areas of a port and the stations of a train line as in the port of Los Angeles or the Metro train line in Los Angeles. The time window was 30 minutes with 3 possible activities for the defender, taking 0, 5, or 15 minutes. In most of the experiments described below, the transition graph required to build the slave problem (see Section 4.2.1) was built using a time discretization of 5 minutes, i.e., any two adjacent nodes of the graph $u_1 = (t_1, \gamma_1)$ and $u_2 = (t_2, \gamma_2)$ were defined such that $\gamma_2 = \gamma_1 + 5$. In contrast, in the experiments where we compared SMART$_O$ and SMART$_H$, we used a larger time discretization of 15 minutes. This discretization was used for comparison purposes, i.e., SMART$_O$ would not run with the large transition graph produced as a result of a larger discretization. In such a setting, one activity had a duration of 0 minutes while the remaining two had a duration of 15 minutes. All experiments were run on a machine with a Dual core 2.0 GHz processor and 4 GB of RAM. We present our results in the remainder of this section.

4.4.1.1 SMART$_H$ vs. SMART$_O$: Runtime

In this experiment, we compare SMART$_H$ and SMART$_O$ in terms of runtime to solve different SMART problem instances. Each instance is generated considering an increasing number of targets. The results are shown in Figure 4.4. In the figure, the number of target is shown on the x-axis while the runtime is shown on the y-axis. We can see that SMART$_O$ is not able to scale to problem instances with more than 4 targets. In contrast, SMART$_H$ takes seconds to scale up to instances of 20 targets. As shown in the figure, when the number of targets increases to 5, SMART$_O$ runs out of memory

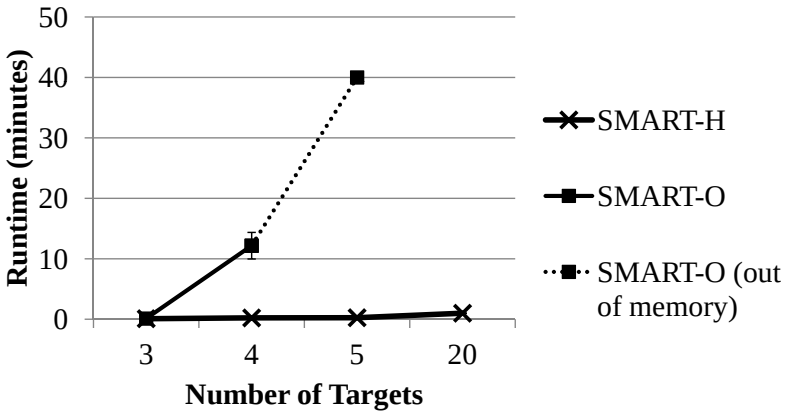


Figure 4.4: Runtime of SMART$_H$ vs SMART$_O$

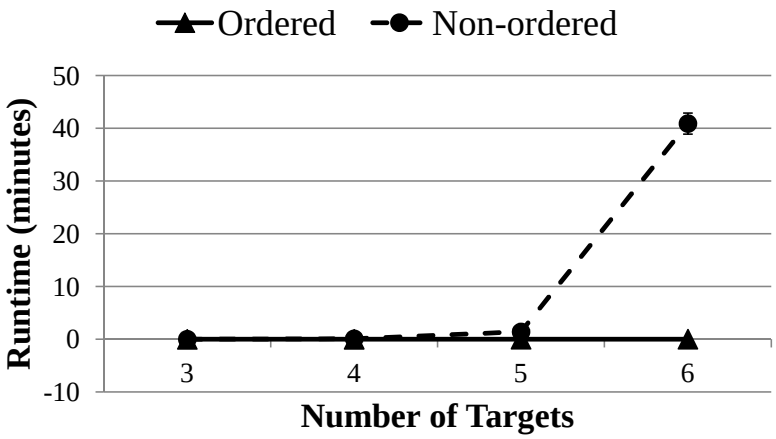
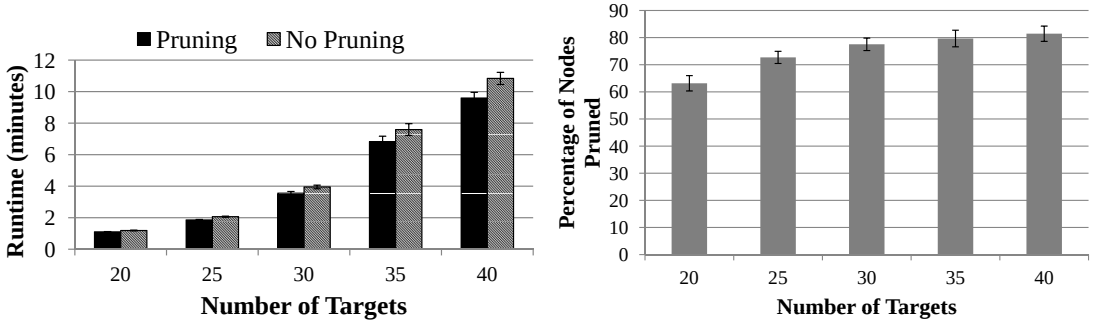


Figure 4.5: Using ordered nodes

and takes at least 40 minutes to run before giving an out-of-memory error. In contrast, SMART$_H$ takes less than a minute to solve problems consisting of up to 20 targets. This shows that SMART$_H$ dramatically improves both runtime computation and memory consumption compared to SMART$_O$.

4.4.1.2 SMART$_H$: Scalability using TSP ordering

In this experiment, we measure the ability of SMART$_H$ to scale up to large problem instances while using the TSP ordering procedure described in Section 4.3.2 and while not using any ordering procedure. The idea is to measure the benefits that ordering the nodes will do to the SMART$_H$ algorithm in terms of its scalability. In the experiment, we consider one single defender resource.



(a) pruning vs. non-pruning

(b) pruning vs. number of targets

Figure 4.6: SMART$_H$: benefits of pruning nodes

Figure 4.5 shows the results of our experiment. In the figure, the x-axis is the number of targets and the y-axis is the runtime. As shown in the figure, by using the MILP and not ordering the nodes, or allowing the defender to visit any node at any order in the path, the runtime for 6 targets is over 40 minutes. For 7 targets, SMART$_H$ not using node ordering runs out of memory. The impact of the TSP ordering heuristic is then significant: SMART$_H$ takes seconds for solving problems with up to 7 targets. In fact, as we will see in the next experiments, this heuristic allows the algorithm to scale up easily to problems with up to 40 targets.

4.4.1.3 SMART$_H$: Pruning

In this experiment, we analyze the benefits of generating tight upper bounds to prune the branch-and-price tree. The idea is to show in detail, why SMART$_H$ can scale up to large problem instances, while preserving memory.

Figure 4.6(a) shows the runtime (in minutes) required by the algorithm to solve SMART problem instances either pruning nodes (using ORIGAMIP) or not. The x-axis shows the number of targets and the y-axis shows the runtime. As shown in the figure, the amount of time saved by pruning nodes increases with the number of targets. We ran a student t-test (p=0.05) which confirmed the statistical significance of these results. In larger problem instances, the branch-and-price tree is

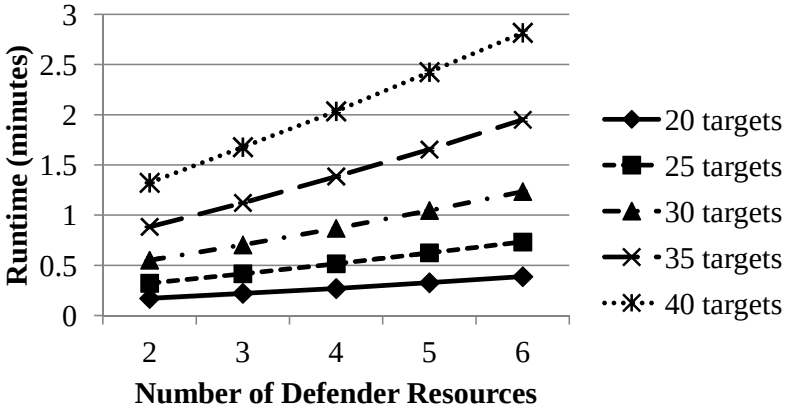


Figure 4.7: Runtime with multiple defender coordinating

bigger. Thus, a larger number of nodes will be pruned. This will have an impact on runtime: by using OrigamiP, the algorithm will require less time to solve bigger problem instances compared to simply solving all the leaf-nodes of the tree (i.e., no pruning).

Figure 4.6(b) shows the percentage of nodes pruned by Smart$_H$ considering Smart problem instances of different size. In the figure, the x-axis shows the number of targets while the y-axis shows the percentage of the nodes in the branch-and-bound tree that were pruned by Smart$_H$. The figure confirms the intuition behind the results in Figure 4.6(a), Smart$_H$ will prune more nodes on larger problem instances, e.g., up to 70% of the nodes for instances of 40 targets.

4.4.1.4 Smart$_H$: n-ary Joint Activities

In this experiment, we measure the ability of Smart$_H$ to solve large problem instances of up to 40 targets and large joint activity spaces of up to 10 activities at the same time. In Section 4.1, we defined the coverage effectiveness of a pure strategy in Equation 4.3. We used a max operator which, however, accounts only for the best single activity and joint couple of activities to calculate the coverage effectiveness. In this experiment, we modify Smart$_H$ to handle n-ary combinations of resources efficiently by re-defining the underlying utility function as an additive joint activity

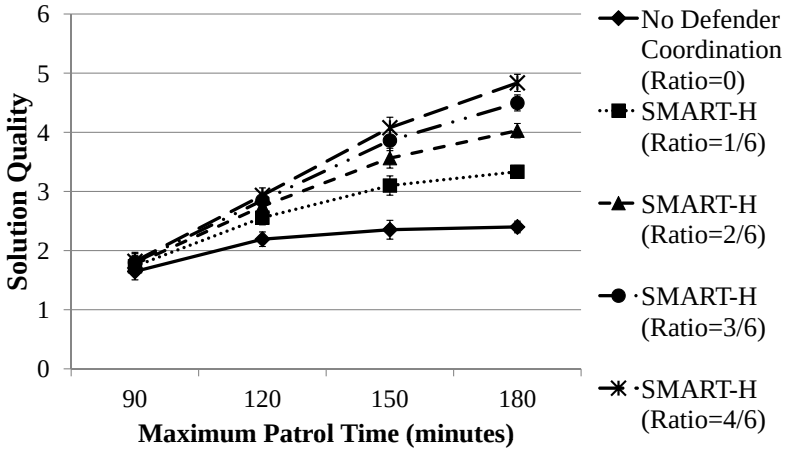


Figure 4.8: Solution quality of SMART$_H$ versus algorithm with no joint activities

function (capped at one). SMART$_H$'s iterative modification of reward then efficiently computes μ based on this new utility function based on the n-ary interaction (line 6 of Algorithm 1).

Figures 4.7 depicts the results of the experiment. In the figure, the x-axis is the number of defender resources. Each of such numbers is also associated with a specific arity of joint actions (e.g., 3 resources with ternary interactions) and the y-axis is the runtime. As we can see, SMART$_H$ can efficiently handle n-ary interactions since, the runtime never exceeds 3 minutes, even for instances consisting of 40 targets and 10 defender resources.

4.4.1.5 SMART$_H$: Effectiveness of Joint Activities

In this experiment, we compare the quality of the solutions obtained by running SMART$_H$ on SMART problem instances where the resources have activities with varying levels of joint effectiveness. The idea is to measure how SMART$_H$ will allocate resources when the effectiveness of their joint activities increases. We calculate the level of effectiveness as a ratio between the highest joint effectiveness value and the highest single effectiveness value, considering the best activity α_{max}:

$$\frac{\texttt{eff}(\alpha_{max}, \alpha_{max}) - \texttt{eff}(\alpha_{max})}{\texttt{eff}(\alpha_{max})} \tag{4.33}$$

For all test scenarios, $\texttt{eff}(\alpha_{max})$ is held constant to a value of 0.6, while varying the values of $\texttt{eff}(\alpha_{max}, \alpha_{max})$. For example, when $\texttt{eff}(\alpha_{max}, \alpha_{max}) = 0.8$, the subsequent ratio would be: $(0.8 - 0.6)/0.6 = 0.2/0.6 = 2/6$. The individual and joint effectiveness, here, are calculated as discussed in Section 4.1. They receive the maximum effectiveness and any additional resource visiting a target within the pre-defined time window will have no additional benefit (see Equation 4.3).

The results are shown in Figure 4.8. In the figure, the y-axis shows the solution quality and the x-axis denotes the maximum patrol time. Considering the experiment settings discussed at the beginning of this section, we can see that when the patrol time is increased, a simple strategy with no defender coordination (no benefit to joint activities) provides very little benefit to the solution quality while the improvement due to the coordination of multiple defender resources can almost double the solution quality. In more detail, taking into account joint activities between multiple defenders provides a solution quality that is double (when the ratio is 4/6) than that of an approach that does not handle defender coordination, i.e., when the ratio is 0.

4.4.1.6 Solution Quality against Hetergeneous Resources

In this experiment, we compare the quality of the solutions obtained by different versions of SMART$_H$ considering an increasing number of heterogeneous resources with different abilities. The idea is to measure the impact that different hetergeneous resources will have on the quality of the solutions recovered by SMART$_H$. We consider two type of resources: type A (T_A) and type B

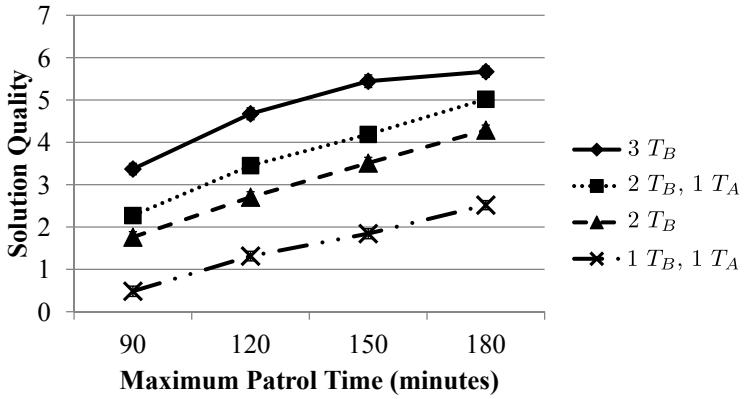


Figure 4.9: Heterogeneous defender resources: type A (T_A) and type B (T_B) for 30 targets. (T_B). Resources of type A are different than resources of type B in the following ways: (1) shorter transit times; (2) shorter patrol time; (3) lower effectiveness values.

Figure 4.9 shows the results considering both type A and type B. The figure shows the solution quality (i.e. the expected utility for the defender) obtained varying the maximum patrol time, the number of resources, their type and considering 30 targets. As we can see, increasing the number of resources lead to a higher solution quality. More specifically, we can see that by considering three resources instead of two, the expected utility increases from 1.77 to 3.46 (i.e., 50%) considering 90 minute patrols and from 4.31 to 5.69 (i.e., 25%) considering 180 minute patrols. Hence, as the number of resources increases, SMART$_H$ is able to allocate them effectively by exploiting their abilities to improve the overall solution quality.

4.4.2 Real-world Experiment

We present in what follows a real-world experiment whereby we compared the game-theoretic allocation of resources computed using SMART$_H$ against a manual allocation, the standard methodology adopted by several security agencies. Security agencies refer to this type of experiment as a mass transit full scale exercise (FSE). It was important to perform this exercise in the real

world rather than purely on paper with imaginary units, in order to ensure that the schedules generated by SMART$_H$ be compared to manual schedules when getting executed in the real-world under real-world constraints with real heterogeneous resource types.

A FSE is a training exercise where multiple security agencies analyze the way their resources cooperate to secure a specific area while simulating a critical scenario. This scenario typically describes a "high level" threat, e.g., intelligence reports confirming that a terrorist attack might take place in the Los Angeles Metro System. The FSE consists of simulating the response to this threat, i.e., increasing the number of resources patrolling a train line on a daily basis to improve the quality of the security. Nonetheless, in most real-world settings, the number of resources deployed by the agencies is not sufficient to cover all the different locations within a train line. For this reason, an intelligent and unpredictable allocation of security resources, which leverages their ability to work individually and cooperate together, is crucial to achieve a more effective security.

All the above reasons make an FSE a very promising test-bed to run our comparison between SMART$_H$ (SMART$_O$ would not scale to the dimensions of the FSE, as discussed in Section 4.4.1) and manual schedules. In addition, it would allow us to collect real-world data which we could use to analyze and improve the current algorithm.

4.4.2.1 Organization of the FSE

The FSE consisted of patrolling 10 stations of one train line of the LA Metro system for 12 hours. Each station on the train line is composed of three levels (street level, platform level and mezzanine) except station 1 which is composed of 5 levels (2 more platform levels). Figure 4.10 shows a graph illustrating the 10 stations.

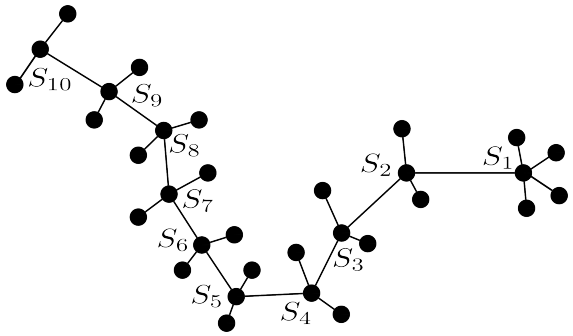


Figure 4.10: The 10 stations of the FSE

Team Description		
Acronym	Name	Deployed Teams
T Teams	High Visibility Uniformed Patrol Teams	T_{16} T_{27} T_{38} T_{49} T_{510} T_{11}
HVWT Teams	High Visibility Weapon Teams	$HVWT_{12}$ $HVWT_{34}$
VIPR Team	Visible Intermodal Interdiction Team	VIPR
CRM Teams	Crisis Response Motors	CRM_1 CRM_2 CRM_3
EK9 Teams	Explosive K9 (canine)	$EK9_1$ $EK9_2$

Table 4.4: Teams deployed during the FSE

The exercise involved multiple security agencies, each participating with a number of resources. Overall, 80 security personnel were involved. These resources were divided into 14 teams, each with different abilities. The resources deployed in the FSE are described in Table 4.4.

Figure 4.11: The smartphone application used to visualize the schedule of the CRM team

The exercise was divided into 3 different "sorties", each consisting of three hours of patrolling and one hour of debriefing. Human-generated schedules were used during the first sortie while SMART$_H$ schedules were used during the second and the third sorties. To visualize the schedules generated using SMART$_H$, each team was given a android smartphone app to visualize the game-theoretic schedule (see Figure 4.11).

The first two sorties were used to run the head-to-head comparison. Hence, the sorties were run under the same settings: the same number of officers had to cover the 10 stations for a cumulative time of 450 minutes. The two sorties were run during off-peak times (9h00 to 12h00 and 13h00 to 16h00, respectively), hence the type and the number of riders of the train lines could be considered to be, approximately, the same.

The purpose of Sortie 3 was to test whether the officers were capable of following SMART$_H$ schedules for a longer period (900 minutes instead of 450) and during peak time, i.e., when traffic and the number of people riding the trains increases. We found out that the officers were

actually able to follow the schedules. Thus, since the purpose of this Sortie was unrelated to our comparison, we will focus on Sorties 1 and 2 in the remainder of this section.

4.4.2.2 The Schedule Generation Process

Each type of schedule was generated as follows:

SMART$_H$ **schedules**: The schedules were generated by (i) instantiating a SMART problem instance using the specifics of the FSE discussed earlier; (ii) solving this problem instance using SMART$_H$ and (iii) sampling a pure strategy to generate the patrol schedule for each of the different resources involved.

To define a problem instance we had to define three different sets of features of the problem. The first sets of features are the graphs for each resource type. By using the graph presented in Figure 4.10 as a baseline, we defined the graph for each resource as follows:

- T teams, HVWT teams and VIPR teams move using the trains and patrol all the different levels of a station. Hence, each resource type was assigned the graph in Figure 4.10.
- CRM teams move using their bikes and patrol only one level (the street / parking level) of the stations. Their graph is then a line connecting ten nodes, each representing the street level of a station.
- EK9 teams move using a car and patrol all levels. Their graph is defined as in Figure 4.10 but edges connect the nodes representing the street level instead of the platform level.

The second sets of features are the payoffs of the game.[5] We defined the payoffs for each target (32 in total) in discussions with security experts from the Los Angeles County Sheriff's

[5]We are not able to reveal the value of these payoffs due to an agreement with the Los Angeles County Sheriff Department (LASD).

Department (LASD). Each set of payoffs for each station was based on the number of people using the station every day and by the economic impact that losing this station would have on the city. The different levels of a single station had slightly different payoffs which were based on the number of persons present at each specific level of the station every weekday.

The payoffs were defined so that the game was zero-sum, i.e., given a target t, we defined $U_d^c(t) = U_a^c(t) = 0$ and $U_d^u(t) = -U_a^u(t) = v$ where $v \in [0, 10]$. This choice of payoffs is reasonable for this setting because the key here is the importance of stations, and the defender will lose exactly as much as the attacker will gain if his attack is successful.

The third features are the activities for each resource type and the corresponding single and joint effectiveness. Each team could perform a series of two types of activities:

Observe: This action consisted of observing a specific target for 15 minutes. The officers had to explore the location (e.g., street level or platform) with great accuracy to ensure that it was secure.

GoTo: This action consisted of moving from one target to another (e.g., by using a car, by riding a train or using the stairs to move between targets at the same station). The duration of this action was estimated based on the train schedules, the traffic at the time of the day and the average time to move from one level of a station to another.

Given these activities, the single and joint effectiveness parameters are summarized in Table 4.5. In the table, the joint activities are represented as a vector of two values, one for each joint activity between the action in the row of the table and an observe action and a goto action, respectively. As we can see, *all* GoTo actions are given a 0 effectiveness, since moving from one station to another (i.e., riding the trains or taking the car) will not have any effect on the security of

Team	Action	Individual eff	Joint eff
T-teams	Observe	0.7	[0.0, 0.0]
	GoTo	0	[0.0,0.0]
HVWT teams	Observe	0.7	[0.0, 0.0]
	GoTo	0	[0.0,0.0]
VIPR team	Observe	0.8	[0.0,0.0]
	GoTo	0	[0.0,0.0]
CRM teams	Observe	0.7	[0.75,0.0]
	GoTo	0	[0.0,0.0]
EK9 teams	Observe	0	[0.75,0.0]
	GoTo	0	[0.0,0.0]

Table 4.5: Individual and joint activities

the stations. Most teams are assigned the same positive individual effectiveness of 0.7, except the VIPR team which has a greater individual effectiveness because it is composed of officers from multiple agencies carrying heavy weapons. VIPR teams, T-teams and HVWT teams typically work alone. Hence, to define their effectiveness values, their individual effectiveness is positive while their joint effectiveness is null (any joint effectiveness value below 0.7 would induce the same type of behavior, but we chose 0 since it is a clear indicator of the type of behavior that we want to obtain). The CRM teams are assigned a joint effectiveness greater then their individual effectiveness because they can perform all type of activites, but, typically, they prefer joint over individual activities. In contrast, EK9 teams typically work only in cooperation with other teams, therefore they are assigned a null individual effectiveness and a positive joint effectiveness of 0.75.[6]

Next, we defined the time window for a joint action to be effective as a 10 minutes interval. This value was chosen considering the time required by teams to move from a station to another, from a level to another and by discussion with the security agencies involved in the exercise.

[6]Whereas these estimates of individual and joint effectiveness could potentially be slightly altered, the purpose of our exercise was comparison with human schedulers. Since the comparison was ultimately conducted by security experts who evaluated the game-theoretic schedule to be superior to human-generated one, we may infer that the effectiveness values we obtained for the individual and joint activities in our SMART model for FSE were reasonable.

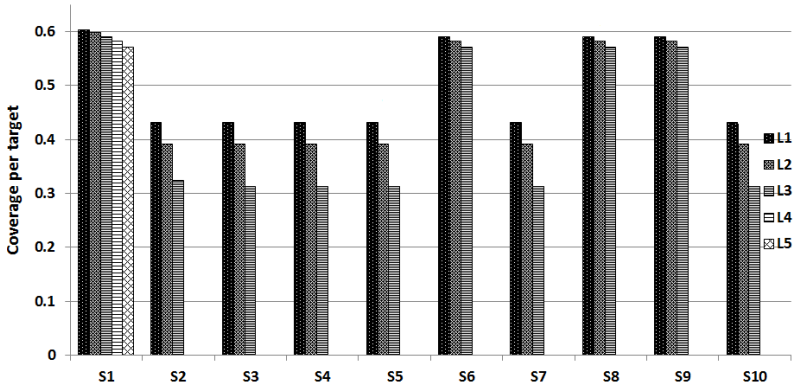


Figure 4.12: The mixed strategy for the FSE

Finally, we decided the discretization of the transition graph to correspond to the shortest duration between the durations of all the actions available to the different resources (i.e., the goto action to move between two levels).

After defining the SMART problem instance, we solved it using SMART$_H$. We run the algorithm considering 14 resources (the teams defined in Table 4.4), and 32 targets (5 levels for station S_1 and 3 levels for the 9 other stations). To reach the cumulative time of 450 minutes, as required by the specifics of Sortie 2, we defined the patrol time of each resource such that a total close to 450 minutes could be obtained. The mixed strategy provided by the algorithm is shown in Figure 4.12. The figure shows the coverage c_t for each target t as defined by Equation 4.8 in the master in Section 4.2. In the figure, the levels of the stations with a higher payoff are assigned a higher coverage.

As a final step, the mixed strategy is sampled to generate a pure strategy. This pure strategy contains a schedule for each resource. It is shown in Table 7.2 in Appendix A.

Manual Schedules: The schedules were by human expert schedulers of the LASD. They were generated using a two-step process. First, each station was assigned a coverage duration of 45 minutes (i.e., $\frac{1}{10}^{th}$ of the time). The idea was to have the officers perform *three observe* actions

	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}
Manual	3	3	3	2	3	2	2	2	2	2
SMART$_H$	2	2	3	3	2	2	2	3	3	2

Table 4.6: Count of Individual Activities

	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}
Manual	0	0	0	1	0	1	1	1	1	1
SMART$_H$	1	0	0	0	0	2	0	1	1	1

Table 4.7: Count of Joint Activities

at each station. Second, the human expert schedulers assigned teams to each station so that each station was covered for exactly 45 minutes. Joint team activities were used 6 times in six different stations. The resulting allocation is shown in Table 7.1 in the Appendix A. This simple two-step process was adopted to avoid the cognitive burden involved with leveraging the effectiveness of each team to cover the different stations individually or while coordinating with other teams. Despite its simplicity, this process was difficult for the human expert schedulers. It involved several discussions and required one entire day of work.

Having defined the two allocations, we can now analyze the results that we obtained.

4.4.3 Results

We first analyze the type of schedules generated as a result of using either SMART$_H$ or manual scheduling. Then, we evaluate the results obtained by deploying the schedules during Sorties 1 and 2 and measuring their performance in the real world. The key is that if SMART$_H$ were to perform equivalently to human schedulers, then it would indicate that we could save precious time so security experts could focus on maintaining security rather than on generating schedules.

4.4.3.1 Schedules Comparison

The allocation of resources generated by Manual scheduling and SMART$_H$ are shown in Tables 7.1 and 7.2 in Appendix A. The numbers of individual and joint activities for both schedules are shown in Tables 4.6 and 4.7. In both tables we can see that the number of individual (IA) and joint (JA) activities for both approaches are the same (IA: both 24; JA: both 6). All the joint activities in the SMART$_H$ schedules are performed by CRM and EK9 teams, i.e., the teams with a positive joint effectiveness. This is similar to the behavior of the manual generated schedules, where joint activities are mostly performed by EK 9 and CRM teams (once by the VIPR team). The remaining individual activities are performed by the T team, the HVWT team and the VIPR team.

There are two important differences between the two types of schedules. The first is that the game-theoretic scheduler sent the most effective VIPR team to the most important stations – because its individual effectiveness is greater than the effectiveness of other teams. This was not seen in the human schedule. The second difference between the two types of schedules is that the schedules generated using SMART$_H$ assign the different teams to cover all the different levels of the different stations, whereas manual schedules do not specify such levels. The reason for this is that human schedulers were not able to reach this level of detail and thus they preferred to leave the decision of which level to patrol to the teams once they were deployed. In addition to accuracy, in SMART$_H$, the human effort was confined to providing the individual and joint effectiveness of available teams, and then the schedules were automatically generated in just a couple of hours. Hence, the effort required to generate the schedules using SMART$_H$ was much lower than the effort required to generate manual schedules, which, as discussed above, required one day of work due to its significant cognitive burden. Since typically such patrols would be conducted

Q_1	"The security scheduling system (SSS) makes the station safer."
Q_2	"The SSS is efficient."
Q_3	"The SSS is an effective deterrent for adversaries."
Q_4	"The SSS results in some areas being patrolled more than needed."
Q_5	"The SSS decreases the number of attempts to infiltrate train stations."
Q_6	"The SSS appears to make securing train stations easier."
Q_7	"The SSS results in enough security at all three levels of the train station."
Q_8	"The SSS results in security officials having a strong presence throughout the station."
Q_9	"The SSS results in ALL areas being patrolled as much as needed."
Q_{10}	"The SSS did NOT prevent patrollers from taking or completing an action."
Q_{11}	"The SSS provides security officials with enough time to secure all areas of the station."

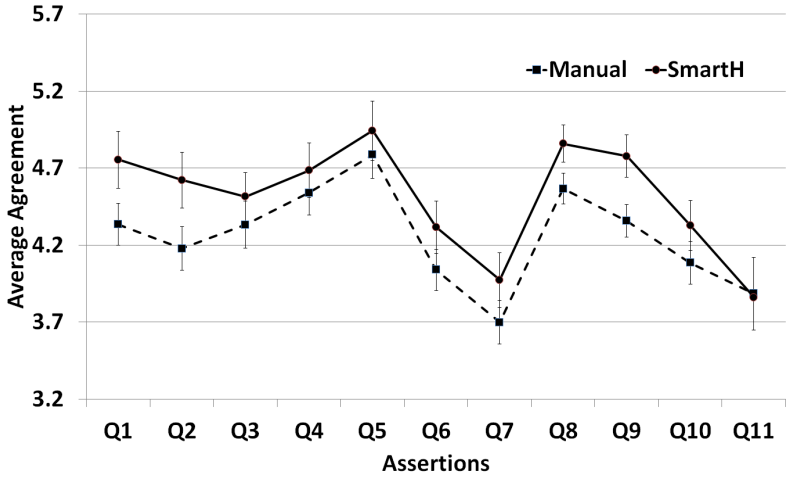
Table 4.8: The 11 assertions used in the questionnaire during the FSE

day-in and day-out for several days in high-threat periods, the savings of human effort achieved by game-theoretic schedulers are thus very significant.

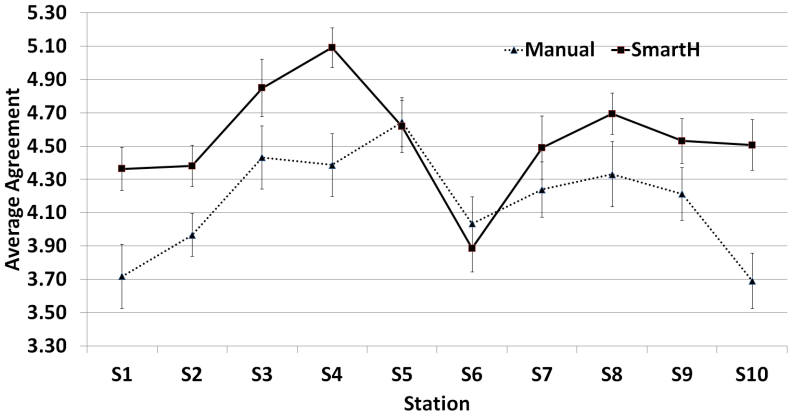
4.4.3.2 Evaluation by Securty Experts

Each type of security allocation (either manual or game-theoretic based on SMART$_H$) was evaluated by security experts. In this setting, individual and joint activities between different resources played an important role.

For the purposes of this evaluation, a team of security experts (SEs) was placed at each station for the entire length of the exercise. Their task was to observe and evaluate the officers' patrolling activity during each sortie, and determine how their behavior was affecting the quality of the security within each station. In what follows, we report the conclusions of their analysis. The SEs did not know the type of schedules (so as to not bias their evaluation). To translate the observers' observations into a comparable value, each observer was asked to fill out a questionnaire every 30 minutes. The objective was to define a number of key sentences that could help to qualify the way in which the security officers had been patrolling the station in the last 30 minutes. Each questionnaire contained 11 *assertions* about the level of security within the station. Table 4.8



(a) Assertions



(b) Stations

Figure 4.13: Evaluation of the FSE: average agreement over the different questions and stations.

summarizes the assertions used in the questionnaire. Each assertion was a sentence defining a key aspect related to the security of a station. The assertions were defined in collaboration with a team of SEs from the LASD and with social scientists. Each SE had to determine his level of agreement with each assertion. The level of agreement was defined in the integer interval {0,6}, where 0 meant a strong disagreement, whereas 6 meant a strong agreement.

Figures 4.13(a) and 4.13(b) show the results that we obtained. Figure 4.13(a) shows the weighted average agreement obtained for each assertion calculated over all the stations (the average was calculated considering each station's corresponding weight). Figure 4.13(b) shows

the average agreement obtained for each station calculated over all the assertions. The error bars in both figures show the standard error of the mean calculated for each specific assertion (in Figure 4.13(a)) and station (in Figure 4.13(b)). As we can see the difference between some data points of the two approaches do not seem to be statistically significant. A student t-test confirmed this trend. This is expected, since we were only able to collect data for few hours of a single day. Nonetheless, we can still acquire some interesting information about the performance of game-theoretic schedules in the field, by analyzing the results that are statistically significant. In the next section then, we will discuss how running additional experiments is a key challenge to confirm the trends presented here.

In Figure 4.13(a), we can see that SMART$_H$ schedules seem to yield a higher level of agreement than manual schedules over all questions. As shown in the figure, the difference is significant only for assertions Q_1, Q_2, Q_8 and Q_9. As shown in Table 4.8, these four assertions correspond to very general statements about the security at each station which address the efficiency of the schedules, their ability to provide a strong feeling of safety and to allow the officers to patrol each area as much as needed.

Similarly, in Figure 4.13(b), we can see that the average agreement is higher for SMART$_H$ schedules over Manual schedules for stations S_1, S_2, S_3, S_4, S_8, S_9 and S_{10}. Some of these stations (S_1, S_8 and S_9) are the ones assigned a higher set of payoffs, as discussed above (see Figure 4.12). Hence, they correspond to the ones given a higher coverage by SMART$_H$ (see Figure 4.12).

These results indicate that game-theoretic schedules were evaluated as more effective than manual schedules. Analysis reveals that whereas the game-theoretic schedules were able to incorporate joint activities in a fashion comparable to the human schedules– and this was important

to ensure security effectiveness –the game-theoretic schedules were able to be more effective in two key ways that showed the limitations of human schedulers. First, manual schedules were generated by leaving the decision of which level of a station to patrol to each deployed team. The officers then, were not able to properly coordinate over the different levels to patrol and therefore they ended up patrolling the same levels. In do doing, the were not able to fully cover the different stations. In contrast, as shown in Appendix A, SMART$_H$ produced a schedule which tackled the comprehensive security of all the ten different stations. The officers knew before-hand which levels they had to patrol and therefore, it was unnecessary for them to coordinate their decisions during the exercise.

Second, SMART$_H$ produced a schedule which more effectively scheduled the VIPR team, i.e., the team with the highest effectiveness (0.8) for covering each target. As we can see in Table 7.2, the schedule generated by SMART$_H$ had the VIPR team patrol *all* the most important stations at key levels. In contrast, manual schedules assigned the VIPR team, without accounting for its effectiveness. This made an impact on the security evaluators. By observing the VIPR at key locations, they considered the game-theoretic allocation more effective than the manual allocation, because it was using leveraging the abilities of the resources in a way that human experts could not achieve.

Overall, these results show the potential of game-theoretic security allocation to solve real world problems. In the next section, we discuss our future work, whereby we describe some new experiments that could be ran to further strengthen the results presented here.

4.5 Chapter Summary

This chapter addressed the challenge of solving security games where multiple defenders resources receive benefits from performing joint coordinated activities. This challenge has not been addressed in previous work in security games. However, incorporating such joint activities into the existing SSG framework is critical for real-world applications.

To address this challenge, this chapter presented four contributions. First, we presented SMART, a new type of SSG which accounts for multiple defenders performing joint activities. Second, we presented two branch-and-price based approaches, SMART$_O$ and SMART$_H$ to solve SMART problem instances. SMART$_O$ computes optimal solutions of SMART problem instances and uses a novel slave formulation that captures coordination in both space and time. SMART$_H$ is an heuristic approach based on reward shaping and TSP ordering to speed-up the computation. Third, we provide proofs of theoretical properties of our algorithms while also showing the improved performance of the key components in simulation.

Fourth, we present the first head-to-head comparison between SSG based schedules and manual schedules in the field. To the best of our knowledge, this evaluation constitutes one of the largest evaluation of *algorithmic* game theory in the field to date. In more detail, we present the results that we obtained by organizing a large scale real-world experiment whereby 80 security officers (divided into 23 teams) patrolled 10 stations of a metro line for one day. In this experiment, we ran a head-to-head comparison between SSG-based schedules, generated using SMART$_H$, and human-generated schedules. Our results were based on an analysis provided by a team of security experts analyzing the performance of each type of schedule at each of the ten stations. The results

showed that in comparison with human schedulers, game-theoretic schedules were able to reduce the effort to generate schedules, while improving coordination and perception of security presence.

In terms of future work, we aim to address one key challenge, that of execution uncertainty. In some security domains, the deployed resources may sometimes be delayed. For instance, USCG officers might be delayed because they need to board a boat. To address this challenge, our idea is to generalize the current model into a decentralized Markov decision process (dec-MDP [Becker et al., 2004]). Recently, this challenge was addressed by Shieh et al. [2014]. In their work, Shieh et al. [2014] propose a new security game model where a Dec-MDP is used to model execution uncertainty between different resources, i.e., situations where one resource is delayed and cannot complete his schedules albeit all the resources are still required to coordinate. The increase in complexity of the new model, which blends a security game and a DecMDP, is addressed by proposing a novel set of approximate algorithms which use column generation in a manner similar to the one presented in our work.

Additionally, as shown by the results presented in this chapter, we believe that our work opens the door of applied research in security games to the realm of field evaluation. Given the strong connection that research in SSGs shares with real world security allocation problems, we argue that field evaluation should become a key area for future research in security games.

Chapter 5: Teamwork with Execution Uncertainty (Utilizing Dec-MDPs in Security Games)

This chapter focuses on this challenge of computing an optimal resource allocation strategy for a defender team while also considering uncertainty in coordination of multiple defender resources. To that end, this chapter 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 resources 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 chapter. First, multiple defenders may be required to coordinate their activities under uncertainty, e.g., delays that may arise from unexpected situations may lead different resources to miscoordinate, making them unable to act simultaneously. Second, some resources 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 outlining these challenges.

To handle teamwork of defender resources in security games, our work makes the following contributions. First, this chapter 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. Third, global events among defender resources (e.g., a defender resource stops patrolling due to a bomb threat) are modeled in handling teamwork. Fourth, we contribute heuristics 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 chapter applies to many of the application domains of security games, including the security of flights, ports and rail [Tambe, 2011], we focus on the metro rail domain for a concrete example, given the increasing amount of rail related terrorism threats [Reuters, 2013]. 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. In the rest of the chapter, we use the term resource and agent interchangeably (i.e., one resource is equivalent to one agent).

This chapter is organized as follows: Section 5.1 starts with exploring the importance and challenges in the metro rail domain. Section 5.2 presents a brief background on Dec-MDPs and security games. Section 5.3 presents the game theoretic model to address uncertainty among defender resources in a security game. Section 5.4 describes the algorithm to solve and compute the defender strategy. Section 5.5 presents heuristics to improve the runtime. Section 5.6 follows by describing heuristics to handle robustness. Section 5.7 provides experimental results for all of

our algorithms and heuristics. Section 5.8 summarizes the contributions of the chapter and future work.

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 Carter [2013]. 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 Reuters [2013]. 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" INFORMS [2012].

In the metro rail domain, the defender resources (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 resources 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 resources 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 [Varakantham et al., 2013b] and security patrolling for fare inspection in the Los Angeles Metro Rail system [Jiang et al., 2013c].

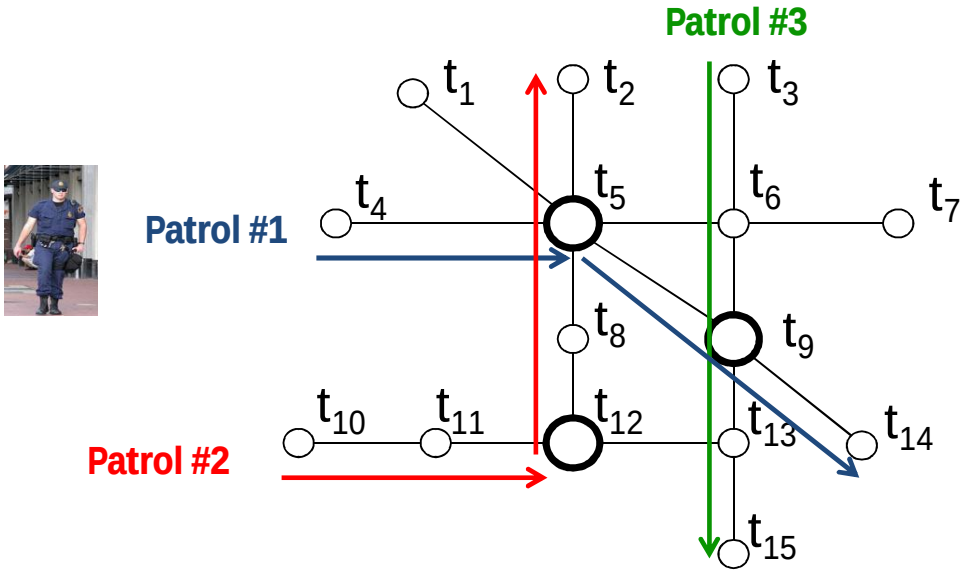


Figure 5.1: Example of the metro rail domain

In Figure 5.1, 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 resource 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 resources 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 resources may execute multiple patrols, e.g., Patrol 1 and 2 in Figure 5.1, and coordinate to visit a single station simultaneously (like station 5). Thus, if the adversary observes a coordinated set of

defender resources 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 resources 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 resource 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 resources 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 resources. This type of uncertainty occurs on a local level to the individual defender resource and is not known by the other defender resources (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 resources visit station 5 together.

Second, a global event may cause a resource to leave the system and stop patrolling. This type of global event affects the entire team and impacts the coordination among patrol resources which is different from the local level events that may occur to an individual defender resource. One of the defender resources 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 resources may take over the response, resulting in the resource stopping the patrol and requiring others to fill in any gaps as a team. The remaining resources 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 resources. 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 resources from constantly communicating with other resources to determine their location.

5.2 Background

This section begins by introducing, motivating and providing background on security games in Section 5.2.1. Section 5.2.2 then discusses Decentralized Markov Decision Processes.

5.2.1 Security Games with an Example Application in the Metro Domain

Security games were first formalized by Kiekintveld et al. [Kiekintveld et al., 2009] 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 [Gatti, 2008; Jakob et al., 2012; Vaněk et al., 2011]. 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 [Kiekintveld et al., 2009; Pita et al., 2008; Yang et al., 2012]. 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 [Jiang et al., 2013c], but without coordination of multiple resources 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 5.3, but based on the concepts of marginals introduced in [Kiekintveld et al., 2009]). The defender's pure strategies correspond to visiting a set of target-time pairs given a set of resources. By convention, in the rest of the chapter, 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[Jain et al., 2011; Shieh et al., 2013; Yin et al., 2010]. 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 [Basilico et al., 2012; Jain et al., 2010c; Tambe, 2011]. The model presented in this chapter 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 resources 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 [Paruchuri et al., 2008]. 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 [Leitmann, 1978]. 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 [Conitzer and Sandholm, 2006;

Jain et al., 2010b, 2011, 2010c; Jiang et al., 2013c; Kiekintveld et al., 2009; Pita et al., 2010, 2008; Shieh et al., 2012, 2013; Yin et al., 2010].

5.2.2 Dec-MDP

In this chapter, 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 [Dibangoye et al., 2012; Goldman and Zilberstein, 2008; Spaan and Melo, 2008; Varakantham et al., 2009], 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) [Becker et al., 2004; Bernstein et al., 2002; Dibangoye et al., 2012, 2013; Spaan and Melo, 2008], 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 [Amato et al., 2014; Amato and Oliehoek, 2015; Bernstein et al., 2002; Oliehoek et al., 2013, 2015; Wu et al., 2013]. As we will explain later, when solving the security game model introduced in this chapter, 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 [Becker et al., 2004] that is defined by the tuple: $\langle Ag, S, A, T, R \rangle$. $Ag = \{1, \ldots, n\}$ represents the set of n agents (or defender resources) [Bernstein et al., 2002]. $S = S_u \times S_1 \times \cdots \times S_n$ 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 [Nair et al., 2005].

$A = A_1 \times \cdots \times A_n$ 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. $T : S \times A \times S \rightarrow \mathbb{R}$ 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 resource i's local states are independent of actions of other agents, we have transition independence [Becker et al., 2004]. 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 $R : S \rightarrow \mathbb{R}$, 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).

5.3 Game Model

This chapter presents a game theoretic model of effective teamwork among multiple decentralized defender resources with execution uncertainty. We are generalizing the security game model introduced in Section 5.2.1 to multiple defender resources coordinating under uncertainty. This section starts with a model for the defender team and uncertainty. Next, it goes into detail of the defender's effectiveness at each target-time pair (Section 5.3.1). Then, the defender's pure strategy along with the attacker and defender's expected utility is discussed (Section 5.3.2). Finally, global events are explained and addressed in the model (Section 5.3.3).

The model for the defender team is represented by the tuple similar to the one for Dec-MDP as described in Section 5.2.2: $\langle Ag, S, A, T, U \rangle$. The main difference between this tuple and the one

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 resources
s_r	State of resource r, composed of a location(target) t, and time τ
ξ	Effectiveness of a single defender resource
$\texttt{eff}(s, b)$	Effectiveness of the resources 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 5.1: Notation for game formulation

presented in Section 5.2.2 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 resource consists of a sequence of commands; each command is of the form: at time τ, the resource should be at target t and execute action a. The action of the current command takes the defender resource to the location and time of the next command. In practice, each defender resource faces execution uncertainty, where taking an action might result in the defender resource 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 resource 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 resources at each target-time pair. So in other words, the attacker cares about how often and with how many resources 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 5.3.3). 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.

5.3.1 Defender Effectiveness

This section explains the value of the defender's effectiveness starting with a single defender resource and then how this changes with the inclusion of multiple defender resources. The defender's effectiveness of a single defender resource 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 resource 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 resources visiting

the same target-time pair provides an additional effectiveness. Given a global state s of defender resources, let $\texttt{eff}(s, b)$ be the effectiveness of the resources 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 resources visiting the same target-time pair to be $1 - (1 - \xi)^k$. This corresponds to the probability of catching the attacker if each resource independently has probability ξ of catching the attacker. Then

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

where $I_{s_i=b}$ is the indicator function that is 1 when $s_i = b$ and 0 otherwise. As more resources 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 resources visit the same target-time pair, b, is that as the attacker observes b, and notices multiple defender resources, this will provide further deterrence of the attacker choosing to target b. If the attacker observes just one defender resource, he can still choose to attack b, by first circumventing one defender resource. However if there are multiple defender resources, 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 resources 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 resources that give an effectiveness value greater than 0 only when paired with another specialized type of defender

resource. The next section explains the defender's pure strategy and the expected utility of both the defender and attacker.

5.3.2 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 resources, then a sample $\boldsymbol{\pi}^j$ includes a policy for defender resource 1 (r_1), and a policy for defender resource 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 resource 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,

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

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

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

The attacker's utility is defined similarly as:

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

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

$$c_b = \sum_j x_j P_b^j$$

be the *marginal* coverage on b by the mixed strategy $\mathbf{x}$ [Yin et al., 2010], 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

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

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.

5.3.3 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 resource 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 resource, which we denote as the qualified defender resource, will be removed from patrolling and allocated to deal with the event once it occurs. This is because certain defender resources have capabilities best suited towards addressing the global event, thereby having the pre-specified, qualified defender resource stop patrolling and handle the global event while the other defender resources 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 resource 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 resource as well as the other defender resources. The qualified defender resource stops patrolling to address the global event while the remaining defender resources may change their strategy and subsequent actions to account for the qualified defender resource 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 resources, i.e., $T_i(\langle s_u, s_i \rangle, a_i, s'_i)$ are dependent on whether the defender resource 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 resource, then the valid joint policy indicates that the qualified defender resource handles the global event and goes out of patrolling duty. If s_u indicates a bomb threat and i is not the qualified defender resource, then resource i would choose an action a_i based on s_u with the knowledge that the qualified defender resource 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.

5.4 Approach

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 5.3). Given the exponential number of defender pure strategies (joint policies) that are needed to solve the LP, we introduce a column generation framework [Barnhart et al., 1994] 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 [Becker et al., 2004; Dibangoye et al., 2012; Petrik and Zilberstein, 2009; Spaan and Melo, 2008], but optimal Dec-MDP algorithms are difficult to scale-up, and hence we use heuristics that leverage ideas from previous work on Dec-MDPs [Varakantham et al., 2009]. 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 5.7.1 or algorithms that attempt to find the optimal solution (which may not suffer from solution quality loss but cannot scale up) in Section 5.7.2 or algorithms that attempt to find even higher quality solutions heuristically (they still fail to perform better) in Section 5.7.3.2.

A standard method for solving Stackelberg games is the Multiple-LP algorithm [Conitzer and Sandholm, 2006]. 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, $\alpha = (t, \tau)$.

$$\max_{\mathbf{c},\mathbf{x}} \quad U_d(\alpha, \mathbf{c}) \tag{5.1}$$

$$U_a(\alpha, \mathbf{c}) \geq \quad U_a(b, \mathbf{c}) \quad \forall b \neq \alpha \tag{5.2}$$

$$c_b - \sum\nolimits_{j \in J} P_b^j x_j \leq \quad 0 \quad \forall b \in B \tag{5.3}$$

$$\sum\nolimits_{j \in J} x_j = \quad 1 \tag{5.4}$$

$$x_j \geq 0 \quad \forall j \in J, \quad c_b \in [0, 1] \quad \forall b \in B \tag{5.5}$$

The LP for α, shown in Equations (5.1) to (5.5), 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 (5.2) enforces that the best response of the attacker is indeed α. In Equation (5.3), $\mathbf{P}^j$ 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:

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

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 (5.3) enforces that given the probabilities x_j of executing mixed strategies π^j, c_b is the marginal coverage of b.

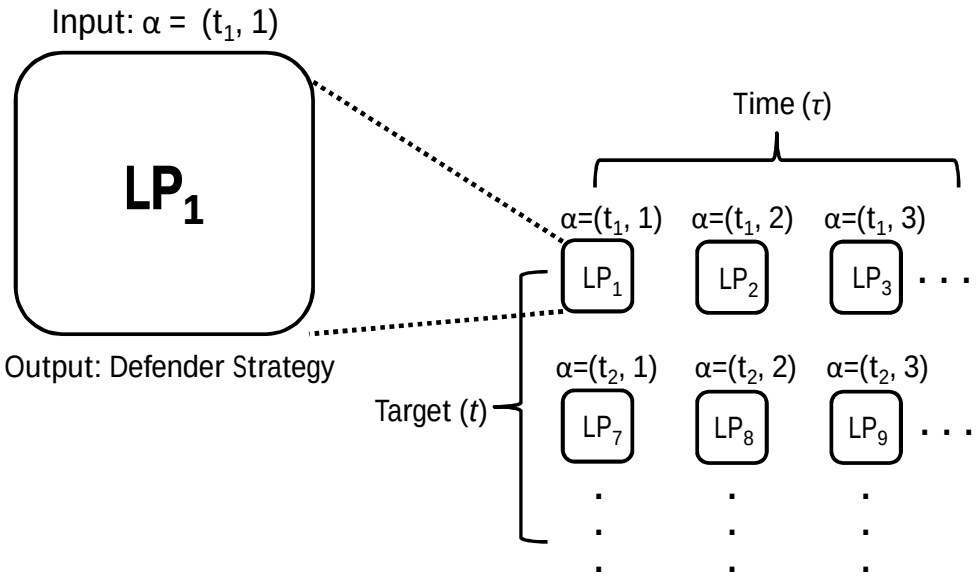


Figure 5.2: Diagram of the Multiple-LP approach

Figure 5.2 gives a diagram of how the Multiple-LP algorithm applies to our solution approach. Focus first on the right side of Figure 5.2. There the figures show several LPs. In particular, this approach generates a separate LP for each attacker pure strategy denoted as α in Equations (5.1) to (5.5). 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 — 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 resources, we will have $(N^T)^R$ policies.

Since this grows exponentially large in proportion to the number of stations, time steps, and defender resources, 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.

5.4.1 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 resources. To deal with this problem, we apply column generation [Barnhart et al., 1994], 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.

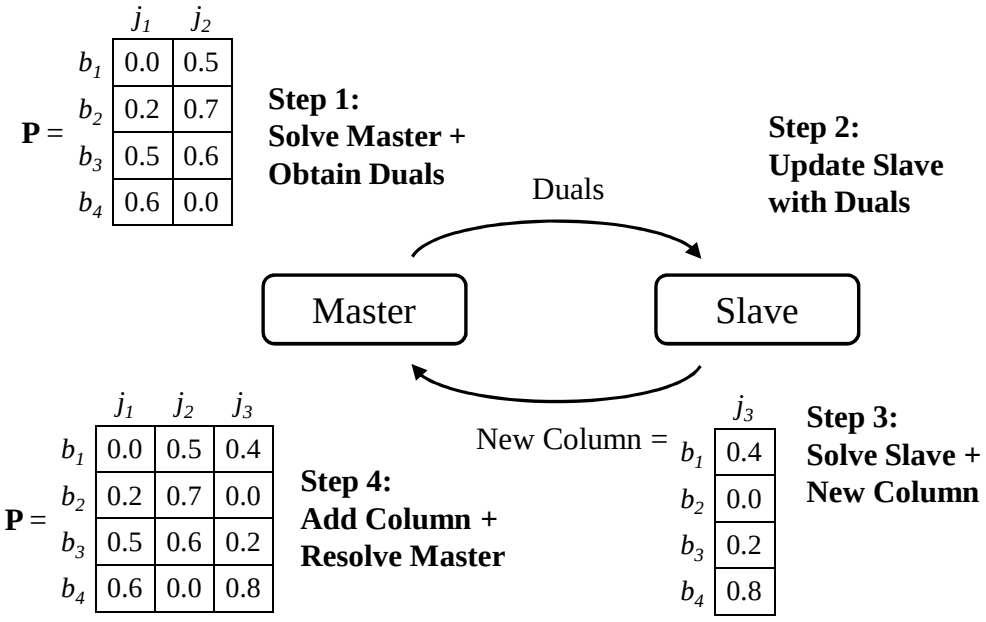


Figure 5.3: Column generation illustration including the master and slave components. The column generation algorithm contains multiple iterations of the master-slave formulation.

Figure 5.3 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 **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 (5.1) to (5.5), 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 [Barnhart et al., 1994], 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:

$$\overline{f}_j = \sum_b y_b \cdot P_b^j - z$$

where z is the dual variable of (5.4) and $\{y_b\}$ are the dual variables of Equation family (5.3), 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.

5.4.2 Dec-MDP Formulation of Slave

We formulate this problem of finding the pure strategy that maximizes reduced cost as a transition independent Dec-MDP [Becker et al., 2004]. 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 resource 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 5.4 shows a sample transition graph showing a subset of the states and actions for resource 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 $T_i((t_1, 0), \text{Visit } t_2, (t_1, 1)) = 0.1$.

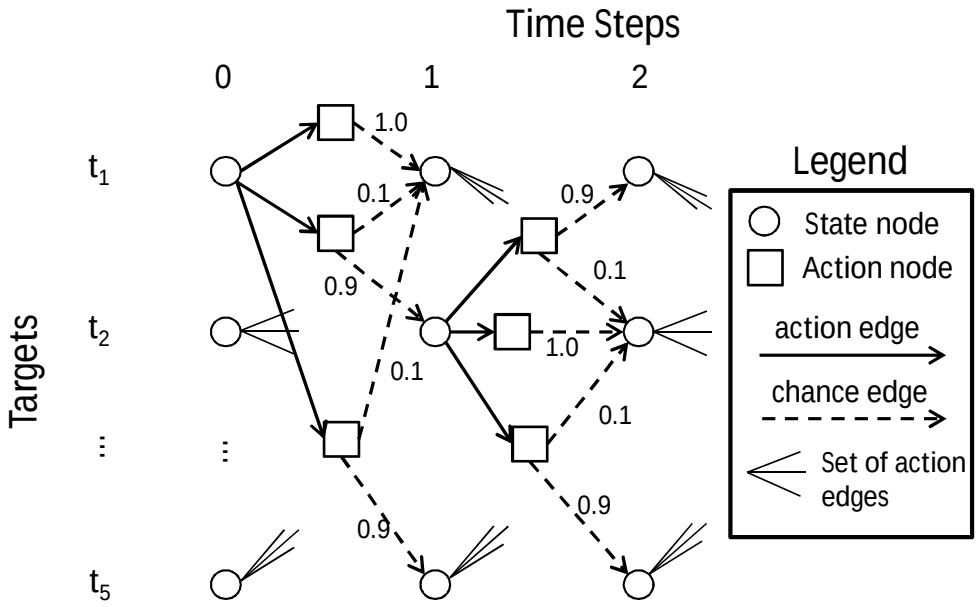


Figure 5.4: Example Transition Graph for one defender resource

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 resources with global state s on target-time pair b, as defined in Section 5.3:

$$R(s) = \sum_b y_b \cdot \texttt{eff}(s, b). \tag{5.6}$$

Multiple transition graphs are needed because each defender resource 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 5.3.2. 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 resource visiting a target-time pair, $b_1 = (t_1, 0)$, and $b_2 = (t_3, 0)$ then:

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

Proposition 2. *Let π^j be the optimal solution of the slave Dec-MDP with reward function defined as in* (5.6)*. 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

$$\begin{aligned}\sum_s \Pr(s|\pi^j)R(s) &= \sum_b y_b \sum_s Pr(s|\pi^j)\texttt{eff}(s, b) \\ &= \sum_b y_b P_b^j = \overline{f}_j + z.\end{aligned}$$

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

5.4.3 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[Spaan and Oliehoek, 2008] along with the MPS algorithm [Dibangoye

et al., 2012] are unable to scale up past four targets and four resources in this problem scenario. Experimental results illustrating this outcome are shown in Section 5.7. 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 5.7 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 2, borrows some ideas from the TREMOR algorithm [Varakantham et al., 2009], which iteratively and greedily updates the reward function for the individual resources 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 resource 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 resources.

The MDP for each resource 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 resource; $T(s_r, a_r, s'_r)$, the transition function of the resource 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 resources that have already been computed from previous iterations.

In more detail, this algorithm takes the dual variables y_b (refer Section 5.4.1) from the master component and $\mathcal{G}$ 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 resource r's states.

Algorithm 2 SolveSlave(y_b, $\mathcal{G}$)

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

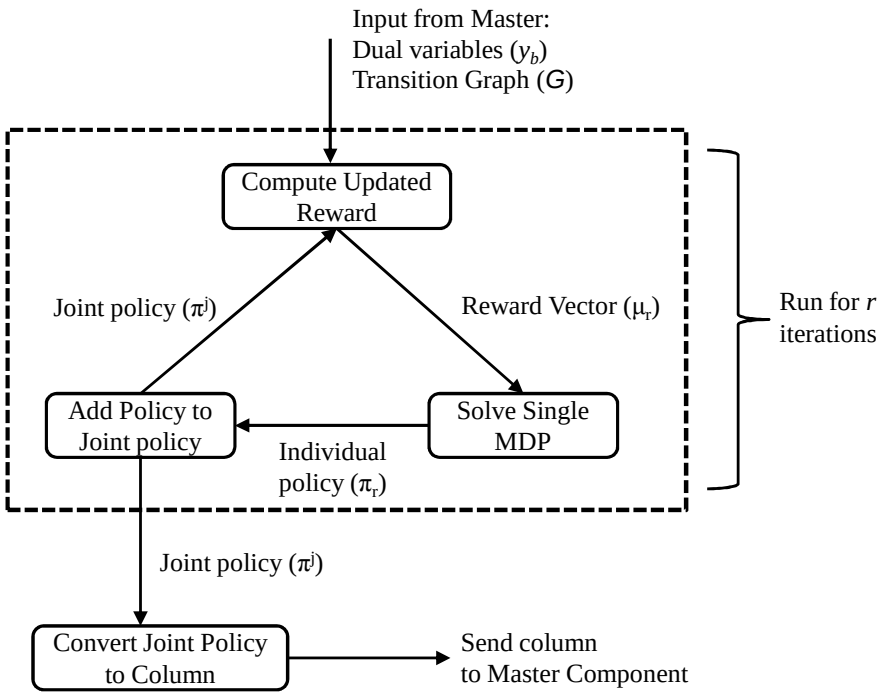


Figure 5.5: Diagram of the algorithm for the slave component

Figure 5.5 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 $\mathcal{G}$. It then solves and generates an individual policy, π_r, for each resource, based on the reward vector. This reward vector takes into account the dual variables from the master along with the individual policies of resources 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 resources $1, \ldots, r$ (with joint reward function (5.6)). The additional reward $\mu_r(s_r)$ for state s_r is the marginal contribution of r visiting s_r to this joint reward, given the policies of the $r-1$ resources 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 $r-1$ resources 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 resource r visits state s_r by computing the effectiveness of resource r visiting state s_r (incorporating the policies of the resources that have already been computed) and subtracting the effectiveness due to just the previous resources and not resource r. For example, if two previously computed resources already visit a state s_r, then if the third resource visits state s_r, the individual reward for the third resource will not be the joint reward of having three resource visit the state, but will instead be the additional effectiveness of having three resources visit the state versus two resources. This avoids double-counting for states that have been visit by other previously computed resources.

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

$$V(s_r, a_r) = \mu_r(s_r) + \sum_{s_r'} T_r(s_r, a_r, s_r') V(s_r')$$

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 resources (in Line 3).

5.5 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 resources, 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 resources 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 resources 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 resources

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 5.4.1, 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 5.5.1. 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 resources in Section 5.5.2. 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 5.5.3).

5.5.1 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 5.6 gives

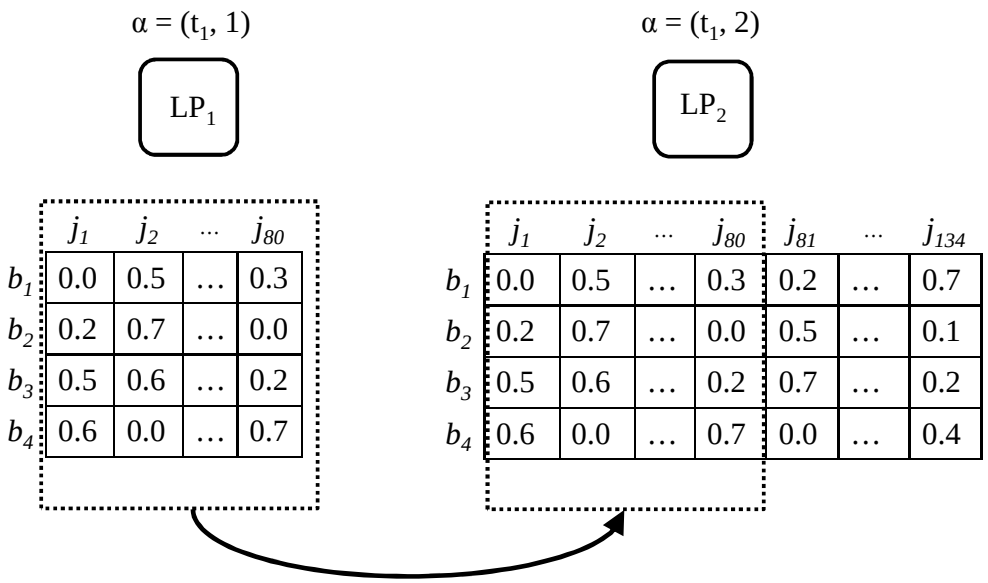


Figure 5.6: Example of the Append heuristic

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 5.2 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 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.

5.5.2 Reducing Runtime for a Single Slave Iteration

The heuristics in Section 5.5.1 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 resources increases. The importance of scaling up to handle defender teams that are comprised of multiple resources is demonstrated in a large scale real-world experiment of security games that had to plan for 23 defender security teams [Fave et al., 2014].

To deal with the inability of the previous heuristics in Section 5.5.1 to handle many defender resources, 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 resources 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 resources 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 2, Line 3). For example, if there are 16 defender resources and each resource 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 resources, or $\binom{16}{1} + \binom{16}{2} + \cdots + \binom{16}{16} = 65,535$ possible combinations of defender resources.

Algorithm 3 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 resources, we used the desiderata as a guideline. We explored setting a limit on the number of resources 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 resources is that the effectiveness for the defender does

not significantly increase when there are already a few defender resources at a state. For example, if a state is already covered by ten defender resources, adding an additional defender resource will not provide a significant increase in effectiveness, compared to the additional benefit if there was just one defender resource and another resource was added. Algorithm 3 gives the algorithm of computing the effectiveness of joint policy π on state b. Algorithm 3 is used in Algorithm 2, for the computation of the updated rewards (Algorithm 2, Line 3) and in transforming the policy that encompasses all resources into a column for the master (Algorithm 2, Line 6). In both cases, we need to enumerate all combinations of resources for each state to compute the effectiveness of the defender resources at each state. The computation of the updated rewards (Algorithm 2, Line 3) is used more expansively in the slave component compared to the conversion of the policy to a column (Algorithm 2, 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 resources 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 resources above the imposed limit, but at the end of the slave calculation (Algorithm 2, Line 6) the column return to the master accurately describes the effectiveness of the joint policy.

Algorithm 3 starts by computing R_s, which is the set of resources that have a non-zero probability of visiting state b (Line 2), by scanning through the policy of each resource to see if there is a possibility of reaching state b. It then iterates from 1 to the total number of resources that have a non-zero probability, or $|R_s|$, of visiting state b. This value of n, represents the number of resources

that visit state b, where the algorithm computes the probability and corresponding effectiveness. In Line 4, the algorithm generates all possible combinations of resources 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 resources. 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 resources at state b, multiplied by the probability of resource 1 and 4 at state b, along with the probability of all other resources 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 $|R_s|$ resources, we place a limit on the maximum number of resources (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 3 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 resources from lines 3 to 7, which grows exponentially large as the number of resources increases. By placing a limit of at most z resources to consider while calculating the effectiveness, we are able to improve the runtime and scale up to a larger number of resources. Despite this limit in calculating the effectiveness, in reality more than z resources may visit this state.

However, when converting the defender's joint policy to a column (Algorithm 2, Line 6), we can compute the exact effectiveness, $\texttt{eff}(s, b)$, by calling Algorithm 3 without placing a limit on the maximum number of resource. In Algorithm 3, 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 5.5, the changes that are made are within the Compute Updated Reward step. This is where the limit is placed on the maximum number of resources 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 resource), this computation does not place a limit on the maximum number of resources 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 resources. 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 5.7.5, is extremely beneficial in speeding up while still providing a high level of solution quality.

5.5.3 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 2, we generate a policy for each resource by iterating over each resource in a single iteration (Line 2). Therefore, the policy of the first resource does not take into account the policies of all other resources. The slave computes the optimal policy for the first resource assuming there are no other resources. The slave component then computes the optimal policy for the second resource given the policy for the first resource (which is now fixed and does not change). The policy of the third resource is computed with the knowledge of the policies of the first two resources. This continues until policies are generated for all resources.

Algorithm 4 SolveRepeatedSlave(y_b, $\mathcal{G}$)

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 resource does not consider the policies of any other resource 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 2, Line 2), we repeatedly iterate Lines 2 - 5, until we reach a local optimum where the policies of the defender resources do not change across iterations.

Algorithm 4 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

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

is used to determine whether the joint policy has changed across iterations. The main difference between Algorithm 4 and Algorithm 2 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 resource r. The updated individual policy for the resource r is then recomputed and re-added to the joint policy. After the individual policies of each resource 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 than the joint policy computed by the single iteration slave. We show this comparison in Table 5.2 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 5.2, 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 resources 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.

5.6 Robustness

There are two types of robustness issues that we explore in this section. 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 5.7.4.1 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 resource 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 5.4.3, the slave generates a policy for each defender resource using value iteration. We present a different way to solve the MDP by using soft-max value iteration (SMVI) [Varakantham et al., 2013a] 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,

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

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

SMVI was first explored for its ability to speed up convergence, as in [Varakantham et al., 2013a]. 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.

5.7 Evaluation

The experiments detailed in 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 resources, 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 5.19, are statistically significant with* $p < 0.01$.

5.7.1 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 resources 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 chapter. 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 resources 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.

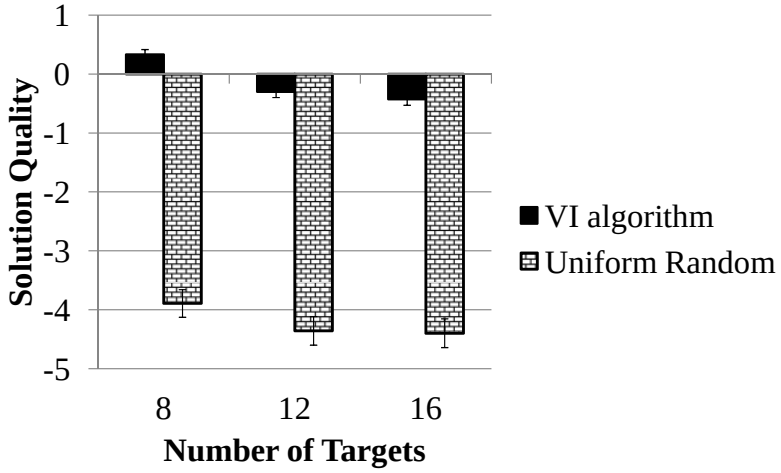


Figure 5.7: Comparison of our VI algorithm versus a uniform random strategy

First, we compare the solution quality of our VI algorithm versus a uniform random strategy in Figure 5.7. 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 5.8 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 resources visit the same state. More specifically, it shows the difference in solution quality between our algorithm that generates policies that takes into account benefit to having multiple resources 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

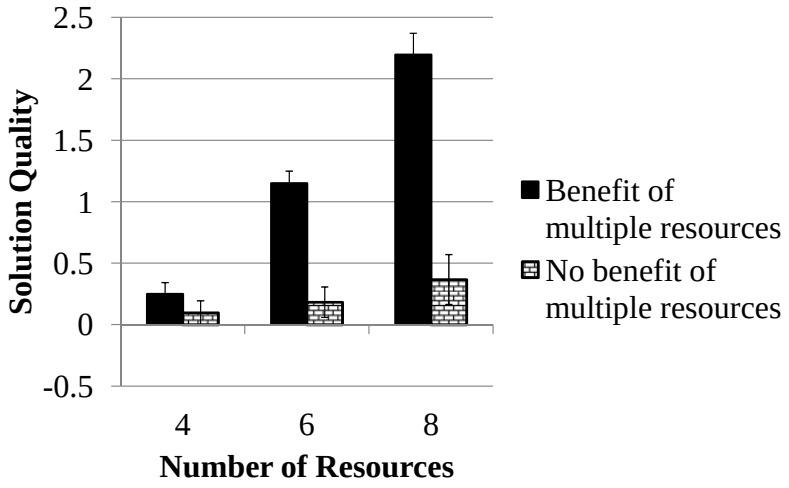


Figure 5.8: Benefit of considering the effectiveness of multiple resources

effectiveness, $\texttt{eff}(s, b) = \xi \cdot I_{b \in s}$ (i.e., it is ξ as long as at least one resource covers b). This algorithm that ignores additional effectiveness is still solving individual MDPs for each defender resource (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 resources visit the same state, the defender receives an additional effectiveness. In other words even for the algorithm that generates policies that ignore multiple resources, when evaluating and computing the defender expected utility, if there is more than one resource at a state, the defender will get an additional effectiveness. As the number of defender resources increases, the solution quality for when there is a benefit to having multiple resources increases at a faster rate than when there is no benefit of multiple resources visiting the same state (no teamwork).

Figure 5.9 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

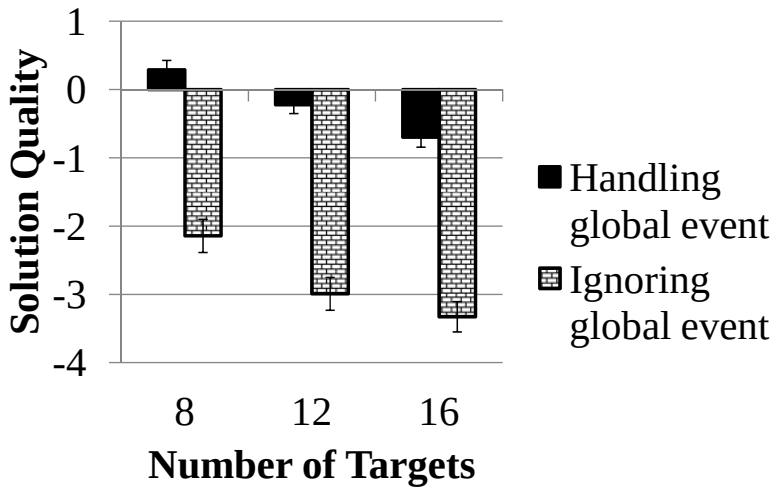


Figure 5.9: Solution quality of handling global events versus ignoring global events

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.

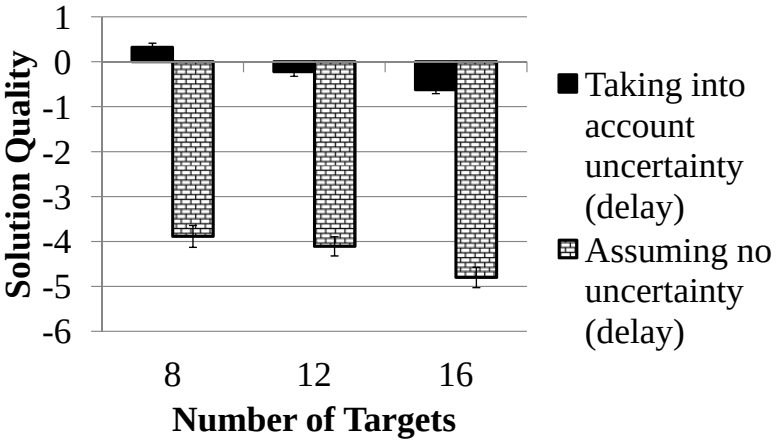


Figure 5.10: Comparison of solution quality taking into account the probability of delay

Figure 5.10 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

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 resources.

5.7.2 Comparison with other Dec-MDP solvers

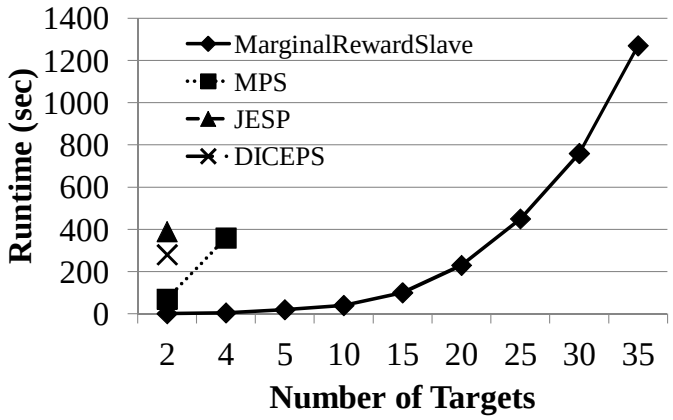


Figure 5.11: Comparison of various Dec-MDP solvers

In Figure 5.11 we compare the runtime of the VI-based slave for one iteration (no column generation) with other algorithms for Dec-MDPs such as MPS [Dibangoye et al., 2012], JESP [Nair et al., 2003] and DICEPS [Oliehoek et al., 2008]—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

[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).

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.

5.7.3 Evaluating Runtime Improvements

This section begins with a comparison among all the runtime heuristics presented in Section 5.5. In Section 5.7.3.1, we show the increased performance when the slave component is modified to place a limit on the number of resources at each state. Then in Section 5.7.3.2 we further explore the impact of the repeated iterative slave.

The first two figures, Figure 5.12 and 5.13 compare the runtime and solution quality across the multiple heuristics as described in Section 5.5. In both figures, the x-axis is the number of targets while the y-axis is the runtime in minutes for Figure 5.12 and the solution quality in Figure 5.13. Focusing on Figure 5.12, the heuristics that provide the fastest runtime are: Append + Cutoff, Append + Cutoff + Ordered, and Max 3 joint resources + Append + Cutoff + Ordered. Both Cold Start and Append take the longest time to run.

When comparing the solution quality, in Figure 5.13, 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

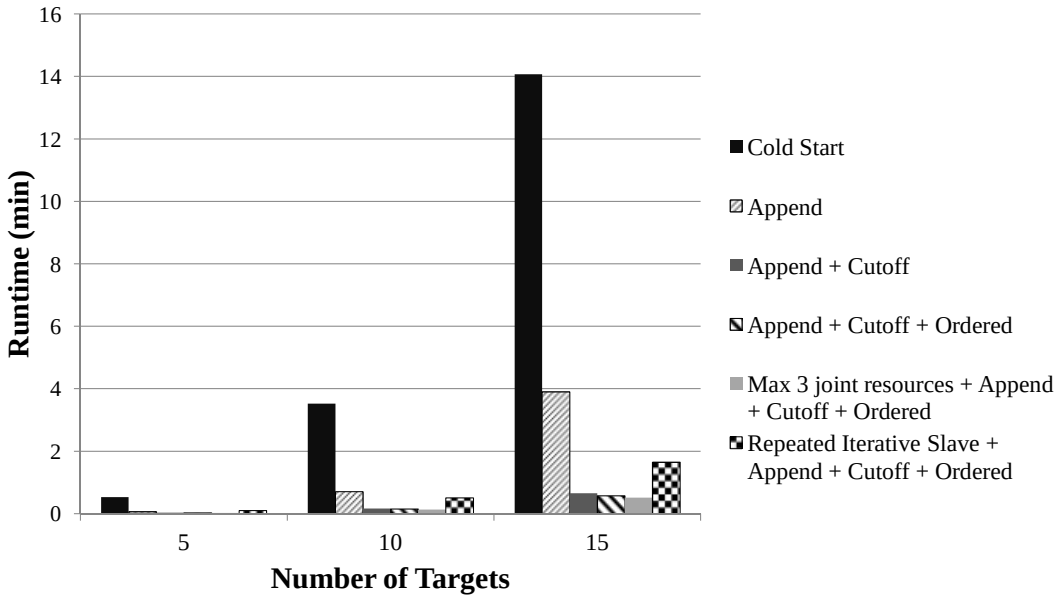


Figure 5.12: Runtime comparison of heuristics

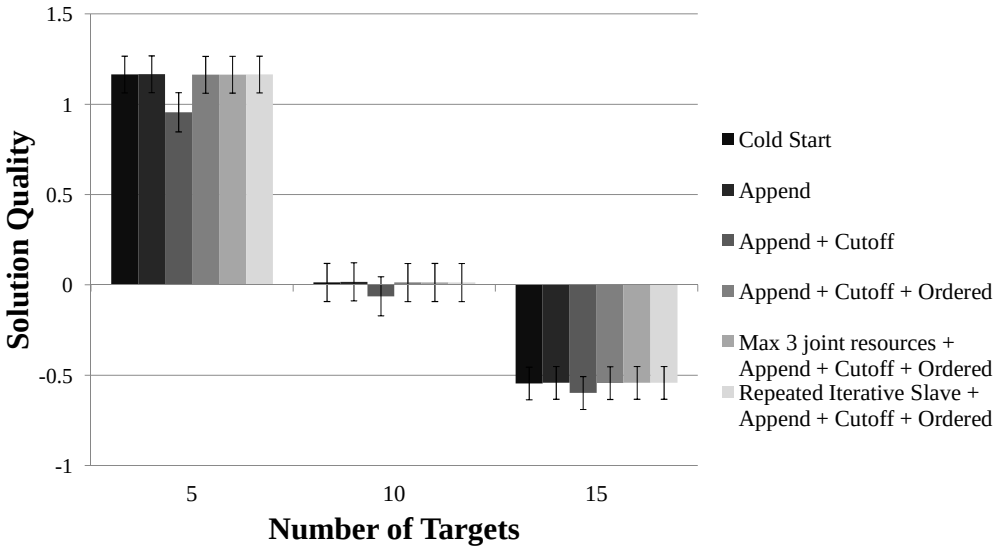


Figure 5.13: Solution quality comparison of heuristics

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

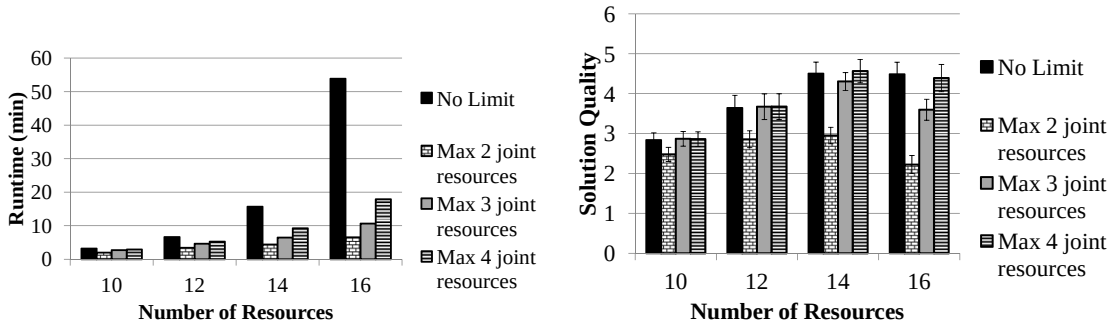
allowing additional columns to be generated and used by the LPs that are solved later in the algorithm to give the defender a larger number of columns or defender pure strategies to use. These two figures show that the Append + Cutoff + Ordered heuristic runs at least as fast as the Cold Start, Append, 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 resources.

5.7.3.1 Maximum resources 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 resources. Figure 5.14 shows the runtime improvements and solution quality when limiting the maximum number of resources 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 resources that can visit a state (equivalent to the number of resources at the same target-time pair). We show the solution quality and runtime when we place a limit of 2, 3, and 4 maximum resources at a state.

Figure 5.14(a) shows the runtime in minutes (y-axis) as the number of resources increases from 10 to 16 (x-axis). Even using the append, cutoff, and ordering improvements, when there are 16 resources, the program takes over 50 minutes to run, compared to under 20 minutes to run when we limit the maximum number of resources at a state to 4. In Figure 5.14(b), the x-axis is the number of resources 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 resources 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

resources that can visit a state, when there are a total of 10 or 12 resources, the solution quality is approximately the same as when we do not place a limit on the number of resources that can visit the same state. However, when the number of resources increases to 14 and 16, placing a limit of only 3 resources results in a loss in solution quality. Overall, these figures show the improvement in runtime for placing limits on the maximum number of resources that can visit a state, while not significantly decreasing in solution quality (e.g., when there is a limit of 4 resources).



(a) Runtime

(b) Solution Quality Comparison

Figure 5.14: Improvements in limiting maximum number of resources

5.7.3.2 Repeated iterative slave

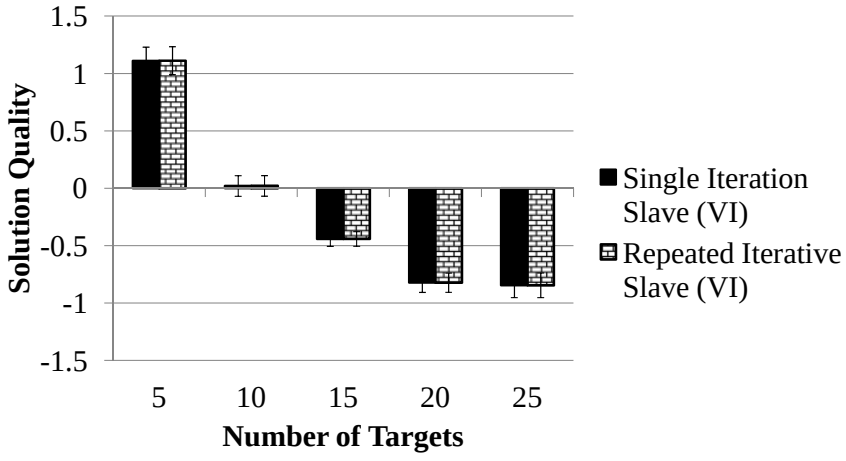


Figure 5.15: Solution quality comparison of the single versus repeated slave

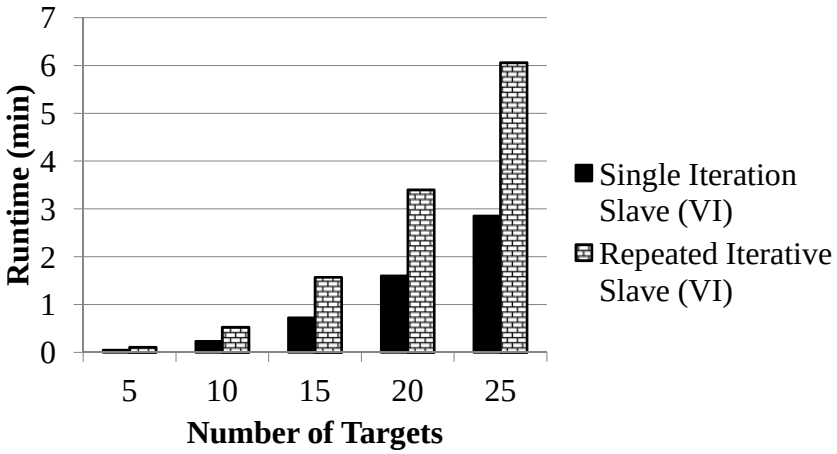


Figure 5.16: Runtime comparison of the single versus repeated slave

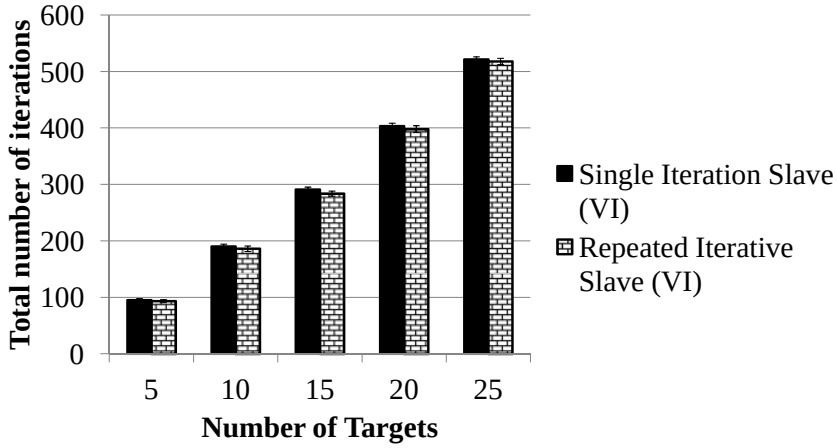


Figure 5.17: Comparison of the number of iterations of the single versus repeated slave

Figures 5.15, 5.16, and 5.17 compare the performance of the single iteration versus the repeated iterative slave algorithm as explained in Section 5.5.3. In all three figures, the x-axis denotes the number of targets. In Figure 5.15, the y-axis is solution quality while in Figure 5.16 the y-axis is the runtime in minutes. For Figure 5.17 the y-axis is the total number of column generation iterations. As mentioned in Section 5.5.3, for initial test cases, the repeated iterative slave computed a higher 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 5.11), going towards a slave that computes a potentially higher quality policy fails to degrade runtime while not improving solution quality (Figures 5.15 and 5.16), and going towards a slave that ignores the uncertainty provides a very low solution quality (Figure 5.10). This does not preclude further improvements to the heuristic slave presented in this chapter, but suggests that such an improvement will require a deeper exploration.

5.7.4 Robustness Experiments

This section explores the levels of robustness of our algorithm. First we test 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.

5.7.4.1 Varying graph structure

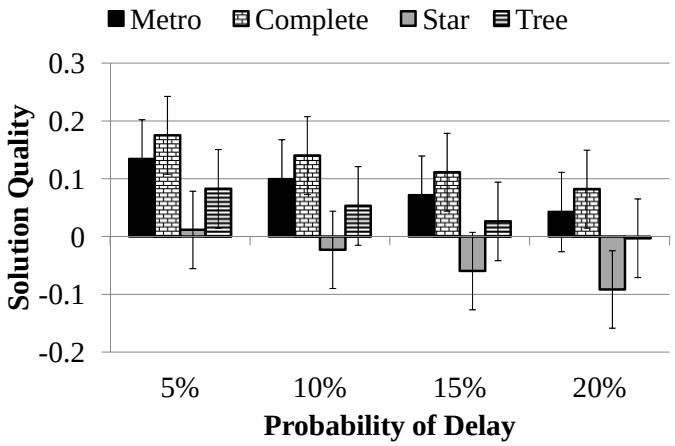


Figure 5.18: Comparison of different graph structures under varying probabilities of delay

In all the prior figures, the graph structure of the targets resemble a metro/train system composed of connected lines. In Figure 5.18, 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 5.7
- 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 resource to visit two leaf targets, the resource 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 resources.

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.

5.7.4.2 Evaluating SMVI and VI

Figure 5.19 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

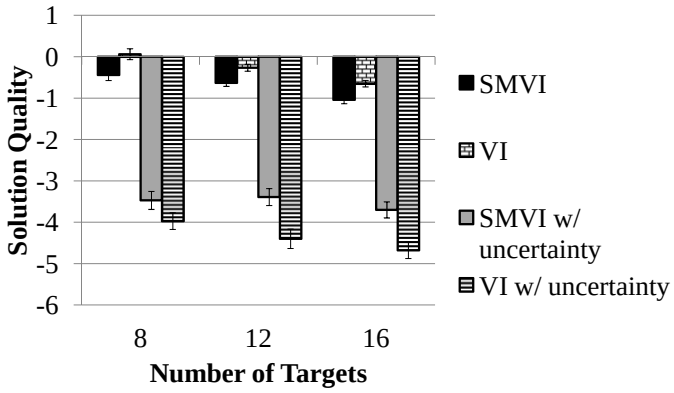


Figure 5.19: Comparison of SMVI and VI under uncertainty in transition probability

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.

5.7.4.3 Varying payoff structure

In Figure 5.19, 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 [Nudelman et al., 2004]. 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.

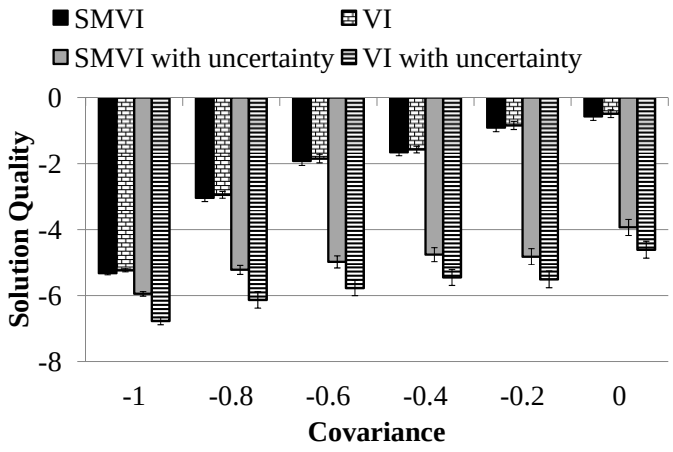


Figure 5.20: Comparison of SMVI and VI under uncertainty in transition probability with varying payoff structures

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 5.20 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 [Breton et al., 1988; Leitmann, 1978; Kiekintveld et al., 2009], 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 5.20, 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.

5.7.5 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 5.3 compares the four primary heuristics/extensions proposed in this chapter. The first heuristic, Append + Cutoff + Ordered, works well for scenarios where the user may have a lot of targets but less than ten defender resources. When there are a significant number of defender resources, the user should choose to use the heuristic that places a maximum number of resources 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 resources increases
Maximum number of resources at a state with Append + Cutoff + Ordered	Scales up as number of resources increases	Have to determine suitable limit for resources 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 5.3: Comparison of Heuristics

5.8 Chapter Summary

The key contribution of this chapter 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 which allows us to decompose the problem into a master and slave component. Within the slave component, we turn to Dec-MDP research to compute defender pure strategies as joint policies. Additionally, we present heuristics to improve the runtime and demonstrate the robustness of using randomized pure strategies.

Chapter 6: Evaluation of Algorithms

This chapter describes my work to evaluate the algorithms developed in modeling and computing defender team strategies in Stackelberg Security Games. First, I provide experiments comparing the SMART$_H$ algorithm against other algorithms in scenarios where there is defender teamwork among multiple resources while assuming no uncertainty. Second, in domains where uncertainty exists for the defender team while executing their respective strategy, I analyze the performance of the value iteration heuristic slave algorithm versus other algorithms. The payoffs for the experiments conducted in this chapter were randomly generated from a uniform distribution between -10 and 10. All experimental results were obtained on a machine with a Dual core 2.8 GHz processor and 4 GB of RAM, and were averaged over 100 trials.

The main focus of the additional experiments in this chapter is to contribute further analysis into the quality of the strategies generated by the algorithms proposed in this thesis. Computing the optimal solution or providing approximation guarantees for the quality of the output generated by the algorithms is important. Many prior work on security games have explored computing the optimal strategy for the defender in scaling up to large state spaces [Jain et al., 2010a,b, 2011; Kiekintveld et al., 2009], in modeling a boundedly rational adversary [Pita et al., 2010, 2012; Jiang

et al., 2013a; Yang et al., 2012, 2013], and in handling uncertainty [Brown et al., 2014a; Nguyen et al., 2014a,b; Qian et al., 2015; Yin and Tambe, 2012; Yin et al., 2011].

Unfortunately, some of the algorithms presented my thesis are not guaranteed to find the optimal defender solution nor am I able to provide approximation guarantees on the solution quality. However, these algorithms are necessary to be able to solve security games with teamwork among multiple defender resources for real-world scenarios. Therefore, in this chapter, I include additional experiments on these heuristic algorithms to provide upper and lower bounds on their performance. I conclude this chapter with a summary of the evaluations of the algorithms presented in this thesis.

6.1 Analysis of SMART$_H$

In this section, I start by comparing the SMART$_H$ algorithm versus the optimal algorithm (SMART$_O$) in Section 6.1.1. Since the optimal algorithm is unable to scale up to more than 5 targets, I compare the solution quality of the SMART$_H$ algorithm versus the computed upper bound from ORIGAMIP. I then compare the performance of SMART$_H$ versus two other heuristic algorithms that are also able to generate defender strategies for more than 5 targets.

6.1.1 SMART$_H$ versus SMART$_O$

In this experiment, we compare the solution quality, or defender expected utility, of SMART$_H$ versus SMART$_O$ while varying the number of targets. The idea is to understand the quality of the solutions recovered by the more "practical" algorithm, SMART$_H$, in comparison with the optimal solutions generated by SMART$_O$. In the following three tables, Table 6.1, 6.2, and 6.3, I compare the solution

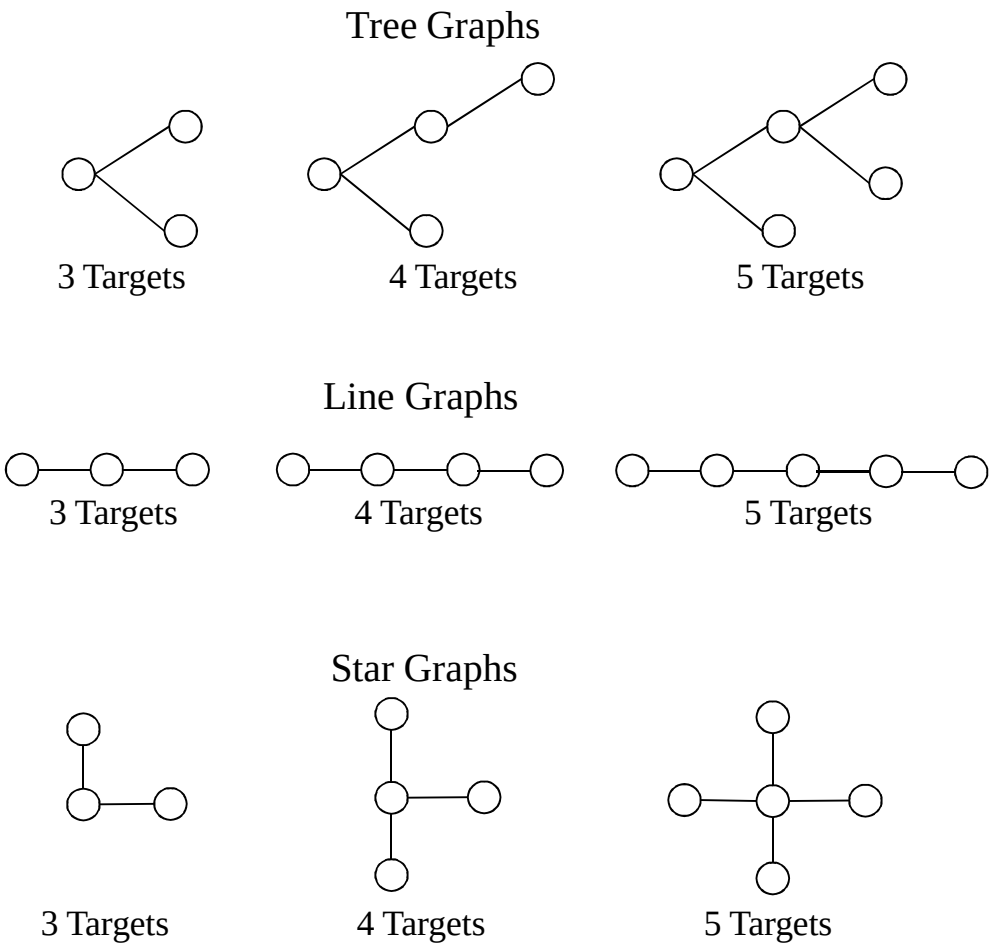


Figure 6.1: Diagram of the three different graph structures tested for 3, 4, and 5 targets.

quality of SMART$_H$ versus SMART$_O$ for varying types of graph structures. The results shown in all three tables represent the defender expected utility averaged over 100 game instances with random payoffs in the range [-10, 10]. In all the experiments, the number of defender resources is kept constant to two defender resources. For the game instances, the graph is fixed (depending on which structure) with just the target payoffs changing over each problem instance. Figure 6.1 shows the graph structure for each of the of the different graphs tested for 3, 4, and 5 targets.

	SMART$_H$	SMART$_O$	Number of scenarios with different solution quality	Average absolute difference	Average percentage difference (%)	Maximum difference
3 targets	3.001	3.080	21	0.078	2.2	0.657
4 targets	-0.862	-0.788	18	0.074	7.5	1.114
5 targets	-0.924	-0.910	19	0.015	1.5	1.06

Table 6.1: Solution Quality and Metrics of SMART$_H$ vs. SMART$_O$ for Tree Graphs

	SMART$_H$	SMART$_O$	Number of scenarios with different solution quality	Average absolute difference	Average percentage difference (%)	Maximum difference
3 targets	2.748	2.810	28	0.062	2.0	0.504
4 targets[1]	0.532	0.532	0	0.0	0.0	0.0
5 targets[1]	0.171	0.171	4	0.0	3.5	0.6

Table 6.2: Solution Quality and Metrics of SMART$_H$ vs. SMART$_O$ for Star Graphs

	SMART$_H$	SMART$_O$	Number of scenarios with different solution quality	Average absolute difference	Average percentage difference (%)	Maximum difference
3 targets	2.967	3.063	30	0.097	2.2	0.784
4 targets	-1.067	-0.988	20	0.079	2.6	0.879
5 targets	-1.740	-1.703	15	0.037	2.2	0.667

Table 6.3: Solution Quality and Metrics of SMART$_H$ vs. SMART$_O$ for Line Graphs

[1]The maximum time of the patrol length was decreased to 30 minutes for these game instances so SMART$_O$ would run and generate a solution. Increasing the patrol length to longer than 40 minutes would result in SMART$_O$ failing to run and generate the optimal solution.

For all three tables, the first column provides the solution quality for SMART$_H$, the second column provides the solution quality for SMART$_O$ (optimal solution), the third column represents the number of scenarios (game instances) where SMART$_H$ computed a different solution quality compared to SMART$_O$, the fourth column computes the average difference in solution quality between SMART$_H$ and SMART$_O$, the fifth column displays the average percentage difference in solution quality of SMART$_H$ versus SMART$_O$, and the sixth (final) column provides the maximum difference in solution quality for SMART$_H$ versus SMART$_O$.

The results shown in Table 6.1 represent the defender expected utility for a tree graph, with the root of the tree corresponding to the home base and each parent having at most two children. When testing for 3 targets, on average SMART$_H$ generated a solution quality of 3.001 while SMART$_O$ generated an average solution quality of 3.080, with 21 out of the 100 game instances resulting in both the algorithms computing a different solution quality. In all three scenarios (comparing from three to five targets), the average difference in solution quality of SMART$_H$ versus SMART$_O$ is less than 0.08. On average for the tree graphs, in 80% of the problems, SMART$_H$ achieved the same solution quality as SMART$_O$. The average absolute difference in solution quality decreases from 0.078 for 3 targets, to 0.074 for 4 targets, and finally to 0.015 for 5 targets. So as the number of targets increases, the average absolute difference decreases. We see a similar trend for star and line graphs.

Table 6.2 represent the solution quality for a star graph. In the star graph, the home base target is at the center, and all other targets are leaf nodes that are connected to the home base target. When testing for more than 3 targets, the maximum patrol time was decreased to 30 minutes so SMART$_O$ would be able to run and produce an output. With a shorter patrol time and running both algorithms for four targets, both SMART$_H$ and SMART$_O$ generated the same exact solution quality

over all of the 100 game instances. When increasing the number of targets to five, only four out of the 100 game instances resulted in different solution qualities for SMART$_H$ and SMART$_O$. The reason why SMART$_H$ performs well compared to the tree and line graphs, could be due to the decrease in maximum patrol time. As this value is decreased, the state space decreases and thus there are less possible valid patrols for each defender resource. This constraint may help SMART$_H$ be able to find the optimal solution.

The results shown in Table 6.3 represent the solution quality for a line graph, where the graph of the targets resembles a single line. As the number of targets increases, the solution quality decreases as there are more targets that need to be covered. Another trend is that as the number of targets increases, the number of game instances where SMART$_H$ gives a different defender expected utility than SMART$_O$ decreases. For five targets, the average solution quality for SMART$_H$ is only 0.037 less than the average solution quality for SMART$_O$. Similar to tree graphs, for line graphs, SMART$_H$ generates the optimal solution quality approximately 80% of the time. Similar to tree graphs, the average absolute difference between SMART$_O$ and SMART$_H$ decreases as the number of targets increases, with the average difference being 0.097 for 3 targets, but then decreasing to 0.037 for 5 targets. In addition, as the number of targets increases, SMART$_H$ generates the optimal solution a higher percentage of the time, from 70% of the time for 3 targets to 85% of the time for 5 targets.

Therefore, even when varying over different graph types, SMART$_H$ computes a solution which is, on average, very close to SMART$_O$. Based on these experiments, SMART$_H$ generates the optimal solution approximately 80% of the time. For all graph structures, as we increased the number of targets, the number of game instances with differences in solution quality actually goes down. We also see a similar trend for the average absolute difference in solution quality, that this value

decreases as the number of targets increases. A rationale for this is that as we increase the number of targets, but keep the maximum patrol time constant, there may be less variety in the types of patrols and activities that can be performed by the defender resource in maximizing each patrol. For example, if there are only 3 targets, then the defender resource has more time to execute activities that give the defender a higher effectiveness. As the number of targets increases, more time must be devoted to visiting all the targets and thus decreases the different types of activities that can be performed at each target.

I also examined the individual game instances where the optimal solution quality was different than the solution quality computed by SMART$_H$ to see if there was any trend into the scenarios where SMART$_H$ fails to find the optimal solution. I looked at the defender payoffs (e.g., when there are larger gaps in the reward and penalties versus smaller gaps), attacker payoffs, defender strategy (e.g., how many defender pure strategies receive a non-zero probability), and size of the support set to investigate why some scenarios the SMART$_H$ algorithm is able to find the optimal solution while in other situations it is generates a lower solution quality. However, there was no noticeable pattern to ascertain when SMART$_H$ would compute an optimal solution.

Unfortunately, we are unable to run SMART$_O$ for greater than five targets. When running SMART$_O$ for six targets, the program did not run more than two iterations of the column generation approach within the 10 hour run-time limit placed on the program. To measure the performance of SMART$_H$ for more than five targets, I looked towards comparing SMART$_H$ versus the upper bounds value computed by ORIGAMIP in the following section.

	SMART$_H$	Upper Bound
10 targets	1.241	1.887
15 targets	0.534	0.840
20 targets	0.382	0.910

Table 6.4: Solution Quality of SMART$_H$ vs. Upper Bounds for Tree Graph

	SMART$_H$	Upper Bound
10 targets	2.149	2.400
15 targets	1.489	1.800
20 targets	1.740	1.799

Table 6.5: Solution Quality of SMART$_H$ vs. Upper Bounds for Star Graph

	SMART$_H$	Upper Bound
10 targets	0.444	1.258
15 targets	-0.301	0.446
20 targets	0.841	1.815

Table 6.6: Solution Quality of SMART$_H$ vs. Upper Bounds for Line Graph

6.1.2 SMART$_H$ compared to Upper Bounds from ORIGAMIP

Since SMART$_O$ is unable to solve problem instances of more than five targets, I turned to using the upper bounds as computed by the ORIGAMIP algorithm (defined in Section 4.2.2) to understand and measure the performance of the solution quality computed by SMART$_H$. As explained in Section 4.2.2, ORIGAMIP provides an over-estimate for the effectiveness of a defender patrol on a target by adding the effectiveness of all individual activities on a target.

Tables 6.4, 6.5, and 6.6 compare the solution quality returned by the SMART$_H$ algorithm with the upper bound value computed by ORIGAMIP for tree, star, and line graphs, respectively. There is a much larger difference between the upper bound from ORIGAMIP versus the solution quality computed by SMART$_H$, compared to the difference between SMART$_O$ and SMART$_H$. This is because the upper bound computed in ORIGAMIP allows the defender to potentially have a higher effectiveness than is possible (and that can be computed in SMART$_H$).

For some experiments, the solution quality compute by SMART$_H$ is close to the upper bounds computed from ORIGAMIP, like for star graph with 10 targets (difference of 0.251) or 20 targets (difference of only 0.059). However, for other scenarios, the gap in value between the solution quality from SMART$_H$ and the upper bound is larger. For example, in the line graph, with 20 targets, the difference in value is 0.974, or in the tree graph with 10 targets, the difference in value is 0.646. While the upper bounds computed from the ORIGAMIP algorithm does give a rough measure of the quality that is computed by SMART$_H$, I next focus on other scalable heuristic algorithms to provide a lower bound in the solution quality.

6.1.3 SMART$_H$ versus other scalable heuristic algorithms

Given the complexity of optimally computing a solution to handle joint activities within defender teams in SSGs, I look to other scalable heuristic based algorithms to compare against SMART$_H$, to provide lower bounds and benchmarks while also demonstrating the quality of the strategy that SMART$_H$ computes.

The first algorithm is the emphSampled Subgame algorithm, that generates random pure defender strategies by solving a subgame of the problem. The Sampled Subgame algorithm computes a set of pure defender strategies for each resource by solving the slave component as described in Section 4.3, but with the difference of using random rewards instead of rewards based on the duals from the master component (See Equation 4.11 for more details about the duals/input from the master component to the slave component). In other words, instead of finding the next best defender team strategy within the slave component, based on the duals form the master component (what SMART$_H$ does), the Sampled Subgame approach uses random inputs to solve the slave component, to generate feasible defender team strategies. Since random rewards

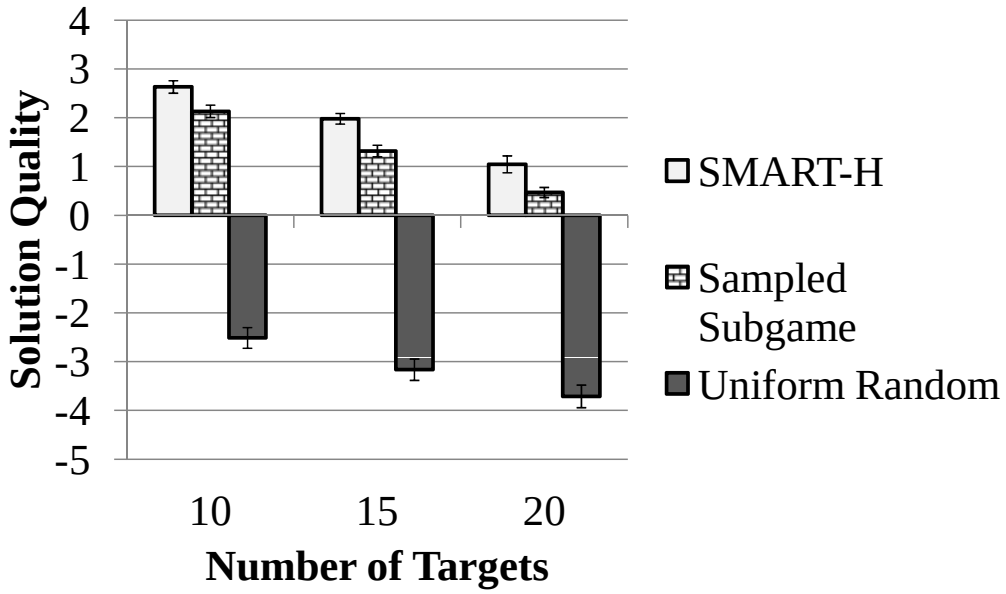


Figure 6.2: Comparison of SMART$_H$ versus other algorithms while varying the number of targets.

are used in the Sampled Subgame algorithm, the master can no longer use the reduced cost to determine when to terminate and thus the slave is run for a fixed amount of iterations to generate a subset of joint policies. The master LP is solved with the subset of joint policies in finding the defender's strategy and corresponding expected utility.

The second algorithm is a uniform random approach where the defender has a set of possible defender team patrols and has a uniform random probability of choosing each defender team patrol strategy.

I provide two experiments comparing SMART$_H$ versus both the Sampled Subgame and Uniform Random algorithms by varying the number of targets and maximum patrol time. Both the Sampled Subgame and Uniform Random algorithms generated a fixed set of 200 defender pure strategies. The SMART$_H$ algorithm on average generated less than 100 defender pure strategies. The main focus of these two experiments is to show the solution quality/defender expected utility that is computed by SMART$_H$ in larger scale games against other heuristics that are also able to scale up.

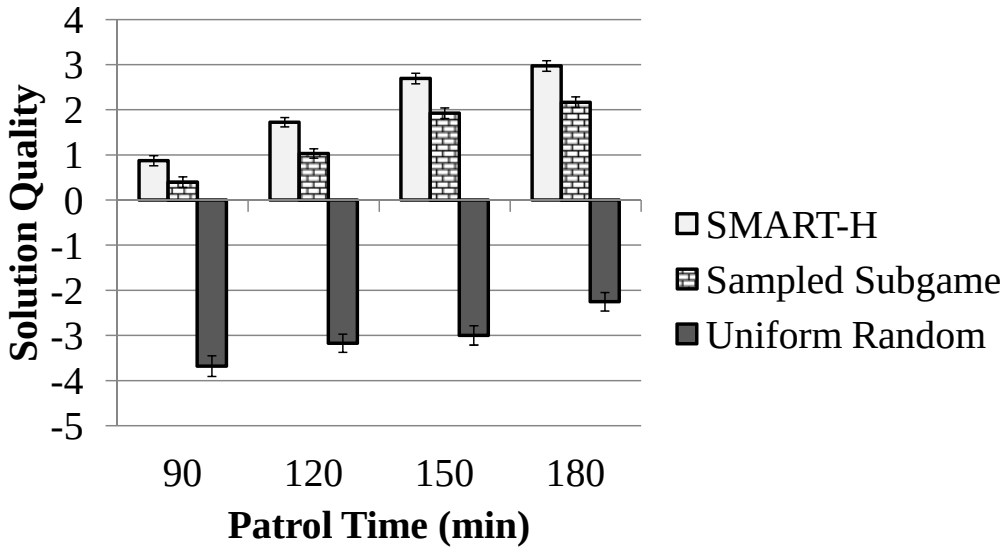


Figure 6.3: Comparison of SMART$_H$ versus other algorithms while varying the maximum patrol time.

Figure 6.2 shows the performance of all three algorithms when the number of targets increases. The x-axis shows the number of targets, while the y-axis shows the solution quality or defender expected utility. As can be seen in this figure, the strategy computed by the SMART$_H$ algorithm provides a significant increase in solution quality compared to the other two algorithms. For example, SMART$_H$ generates a strategy that gives a defender expected utility on average of 1.05 for a Stackelberg Security Game with 20 targets, whereas the Sampled Subgame approach provides a defender strategy that only gives a defender expected utility of 0.469, and the Uniform Random approach gives a defender expected utility of -3.71. In this figure, the solution quality for all three algorithms decrease as the number of targets increase, as there are additional targets that need to be covered while keeping the number of resources and maximum patrol time constant.

Figure 6.2 shows the performance of all three algorithms when varying the maximum patrol time. The x-axis shows the maximum patrol time in minutes, while the y-axis shows the solution quality or defender expected utility. Consistent with the previous figure, the SMART$_H$ algorithm outperforms the Sampled Subgame and Uniform Random algorithms for all lengths of patrol time.

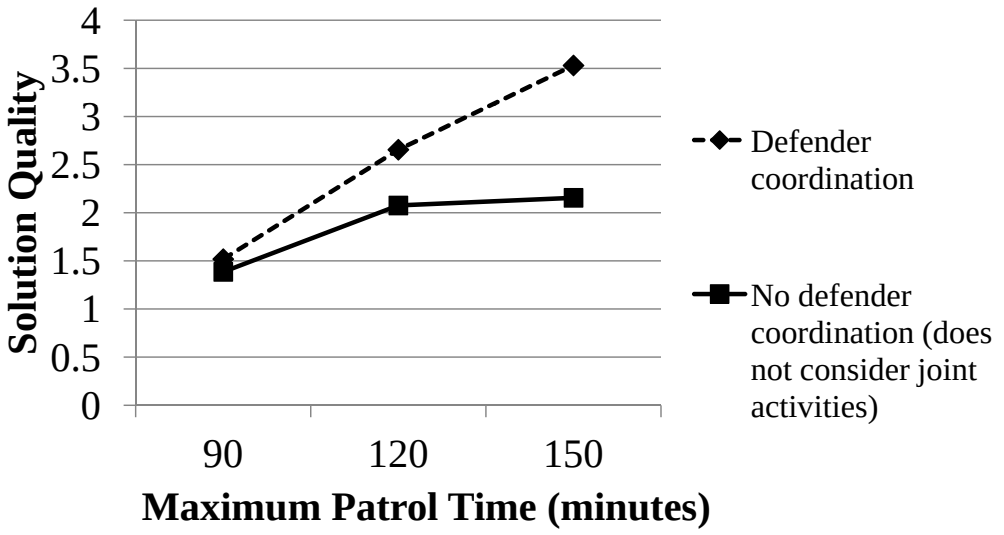


Figure 6.4: Solution quality of an algorithm that incorporates defender coordination in the form of joint activities versus an algorithm that does not consider joint activities.

6.1.4 SMART$_H$ versus no defender coordination

In addition to comparing SMART$_H$ against other scalable algorithms, this section presents results of SMART$_H$ versus an algorithm that does not consider defender coordination, or the increased effectiveness from joint activities. The purpose of this comparison is to demonstrate the importance of defender teamwork in SSGs and that simply implementing an algorithm that ignores joint activities will significantly decrease the defender's expected utility.

As seen in Figure 6.4, taking into account defender teamwork in the form of joint activities is important and simply using an existing algorithm that ignores joint activities among multiple defender resources will result in a lower defender expected utility. In this figure, the x-axis shows the maximum patrol time in minutes, while the y-axis shows the solution quality. The algorithm that does not consider joint activities generates a patrol strategy that assumes no benefit to joint activities, but when computing the defender expected utility, if there does happen to be a joint activities, the defender still receives an increased effectiveness from having multiple defender resources visit the same target. The algorithm that considers defender coordination via joint

activities corresponds to the SMART$_H$ algorithm. When the maximum patrol time is just 90 minutes, then the difference in solution quality is only about 0.15, compared to when the maximum patrol time increases to 150 minutes where there is an improvement of over 1.25 in the defender's expected utility when considering joint activities.

6.2 Analysis of Value Iteration Heuristic Slave

This section focuses on evaluating the value iteration (VI) heuristic slave (see Section 5.4.3) versus other heuristic algorithms solving Stackelberg Security Games under both defender teamwork and execution uncertainty. Given the increased complexity in solving SSGs when taking into account defender teamwork and execution uncertainty, we compare against other heuristic algorithms to provide lower bounds on our VI heuristic slave algorithm.

Similar to the two algorithms introduced in Section 6.1.3, I modified both the Sampled Subgame and uniform random algorithms to use the updated slave as described in Section 5.4.3, to solve MDPs and generate a policy for each defender resource that deals with execution uncertainty. The Sampled Subgame algorithm now generates a fixed set of defender team strategies that include a policy for each resource. I compare these three algorithms as the input problem size and parameters are varied.

Vary Number of Targets: Figure 6.5 presents the results when varying the number of targets in the input problem. The x-axis shows the number of targets, whereas the y-axis shows the solution quality (defender expected utility). The three bars in the graph compare the solution quality of the VI heuristic slave (labeled as Iterative-based Slave), Sampled Subgame, and uniform random. These results show that the VI heuristic slave is better than both the Sampled Subgame

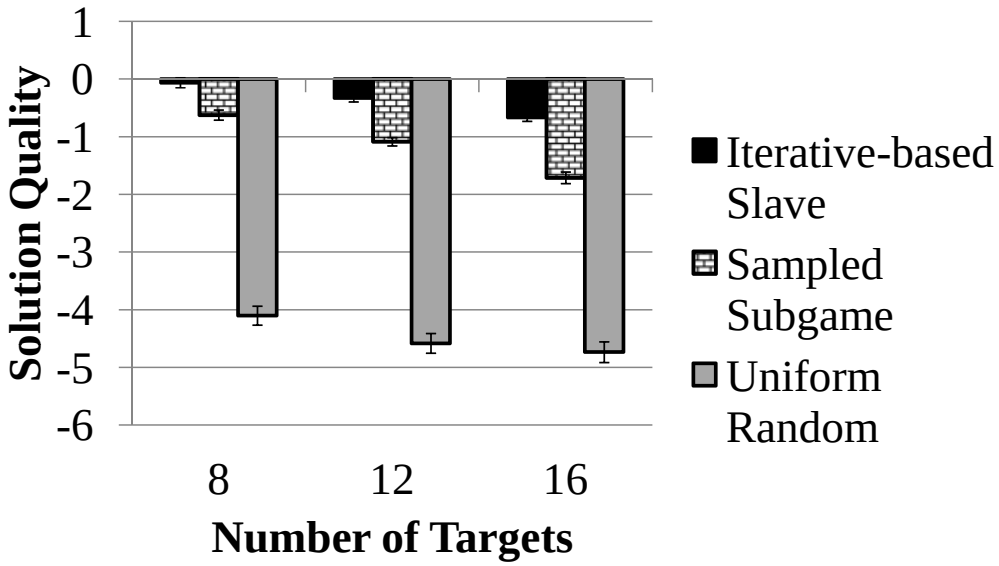


Figure 6.5: Varying the number of targets.

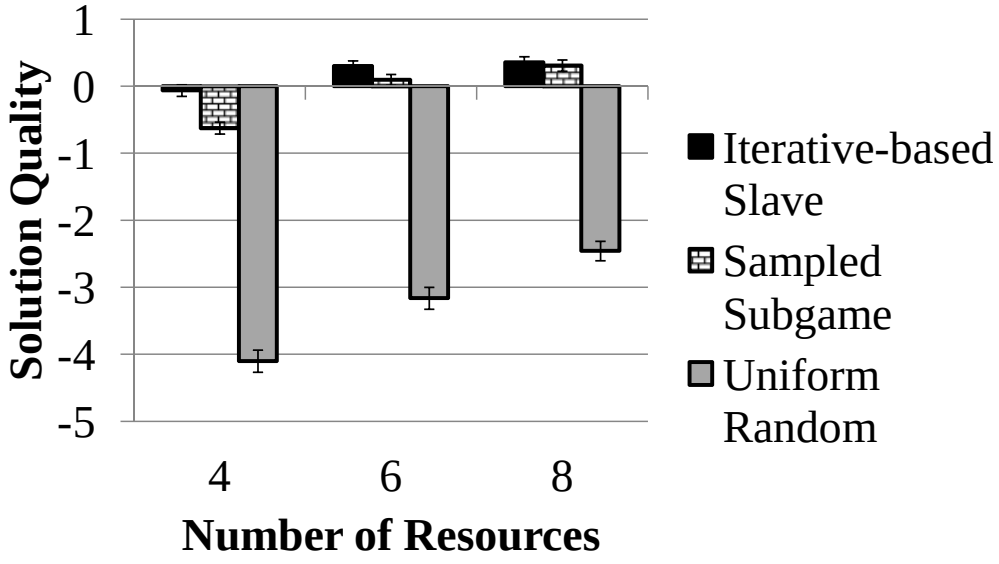


Figure 6.6: Varying the number of defender resources.

and uniform random algorithms across all numbers of targets. For example, for 16 targets, the VI heuristic slave generated a defender team strategy that results in a defender expected utility of -0.67 compared to the defender expected utility of -1.71 for the Sampled Subgame approach and -4.74 for the uniform random strategy.

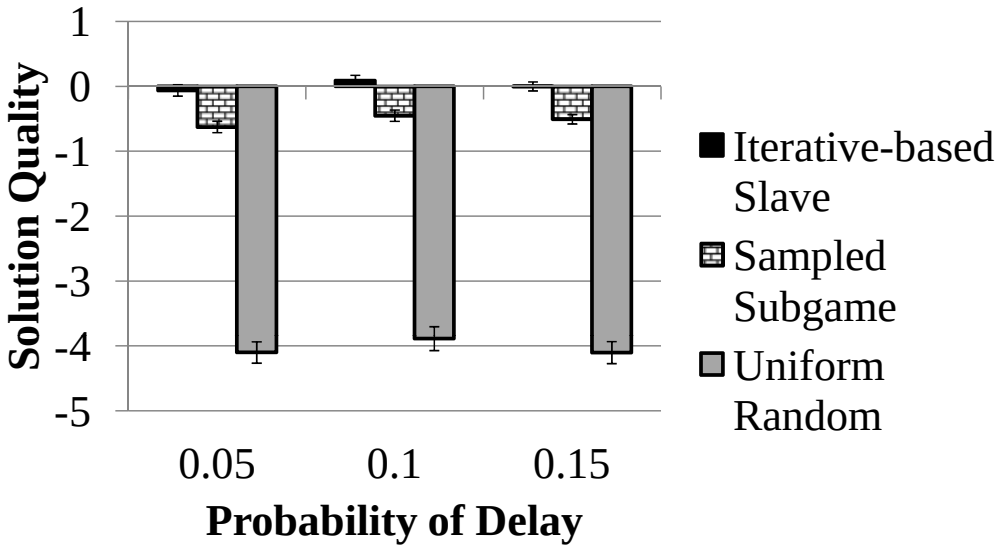


Figure 6.7: Varying the probability of delay.

Vary Number of Defender Resources: Figure 6.6 presents the results when varying the number of targets in the input problem. The x-axis shows the number of defender resources, whereas the y-axis shows the solution quality (defender expected utility). These results demonstrate that the VI heuristic slave outperforms both the Sampled Subgame and uniform random algorithms across varying number of defender resources. For example, for 4 defender resources, the VI heuristic slave generated a strategy that gives a defender expected utility of -0.06 compared to the defender expected utility of -0.63 for the Sampled Subgame approach and -4.10 for the uniform random strategy.

Vary Number of Probability of Delay: I now present the results when varying the probability of delay in Figure 6.7. The x-axis shows the probability of delay ranging from 5% to 15% delay, whereas the y-axis shows the solution quality (defender expected utility). These results show that the performance trend remains the same: the VI heuristic slave outperforms both the Sampled Subgame and uniform random algorithms. For example, when the probability of delay is 0.1 or 10%, the VI heuristic slave generated a strategy that gives a defender expected utility of 0.09

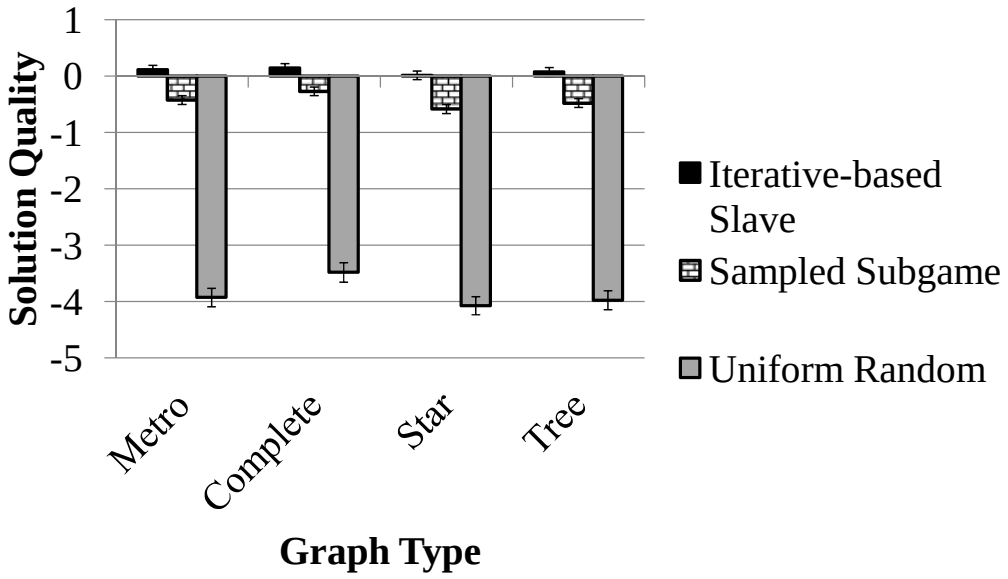


Figure 6.8: Varying the graph type for non-zero-sum games.

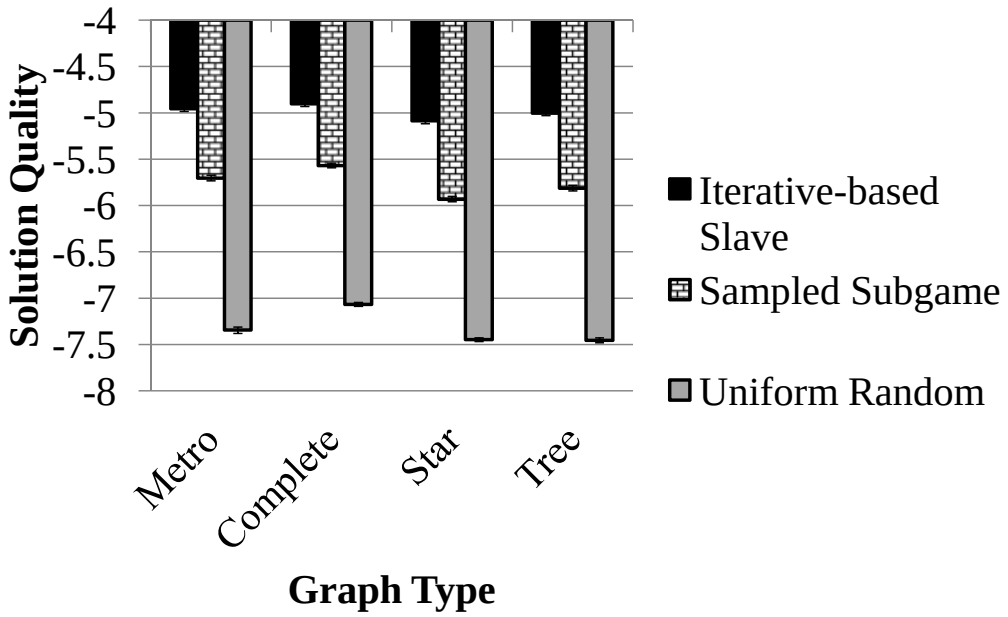


Figure 6.9: Varying the graph type for zero-sum games.

compared to the defender expected utility of -0.45 for the Sampled Subgame approach and -3.89 for the uniform random strategy.

Vary Graph Type and Payoffs: I now present the results when varying the graph type while also comparing the performance for non-zero-sum games and zero-sum games in Figures 6.8 and

6.9. The x-axis for both of these figures shows the different graph types, whereas the y-axis shows the solution quality (defender expected utility). The four graph types that were examined are described in detail in Section 5.7.4.1.

For both varying the graph structure and non-zero-sum/zero-sum games, the results show that the performance trend remains consistent: the VI heuristic slave outperforms both the Sampled Subgame and uniform random algorithms. For example, in Figure 6.9, for a complete graph and zero-sum payoffs, the VI heuristic slave computes a defender team strategy with a solution quality of -4.9 while the Sampled Subgame algorithm has a solution quality of -5.6 and the uniform random algorithm results in a solution quality of -7.1.

6.3 Summary of Evaluations

In this section, I summarize all the different types of evaluation that I have conducted in measuring the performance of SMART$_H$ for SSGs that handle teamwork, and the VI heuristic slave for SSGs that address execution uncertainty among multiple coordinating defender resources.

6.3.1 Evaluation of SMART$_H$

- Comparison with optimal algorithm SMART$_O$ for 3, 4, and 5 targets for different graph types shows that SMART$_H$ computes the optimal solution approximately 80% of the time with the average difference in solution quality decreasing as the number of targets increases.
- Comparison with two scalable heuristic algorithms (Sampled Subgame and Uniform Random) provides lower bounds on solution quality.
- Outperforms an algorithm that does not consider coordination among defender resources.

- Real-world test on the train domain goes beyond simulations and uses real-world data to evaluate the schedule generated by SMART$_H$ versus a manual schedule, and show the improved level of safety and effectiveness of SMART$_H$.

6.3.2 Evaluation of VI heuristic slave

- Comparison with two scalable heuristic algorithms (Sampled Subgame and Uniform Random) while varying targets, resources, probability of delay, and graph type to provide lower bounds and benchmarks for solution quality.
- Performs better than an algorithm that solves for MDPs without coordination.
- Comparison with algorithm that ignores global events to show increased performance of VI heuristic slave which handles global events.
- Achieves a higher quality solution compared to an algorithm that assumes no execution uncertainty.
- Generates approximately the same solution quality as an algorithm that computes a locally optimal joint policy within the slave component.
- Outperforms an algorithm that generates randomized policies, but is less robust to noise.
- Significantly faster than other Dec-MDP solvers that focus on finding a single optimal joint policy.

Although these two algorithms do not always generate the optimal solution, I have shown that they perform far better than many other types of algorithms that can be used to solve these types of Stackelberg Security Games which include defender teamwork. In addition, SMART$_H$ was tested in

the first-of-its-kind large scale real-world experiment that compared my game theoretic schedule against a schedule generated by humans and demonstrated to provide more security and greater effectiveness than the manual schedule.

Chapter 7: Conclusions

The use of game theory in the security domain has seen success in the real world via deployed algorithms and applications such as ARMOR at the Los Angeles International Airport [Pita et al., 2008], IRIS (Intelligent Randomization in Scheduling) for the United States Federal Air Marshal Service [Tsai et al., 2009], GUARDS for TSA [Pita et al., 2011], PROTECT for the US Coast Guard [Shieh et al., 2012], TRUSTS for the Los Angeles Sheriff's Department [Yin et al., 2012], and STREETS, a traffic patrolling application for the Singapore police [Brown et al., 2014b]. However, these systems often have an assumption of a single defender resource or multiple independent defender resources. While this is a reasonable assumption for the first generation of security applications, there is an increased need to address coordination among multiple defender resources.

My thesis aims to handle this challenge first by modeling security games with multiple coordinated defender resources, and then by efficiently solving these types of security games to be able to scale up to real-world scenarios. I further explored the impact of execution uncertainty, both from local events (e.g., a defender resource gets delayed) and global events (e.g., a defender resource stops patrolling), on coordinated defender resources. My thesis makes the following contributions:

7.1 Contributions

- SMART, SMART$_O$, and SMART$_H$: The SMART model extends the framework of security games to represent jointly coordinated activities among the defender resources while the SMART$_O$ algorithm optimally solves SMART game instances and the SMART$_H$ algorithm achieves significant scale-up to be able to solve real-world SMART problems. These algorithms use a branch-and-price framework to deal with the large strategy space of the defender, with SMART$_H$ exploiting the structure of the joint activity coordination problem to gain speed up. In fact, the SMART$_H$ algorithm was used in a mass transit full scale exercise involving 14 heterogeneous teams to compare the benefit of game-theoretic schedules versus human generated schedules.

- Defender coordination under execution uncertainty : To address execution uncertainty in security games with coordinated defender resources, I designed a model and algorithm that integrates the teamwork mechanisms under uncertainty from Dec-MDPs into a security game framework. This model of security games represents the defender pure strategy as a joint policy to address coordination under uncertainty. I provide additional heuristics to both enable scale up in the number of targets in the real world and with respect to the number of defender resources. I explore the robustness of randomized defender pure strategies. This opens the door to a potentially new area of combining computational game theory and multi-agent teamwork.

7.2 Future Plans

My research in this thesis focuses on using a game theoretic approach to optimally allocate multiple coordinated defender resources in the security domain. Additional future research areas include: (i) Scalability: one potential idea is to improve the algorithm by generating a diverse set of defender strategies instead of a single defender strategy; (ii) Team Formation in Security Games: given a set of heterogeneous defender resource types that provide varying levels of effectiveness along with a fixed cost, what is the optimal defender team of heterogeneous resources that gives the greatest benefit to the defender; (iii) Coordinated Adversary Resources: model and compute the defender strategy against multiple adversary resources that are able to coordinate and simultaneous attack multiple targets.

7.2.1 Scalability

My current work has provided models and efficient algorithms to handle coordination and teamwork among multiple defender resources in security games for both domains where there is no execution uncertainty and those where there is execution uncertainty for the defender resources. However, newer algorithms and solution approaches are needed to handle larger domains with greater number of resources. For example, in the context of counter-terrorism, security forces organized a full-scale exercise in which 80 security officers were divided into 23 defender teams [Fave et al., 2014].

New algorithms are needed to improve the efficiency and scalability in solving problems that require a larger number of coordinating defender resources. One way to improve scalability is to explore generating a diverse set of Dec-MDP policies for each iteration, thus potentially reducing

the total number of iterations that are needed to compute the optimal defender strategy. Currently, as part of column generation, each iteration of the slave component generates a single joint policy (which contains a policy for each defender resource). Figure 7.1 shows the steps of column generation, with the first step starting at the master component with a single pure strategy. The second step includes the slave component generating the next best strategy, which is then returned to the master component. In the third step, the master component adds the newly generated pure strategy from the slave to the set of pure strategies. The master component then resolves the security game, and then repeats the steps 1-3. Note that in each iteration, a single pure strategy is generated by the slave component.

A new idea is to have the slave component generate multiple strategies in each iteration. Figure 7.2 shows how the column generation iteration changes and provides additional strategies at each iteration. In step 2, instead of the slave just generating one pure strategy, it generates three new patrol strategies for the master component. This has the capability to both improve the solution quality (defender's expected utility) and reduce the number of times that the slave component needs to be executed, thereby improving the runtime and scalability of the algorithm.

7.2.2 Team Formation in Security Games

In additional to scalability, a different direction for future research is to look at various team types that allow heterogeneous team members (e.g., 3 boats, 2 boats and a helicopter, one boat and 2 helicopters; see Figure 7.3 for some examples of different defender teams) in security games, where there are costs associated with each defender resource (e.g., one boat has the same cost as two helicopters). These heterogeneous resources will also have different levels of effectiveness based on which resources are conducting a joint activity/working together. For example, having

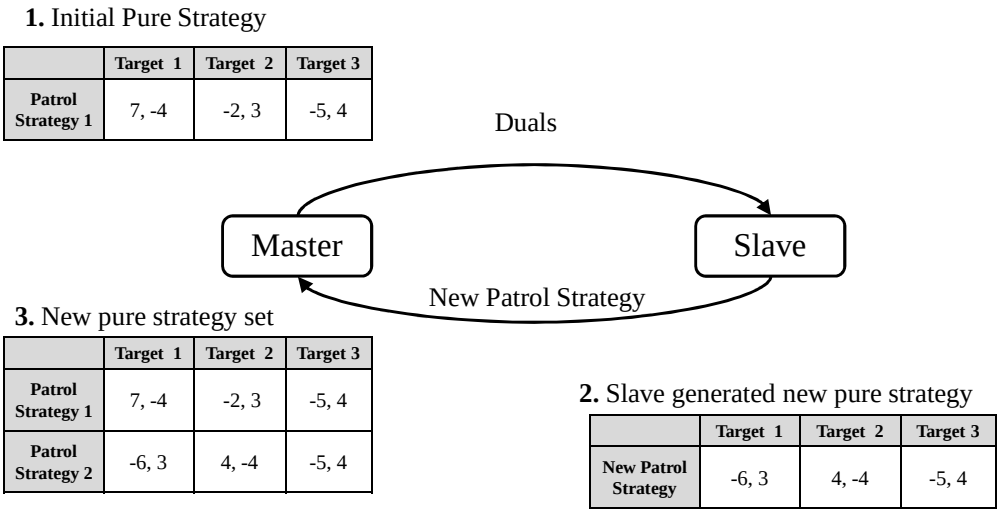


Figure 7.1: Column generation with a single strategy(policy)

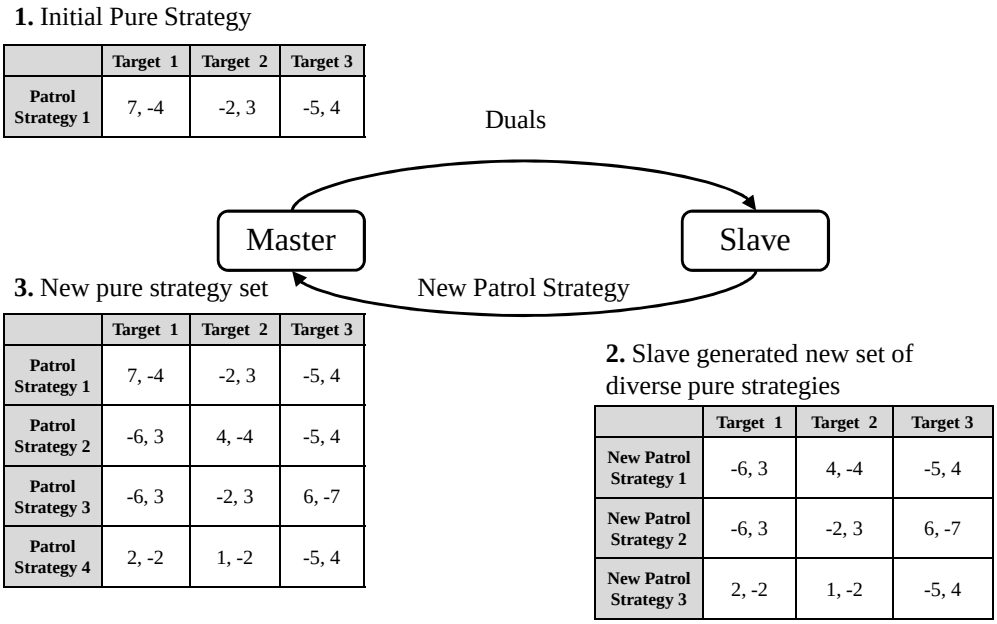


Figure 7.2: Column generation with a diverse set of strategies(policies)

a helicopter and a boat visit the same target will provide increased effectiveness compared to having two helicopters or two boats visiting that target. Then given a set cost, what would be the composition of the team that provides the optimal defender expected utility.

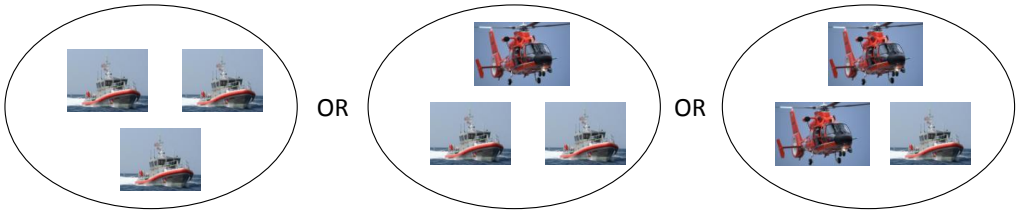


Figure 7.3: Examples of different heterogeneous defender teams

7.2.3 Coordinated Adversary Resources

A common assumption in security games is that there is either a single adversary, or multiple independent adversaries with each target being able to be attacked by a separate adversary. While my research has focused on modeling and computing defender strategies for coordinated defender resources, there has been a lack in focus on studying multiple coordinated adversary resources. This is a relevant area to study as there have been numerous terrorist attacks in the past with multiple coordinated attacks from the terrorist organizations such as the September 11 attacks where multiple planes were used to attack the World Trade Center towers along with the Pentagon, the 2004 Madrid bombings that contained ten explosions on four trains where all the explosions took place within 5 minutes of each other, and the 2008 Mumbai terrorist attacks that was conducted by 10 attackers that split up into different groups to target various buildings along with bomb blasts during the attack. A necessary step is to model these types of coordinated attacks to better compute and generate defender strategies that take these type of attackers into account.

Bibliography

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

N. Agmon, G. A. Kaminka, and S. Kraus. Multi-robot adversarial patrolling: facing a full-knowledge opponent. *Journal of Artificial Intelligence Research (JAIR)*, 42(1):887–916, 2011.

Noa Agmon, Vladimir Sadov, Gal A. Kaminka, and Sarit Kraus. The Impact of Adversarial Knowledge on Adversarial Planning in Perimeter Patrol. In *AAMAS*, volume 1, pages 55–62, 2008b.

Christopher Amato and Frans A Oliehoek. Scalable planning and learning for multiagent pomdps. *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI-15) (to appear)*, 2015.

Christopher Amato, George D Konidaris, and Leslie P Kaelbling. Planning with macro-actions in decentralized pomdps. In *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*, pages 1273–1280. International Foundation for Autonomous Agents and Multiagent Systems, 2014.

C. Barnhart, E.L. Johnson, G.L. Nemhauser, M.W.P. Savelsbergh, and P.H. Vance. Branch and price: Column generation for solving huge integer programs. In *Operations Research*, 1994.

N. Basilico, N. Gatti, and F. Amigoni. Leader-Follower Strategies for Robotic Patrolling in Environments with Arbitrary Topologies. In *AAMAS*, pages 500–503, 2009.

Nicola Basilico, Nicola Gatti, and Francesco Amigoni. Patrolling security games: Definition and algorithms for solving large instances with single patroller and single intruder. *Artificial Intelligence*, 184-185:78–123, 2012.

BBC News Europe. Charlie hebdo attack: Three days of terror. *BBC News Europe*, 2015. Retrieved January 15, 2015 from `http://www.bbc.com/news/world-europe-30708237`.

Raphen Becker, Shlomo Zilberstein, Victor Lesser, and Claudia V Goldman. Solving transition independent decentralized markov decision processes. In *JAIR*, volume 22, pages 423–455, 2004.

Daniel S Bernstein, Robert Givan, Neil Immerman, and Shlomo Zilberstein. The complexity of decentralized control of markov decision processes. *Mathematics of operations research*, 27(4): 819–840, 2002.

Dimitris Bertsimas and John N. Tsitsiklis. *Introduction to Linear Optimization*. Athena Scientific, 1994.

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

Matthew Brown, William B. Haskell, and Milind Tambe. Addressing scalability and robustness in security games with multiple boundedly rational adversaries. In *Conference on Decision and Game Theory for Security (GameSec)*, 2014a.

Matthew Brown, Sandhya Saisubramanian, Pradeep Varakantham, and Milind Tambe. STREETS: Game-theoretic traffic patrolling with exploration and exploitation. In *Innovative applications of Artificial Intelligence (IAAI)*, 2014b.

Chelsea J. Carter. 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.

Vincent Conitzer and Tuomas Sandholm. Computing the optimal strategy to commit to. In *ACM EC-06*, pages 82–90, 2006.

Keith Decker and Victor R Lesser. Designing a family of coordination algorithms. In *ICMAS*, volume 95, pages 73–80, 1995.

Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 1–38, 1977.

Jilles S Dibangoye, Christopher Amato, and Arnoud Doniec. Scaling up decentralized MDPs through heuristic search. In *UAI*, pages 217–226, 2012.

Jilles S. Dibangoye, Christopher Amato, Arnaud Doniec, and François Charpillet. Producing efficient error-bounded solutions for transition independent decentralized mdps. In *Proceedings of the 2013 International Conference on Autonomous Agents and Multi-agent Systems*, pages 539–546, Richland, SC, 2013. International Foundation for Autonomous Agents and Multiagent Systems.

J. P. Dickerson, G. I. Simari, V. S. Subrahmanian, and Sarit Kraus. A graph-theoretic approach to protect static and moving targets from adversaries. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems*, 2010.

Prashant Doshi and Piotr J Gmytrasiewicz. A particle filtering based approach to approximating interactive pomdps. In *AAAI*, pages 969–974, 2005.

Prashant Doshi, Yifeng Zeng, and Qiongyu Chen. Graphical models for online solutions to interactive pomdps. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, page 217. ACM, 2007.

Prashant Doshi, Yifeng Zeng, and Qiongyu Chen. Graphical models for interactive pomdps: representations and solutions. *Autonomous Agents and Multi-Agent Systems*, 18(3):376–416, 2009.

Francesco M. Delle Fave, Eric Shieh, Manish Jain, Albert Xin Jiang, Heather Rosoff, Milind Tambe, and John P. Sullivan. 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.

Nicola Gatti. Game theoretical insights in strategic patrolling: Model and algorithm in normal-form. In *ECAI-08*, pages 403–407, 2008.

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

Claudia V Goldman and Shlomo Zilberstein. Communication-based decomposition mechanisms for decentralized mdps. *Journal of Artificial Intelligence Research (JAIR)*, 32:169–202, 2008.

Claudia V Goldman, Martin Allen, and Shlomo Zilberstein. Learning to communicate in a decentralized environment. *Autonomous Agents and Multi-Agent Systems*, 15(1):47–90, 2007.

Barbara J Grosz and Sarit Kraus. Collaborative plans for complex group action. *Artificial Intelligence*, 1996.

G. Gutin, A. Yeo, and A. Zverovich. Traveling salesman should not be greedy: domination analysis of greedy-type heuristics for the tsp. *Discrete Applied Mathematics*, 117:81–86, 2002.

Christopher Hewitt. *Understanding terrorism in America: from the Klan to al Qaeda*. Psychology Press, 2003.

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.

Institute for the Analysis of Global Security. How much did the september 11 terrorist attack cost america? 2004. Retrieved November 24, 2014 from `http://www.iags.org/costof911.html`.

M. Jain, E. Kardes, C. Kiekintveld, M. Tambe, and F. Ordonez. Security games with arbitrary schedules: A branch and price approach. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pages 792–797, 2010a.

Manish Jain, Erim Kardes, Christopher Kiekintveld, Fernando Ordóñez, and Milind Tambe. Security Games with Arbitrary Schedules: A Branch and Price Approach. In *AAAI*, pages 792–797, 2010b.

Manish Jain, Jason Tsai, James Pita, Christopher Kiekintveld, Shyamsunder Rathi, Milind Tambe, and Fernando Ordóñez. Software assistants for randomized patrol planning for the lax airport police and the federal air marshal service. *Interfaces*, 40(4):267–290, 2010c.

Manish Jain, Dmytro Korzhyk, Ondrej Vanek, Vincent Conitzer, Michal Pechoucek, and Milind Tambe. A double oracle algorithm for zero-sum security games on graphs. In *AAMAS*, volume 1, pages 327–334, 2011.

Michal Jakob, Ondřej Vaněk, Ondřej Hrstka, and Michal Pěchouček. Agents vs. pirates: multi-agent simulation and optimization to fight maritime piracy. In *AAMAS*, volume 1, pages 37–44, 2012.

Albert Xin Jiang, Thanh H. Nguyen, Milind Tambe, and Ariel D. Procaccia. Monotonic maximin: A robust stackelberg solution against boundedly rational followers. In *Conference on Decision and Game Theory for Security (GameSec)*, 2013a.

Albert Xin Jiang, Ariel D. Procaccia, Yundi Qian, Nisarg Shah, and Milind Tambe. Defender (mis)coordination in security games. In *IJCAI*, 2013b.

Albert Xin Jiang, Zhengyu Yin, Chao Zhang, Milind Tambe, and Sarit Kraus. Game-theoretic randomization for security patrolling with dynamic execution uncertainty. In *AAMAS*, volume 1, pages 207–214, 2013c.

Christopher Kiekintveld, Manish Jain, Jason Tsai, James Pita, Milind Tambe, and Fernando Ordóñez. Computing optimal randomized resource allocations for massive security games. In *AAMAS*, volume 1, pages 689–696, 2009.

D. Korzhyk, V. Conitzer, and R. Parr. Security games with multiple attacker resources. In *Proceedings of the Twenty-second International Joint Conference on Artificial Intelligence (IJCAI)*, pages 273–279, 2011a.

D. Korzhyk, V. Conitzer, and R. Parr. Solving stackelberg games with uncertain observability. In *Proceedings of the Tenth International Conference on Agents and Multi-agent Systems (AAMAS)*, pages 1013–1020, 2011b.

Deborah Kotz. Injury toll from marathon bombs reduced to 264. *The Boston Globe*, 2013.

Akshat Kumar and Shlomo Zilberstein. Anytime planning for decentralized pomdps using expectation maximization. *Proceedings of the Twenty-Sixth Conference on Uncertainty in Artificial Intelligence*, pages 294–301, 2010.

Akshat Kumar, Shlomo Zilberstein, and Marc Toussaint. Scalable multiagent planning using probabilistic inference. In *IJCAI*, volume 22, pages 2140–2146, 2011.

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

J. Letchford and V. Conitzer. Solving security games on graphs via marginal probabilities. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pages 591–597, 2013.

J. Letchford and Vorobeychik. Optimal interdiction of attack plans. In *Proceedings of the Twelfth International Conference of Autonomous Agents and Multi-agent Systems (AAMAS).*, pages 199–206, 2013.

J. Letchford, L. MacDermed, V. Conitzer, R. Parr, and C. L. Isbell. Computing optimal strategies to commit to in stochastic games. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 1380–1386, 2012.

Aydano Machado, Geber Ramalho, Jean-Daniel Zucker, and Alexis Drogoul. Multi-agent patrolling: An empirical analysis of alternative architectures. In *MABS*, pages 155–170, 2003.

Laëtitia Matignon, Laurent Jeanpierre, and Abdel-Illah Mouaddib. Coordinated multi-robot exploration under communication constraints using decentralized markov decision processes. In *AAAI*, 2012.

Francisco S Melo and Manuela Veloso. Decentralized MDPs with sparse interactions. *Artificial Intelligence*, 175(11):1757–1789, 2011.

Ranjit Nair, Milind Tambe, Makoto Yokoo, David Pynadath, and Stacy Marsella. Taming decentralized POMDPs: Towards efficient policy computation for multiagent settings. In *IJCAI*, volume 1, pages 705–711, 2003.

Ranjit Nair, Pradeep Varakantham, Milind Tambe, and Makoto Yokoo. Networked distributed POMDPs: A synthesis of distributed constraint optimization and POMDPs. In *AAAI*, volume 1, pages 133–139, 2005.

G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An analysis of approximations for maximizing submodular set functions–I. *Mathematical Programming*, 14(1):265–294, Dec 1978.

Thanh Nguyen, Albert Jiang, and Milind Tambe. Stop the compartmentalization: Unified robust algorithms for handling uncertainties in security games. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2014a.

Thanh H. Nguyen, Amulya Yadav, Bo An, Milind Tambe, and Craig Boutilier. Regret-based optimization and preference elicitation for stackelberg security games with uncertainty. In *In Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 2014b.

Eugene Nudelman, Jennifer Wortman, Yoav Shoham, and Kevin Leyton-Brown. 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*, pages 880–887. IEEE Computer Society, 2004.

Frans A. Oliehoek, Julian F.P. Kooi, and Nikos Vlassis. The cross-entropy method for policy search in decentralized POMDPs. *Informatica*, 32:341–357, 2008.

Frans A Oliehoek, Matthijs TJ Spaan, Christopher Amato, and Shimon Whiteson. Incremental clustering and expansion for faster optimal planning in dec-pomdps. *Journal of Artificial Intelligence Research*, pages 449–509, 2013.

Frans A Oliehoek, Matthijs TJ Spaan, and Stefan J Witwicki. Influence-optimistic local values for multiagent planning. *Proceedings of the Fourteenth International Conference on Autonomous Agents and Multiagent Systems. Extended Abstract. (To appear)*, 2015.

Joni K Pajarinen and Jaakko Peltonen. Periodic finite state controllers for efficient pomdp and dec-pomdp planning. In *Advances in Neural Information Processing Systems*, pages 2636–2644, 2011.

Praveen Paruchuri, Jonathan P. Pearce, Janusz Marecki, Milind Tambe, Fernando Ordóñez, and Sarit Kraus. Playing games with security: An efficient exact algorithm for Bayesian Stackelberg games. In *AAMAS-08*, pages 895–902, 2008.

Marek Petrik and Shlomo Zilberstein. A bilinear programming approach for multiagent planning. *JAIR*, 35(1):235–274, 2009.

James Pita, Manish Jain, Craig Western, Christopher Portway, Milind Tambe, Fernando Ordonez, Sarit Kraus, and Praveen Parachuri. Deployed ARMOR protection: The application of a game-theoretic model for security at the Los Angeles International Airport. In *AAMAS*, volume 1, pages 125–132, 2008.

James Pita, Manish Jain, Fernando Ordonez, Milind Tambe, and Sarit Kraus. Robust solutions to stackelberg games: Addressing bounded rationality and limited observations in human cognition. *Artificial Intelligence Journal, 174(15):1142-1171*, 2010.

James Pita, Milind Tambe, Chris Kiekintveld, Shane Cullen, and Erin Steigerwald. Guards - game theoretic security allocation on a national scale. In *International Conference on Autonomous Agents and Multiagent Systems*, 2011.

James Pita, Richard John, Rajiv Maheswaran, Milind Tambe, and Sarit Kraus. A robust approach to addressing human adversaries in security games. In *ECAI*, 2012.

Press Information Bureau (Government of India). Hm announces measures to enhance security. 2008. Retrieved November 24, 2014 from `http://pib.nic.in/newsite/erelease.aspx?relid=45446`.

Ariel D Procaccia, Sashank J Reddi, and Nisarg Shah. Coordination in security games via admm.

Yundi Qian, William B. Haskell, and Milind Tambe. Robust strategy against unknown risk-averse attackers in security games. In *14th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2015)*, 2015.

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.

Maayan Roth, Reid Simmons, and Manuela Veloso. Exploiting factored representations for decentralized execution in multiagent teams. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, page 72. ACM, 2007.

Eric Shieh, Bo An, Rong Yang, Milind Tambe, Craig Baldwin, Joseph DiRenzo, Ben Maule, and Garrett Meyer. PROTECT: A deployed game theoretic system to protect the ports of the united states. In *AAMAS*, volume 1, pages 13–20, 2012.

Eric Shieh, Manish Jain, Albert Xin Jiang, and Milind Tambe. Efficiently solving joint activity based security games. In *IJCAI*, volume 1, pages 346–352, 2013.

Eric Shieh, Albert Xin Jiang, Amulya Yadav, Pradeep Varakantham, and Milind Tambe. Unleashing dec-mdps in security games: Enabling effective defender teamwork. In *European Conference on Artificial Intelligence (ECAI)*, volume 1, pages 819–824, 2014.

E. Sless, N. Agmon, and S. Kraus. Multi-robot adversarial patrolling: Facing coordinated attacks. In *Proceedings of the Thirteenth International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 1093–1100, 2014.

Ekhlas Sonu and Prashant Doshi. Generalized and bounded policy iteration for finitely-nested interactive pomdps: scaling up. In *AAMAS*, volume 2, pages 1039–1048, 2012.

Matthijs TJ Spaan and Francisco S Melo. Interaction-driven markov games for decentralized multiagent planning under uncertainty. In *AAMAS*, volume 1, pages 525–532, 2008.

Matthijs TJ Spaan and Frans A Oliehoek. 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.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

Milind Tambe. Towards flexible teamwork. In *JAIR*, 1997.

Milind Tambe. *Security and Game Theory: Algorithms, Deployed Systems, Lessons Learned*. Cambridge University Press, 2011.

Jason Tsai, Shyamsunder Rathi, Christopher Kiekintveld, Fernando Ordóñez, and Milind Tambe. IRIS a tool for strategic security allocation in transportation networks. In *AAMAS-09 (Industry Track)*, pages 37–44, 2009.

O. Vanek, B. Bosansky, M. Jakob, and M. Pechoucek. Transiting areas patrolled by a mobile adversary. In *IEEE Symposium on Computational Intelligence and Games (CIG)*, pages 9–16. IEEE, 2010.

Ondrej Vanek, Michal Jakob, Ondrej Hrstka, and Michal Pechoucek. Using multi-agent simulation to improve the security of maritime transit. In *MABS*, 2011.

Ondřej Vaněk, Michal Jakob, Viliam Lisý, Branislav Bošanský, and Michal Pěchouček. Iterative game-theoretic route selection for hostile area transit and patrolling. In *The 10th International Conference on Autonomous Agents and Multiagent Systems*, volume 3, pages 1273–1274, 2011.

Pradeep Varakantham, Junyoung Kwak, Matthew Taylor, Janusz Marecki, Paul Scerri, and Milind Tambe. Exploiting coordination locales in distributed POMDPs via social model shaping. In *ICAPS*, pages 313–320, 2009.

Pradeep Varakantham, Asrar Ahmed, and Shih-Fen Cheng. Decision support for assorted populations in uncertain and congested environments. *In submission to JAIR*, 2013a.

Pradeep Varakantham, Hoong Chuin Lau, and Zhi Yuan. Scalable randomized patrolling for securing rapid transit networks. In *IAAI*, pages 1563–1568, 2013b.

Wikipedia. 2004 madrid train bombings — Wikipedia, the free encyclopedia. 2014. Retrieved November 24, 2014 from `http://en.wikipedia.org/wiki/2004_Madrid_train_bombings`.

Feng Wu, Shlomo Zilberstein, and Nicholas R Jennings. Monte-carlo expectation maximization for decentralized pomdps. In *Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*, pages 397–403. AAAI Press, 2013.

Rong Yang, Milind Tambe, and Fernando Ordonez. Computing optimal strategy against quantal response in security games. In *AAMAS*, volume 2, pages 847–854, 2012.

Rong Yang, Albert Xin Jiang, Milind Tambe, and Fernando Ordonez. Scaling-up security games with boundedly rational adversaries: A cutting-plane approach. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2013.

Rong Yang, Benjamin Ford, Milind Tambe, and Andrew Lemieux. Adaptive resource allocation for wildlife protection against illegal poachers. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, volume 1, pages 453–460, 2014.

Zhengyu Yin and Milind Tambe. A unified method for handling discrete and continuous uncertainty in bayesian stackelberg games. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2012.

Zhengyu Yin, Dmytro Korzhyk, Christopher Kiekintveld, Vincent Conitzer, and Milind Tambe. Stackelberg vs. Nash in Security Games: Interchangeability, Equivalence, and Uniqueness. In *AAMAS*, volume 1, pages 1139–1146, 2010.

Zhengyu Yin, Manish Jain, Milind Tambe, and Fernando Ordonez. Risk-averse strategies for security games with execution and observational uncertainty. In *Conference on Artificial Intelligence (AAAI)*, 2011.

Zhengyu Yin, Albert Jiang, Matthew Johnson, Milind Tambe, Christopher Kiekintveld, Kevin Leyton-Brown, Tuomas Sandholm, and John Sullivan. Trusts: Scheduling randomized patrols for fare inspection in transit systems. In *Conference on Innovative Applications of Artificial Intelligence (IAAI)*, 2012.

Appendix A

This appendix presents two tables:

- Table 7.1 depicts the security allocation resulting from the manual allocation process
- Table 7.2 depicts the security allocation resulting from the game-theoretic allocation process based on the SMART$_H$ algorithm.

	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}
15 *min*		*VIPR*			T_{510}	T_{510}	T_{49}	T_{38}		T_{16}
30 *min*			*VIPR*					T_{38}		T_{16}
45 *min*			*VIPR*	$\frac{VIPR}{CRM_3}$				$\frac{EK9_2}{CRM_1}$		$\frac{EK9_1}{CRM_2}$
1 *Hour*										
15 *min*				*VIPR*		$\frac{EK9_1}{CRM_2}$				
30 *min*		CRM_3					$\frac{EK9_1}{CRM_1}$		$\frac{EK9_2}{CRM_2}$	
45 *min*	CRM_3		T_{38}	T_{49}	$HVWT_{12}$					
2 *Hours*					$HVWT_{12}$					
15 *min*										
30 *min*	T_{11}						*VIPR*		T_{27}	
45 *min*	T_{11}					*VIPR*			T_{27}	
3 *Hours*		*VIPR*								

Table 7.1: The human-generated security allocation

	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}
15 *min*	$T_{510}(s)$		$HVWT_{12}$(p)	$T_{38}(m)$			$T_{16}(m)$			
30 *min*	$\frac{T_{11}(m)}{VIPR(p_1)}$								$T_{27}(p)$	$T_{510}(p)$
45 *min*	$T_{11}(p_2)$			$HVWT_{12}(p)$	$T_{38}(p)$					
1 *Hour*										
15 *min*			$CRM_2(s)$			$VIPR(p)$		$T_{16}(p)$	$T_{49}(p)$	
30 *min*		$HVWT_{34}(p)$				$\frac{CRM_1}{EK9_1}(s)$	$VIPR(p)$	$T_{27}(m)$		$T_{49}(m)$
45 *min*				$CRM_3(s)$				$\frac{CRM_1}{EK9_1}(s)$		
2 *Hours*										
15 *min*						$HVWT_{34}(m)$		$VIPR(p)$		$\frac{CRM_2}{EK9_2}(s)$
30 *min*		$CRM_3(s)$	$T_{49}(m)$						$\frac{CRM_1}{EK9_1}(s)$	
45 *min*						$\frac{CRM_2}{EK9_2}(s)$			$VIPR(m)$	
3 *Hours*					$CRM_3(s)$					

Table 7.2: Security allocation generated by SMART$_H$: s represents the street level of a station, m the mezzanine level and p the platform level.